BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 73(2), 2025, Article number: e153226
DOI: 10.24425/bpasts.2025.153226

CONTROL, INFORMATICS AND ROBOTICS

Importance of C/C++ compiler choice for performance and energy consumption of multithreaded WZ factorization

Beata BYLINA, Monika PIEKARZ*, and Jarosław BYLINA

Maria Curie-Sklodowska University, Pl. M. Curie-Skłodowskiej 5, 20-031 Lublin, Poland

Abstract. The choice of C/C++ compiler significantly impacts the performance and energy consumption of multithreaded numerical algorithms related to linear algebra. This study investigates the effects of the C/C++ compiler choice and processor frequency scaling (using dynamic voltage frequency scaling) on the performance and energy consumption of the multithreaded WZ factorization on three different computing platforms, two featuring Intel Xeon processors and one featuring AMD EPYC processor. The factorization is implemented both without optimization techniques and with strip-mining. Based on time and energy tests, we have demonstrated that, for the WZ factorization (in both implementations), each compiler reacts somewhat differently to frequency changes, thus affecting overall performance and energy consumption. The Intel compilers achieved the best performance and energy savings in a multithreaded environment compared to the other compilers on each of the tested computing platforms.

Keywords: processor frequency scaling; performance; energy; WZ factorization; compiler.

1. INTRODUCTION

The choice of a C/C++ compiler and its associated configurations holds a pivotal sway over the performance and energy consumption characteristics of multithreaded numerical algorithms, particularly those in the domain of linear algebra. A compiler with a robust support for multithreading can produce the code that effectively exploits parallelism, thereby enhancing performance and reducing energy consumption on multicore systems. Researchers and practitioners confront the imperative task of meticulously evaluating and selecting compilers that align precisely with the characteristics and requirements of the targeted multi-core architecture. This judicious selection emerges as a significant contributing factor in achieving the desired equilibrium between high-performance computing and energy efficiency. This judicious selection emerges as a significant contributing factor in achieving the desired balance between high-performance computing and energy efficiency, which is a crucial aspect of complex systems informatiks. As highlighted in the article [1], energy efficiency and the proper choice of algorithms play a fundamental role in the functioning of parallel systems.

In today's computing environment, which is characterized by the proliferation of multi-core processors, the limitations of conventional programming languages are evident. As a result, there is a growing need for specialized frameworks and extensions dedicated to high-performance computing (HPC) to bridge this gap. Frameworks like OpenMP, utilized in the research under discussion, empower programmers to incorporate parallelism into their codebases. Consequently, it is essential for compilers to work in tandem with runtime libraries to effectively translate parallel code into the complexities of processor architecture. Many HPC applications heavily rely on such extensions, continuously evolving through collaborative efforts between creators of optimized computational algorithms, such as those in the article [2], developers and hardware vendors. Thus, compilers must remain updated with the evolving standards for language extensions to ensure smooth adaptation to the ever-changing landscape of parallel computing paradigms.

This article serves as a natural extension of our previous work [3]. Previous study explored the impact of processor frequency scaling using dynamic voltage frequency scaling (DVFS) on the performance and energy consumption of the WZ factorization, concentrating exclusively on the Intel C++ compiler. The conclusion from our tests was that the highest frequency is not always the best in terms of time and energy consumption. For the WZ factorization algorithm, it pays to reduce the frequency to save energy without losing performance. Because the choice of compiler may prove to be a significant factor in achieving the desired equilibrium between high-performance computing and energy efficiency in this extended investigation, our scope intentionally broadens to encompass compilers adhering to the OpenMP standard, specifically GCC (GNU Compiler Collection), and two versions of the Intel compiler – ICC and its latest iteration, OneAPI. OpenMP support was systematically enabled for each compiler during the compilation process using the appropriate flags, resulting in the creation of a multithreaded implementation on the CPU. Additionally, we extended our research to three platforms: two computing platforms with Intel Xeon processors – Intel Xeon Gold like in [3] and Intel Xeon

*e-mail: monika.piekarz@mail.umcs.pl

Manuscript submitted 2024-04-29, revised 2024-10-01, initially accepted for publication 2024-12-17, published in March 2025.

Platinum—and one platform with an AMD EPYC processor. This selection allows us to evaluate the impact of compilers on the performance of the WZ algorithm code across a broad spectrum of architectures, providing insights into how different hardware setups influence the optimization results.

The WZ matrix factorization, also known as quadrant interlocking factorization (QIF), made its debut in 1979 courtesy of Evans and Hatzopoulos [4]. Their primary objective was to design a factorization method with superior parallelization potential compared to the well-established LU factorization. The distinctive feature of WZ factorization lies in its simultaneous zeroing of two columns/rows, contrasting with the LU factorization, which zeros only one column/row. Since its introduction, WZ factorization has garnered attention from various researchers, as evidenced by the works of [5–10]. These studies contribute to the understanding of the applications of WZ factorization in solving linear systems. As highlighted in the article [11], numerical algebra methods are crucial for optimizing computations, particularly in robotics.

The WZ factorization algorithm is inherently complex and particularly well-suited for studying the impact of compiler optimizations. Its inherent challenges, such as avoiding issues like over-synchronization and memory bottlenecks, make it a rigorous test case for compilers. This allows for focused research on maximizing the effectiveness of various optimizations, including multithreading and frequency scaling, across different architectures.

Publication [8] presented a detailed implementation of multithreaded WZ factorization using OpenMP on a multicore architecture, incorporating various nested loop transformation strategies to optimize the program. The implementation of an algorithm plays a crucial role in determining its performance. Hence, this article examines two distinct multithreaded implementations of the WZ row algorithm. One implementation serves as the baseline, while the other incorporates a loop optimization technique and employs strip-mining. Through this comparative analysis, we aim to explore how implementation choices, beyond compilation, significantly impact algorithmic performance.

Our current research utilized the GCC, Intel ICC, and Intel OneAPI compilers, along with the Intel Xeon Gold, Intel Xeon Platinum, and AMD EPYC platforms. GCC is an open-source compiler known for its broad support across many architectures, while Intel ICC and Intel OneAPI are optimized for Intel processors, enabling full use of advanced features like AVX-512. Intel Xeon Gold and Platinum were chosen due to differences in core count and cache, allowing for performance comparison across varying levels of computing power. AMD EPYC, with its high core count, was included to assess the algorithm on a platform with significant parallel processing potential.

Energy savings can be achieved through both hardware [12] and software approaches [13–19]. The former involves innovations in computer hardware, encompassing microarchitecture advancements and integrated circuit design. On the software front, energy optimization operates at both the operating system and application levels. In this article, we concentrate on a hybrid approach, combining dynamic voltage and frequency scaling (DVFS) at the operating system level with C/C++ compiler selection for the program algorithm at both levels (hardware and software).

The main contributions of this paper are the following.

- Evaluation of multithreaded implementations: We conducted thorough testing and evaluation of two multithreaded implementations (basic and strip-mining) of the WZ factorization on multicore CPUs. This investigation spans the utilization of three compilers—GCC, Intel Compiler ICC, and its latest iteration, OneAPI. Additionally, it is conducted on three different computing platforms: Intel Xeon Gold, Intel Xeon Platinum, and AMD EPYC.
- Compiler sensitivity to frequency changes: Our analysis reveals nuanced variations in the reactions of each compiler to frequency changes. These differences manifest in discernible impacts on overall performance and energy consumption during the execution of the WZ factorization algorithm.

The rest of the paper is organized as follows. Section 2 presents a literature review on the impact of C/C++ compilers on performance and energy consumption. Section 3 describes the WZ factorization algorithm using two versions of OpenMP programming models. Section 4 presents the methodology we used in our research. It presents the computational platforms, compilers, the DVFS technique and the RAPL interface we used to perform time and energy measurements. Sections 5 and 6 present a numerical experimental evaluation of the impact of the choice of C/C++ compiler and processor frequency scaling on the performance and energy consumption of WZ factorization on multicore architectures. Finally, Section 7 concludes the paper.

2. RELATED WORK

When examining the impact of C/C++ compiler selection on the performance and energy consumption of multithreaded algorithms, various approaches are discussed in the existing literature. The research focused on energy efficiency in multithreaded numerical algorithms, as highlighted in the works of [13, 16, 18, 20], provides insights into compiler considerations and runtime systems aiming to minimize energy consumption and optimize computational performance for numerical computing. The studies conducted by [20] and [18] involve an analysis of performance and energy consumption per CPU for real-world scientific codes related to the solidification modeling application. These applications utilize the phase-field (PF) method and a generalized finite difference scheme for solving governing partial differential equations (PDEs). Specifically, the paper [20] focuses on various C/C++ compilers tailored for AMD EPYC processors in the context of numerical modeling of solidification. Moreover, the works presented in [13,16] delve into the examination of performance and energy consumption across four OpenMP runtime systems on a non-uniform memory access (NUMA) platform. Those papers present experimental studies characterizing OpenMP runtime systems for three factorizations: Cholesky, LU, and QR. The goal is to gain a deeper understanding of the behavior of these runtime systems in various multithreaded scenarios. The works [3] and [10] focus on the utilization of a specific multi-core algorithm, WZ factorization, to investigate performance and energy consumption. These studies employ the Intel compiler, specifically ICC. The publication of such research allows for concluding in the context of specific challenges related to numerical linear algebra algorithms. This article serves as an extension of the content presented in article [3], delving into the role of C/C++ compilers in influencing the performance, computational efficiency, and energy aspects of WZ factorization.

3. WZ FACTORIZATION

We present shortly the WZ factorization [4]. We transform a square and nonsingular matrix **A** into a product of two matrices, namely **WZ**. The matrix **W** is a matrix of the form of a butterfly with units on its main diagonal, the matrix **Z** is a matrix of the form of an hourglass. Both the matrices are complements of each other in the sense of the structure of nontrivial elements (one has nontrivial elements in places where the other has zeros/units – and vice versa). The forms of these matrices can be seen in Fig. 1.

We chose this numerical algorithm here because it is quite complicated and difficult to optimize by the compiler. The WZ factorization has been parallelized and vectorized using OpenMP pragmas.

Figure 2 presents a basic algorithm for the WZ factorization for an even size of the matrix (we only consider even sizes – without loss of generality).

Considering performance and energy consumption, it is important to have optimized algorithms and their implementation. A general technique for improving performance is to take full advantage of multicore architecture features. A good example is the use of loop optimization in the code as the most common critical places are just the loops. One of the known loop optimization techniques is strip-mining. A loop in the process of strip-mining is divided into two loops, where the inner one has `BLOCK_SIZE` iterations and the outer one has `n/BLOCK_SIZE` iterations (`n` being the number of iterations in the original loop). The strip-mining alone can have some positive impact on the performance (by easing the automatic vectorization process).

```
for(k = 0; k < n/2-1; k++)
{
  p = n-k-1;
  akk = a[k][k];
  akp = a[k][p];
  apk = a[p][k];
  app = a[p][p];
  detinv = 1 / (apk*akp - akk*app);
  #pragma omp parallel for
  for(i = k+1; i < p; i++)
  {
    w[i][k] = (apk*a[i][p] - app*a[i][k])
              * detinv;
    w[i][p] = (akp*a[i][k] - akk*a[i][p])
              * detinv;
    #pragma omp simd
    for(j = k+1; j < p; j++)
      a[i][j] += - w[i][k]*a[k][j]
                 - w[i][p]*a[p][j];
  }
}
```

Fig. 2. The basic algorithm for the multithreaded WZ factorization – pseudocode

In Fig. 3, we present a strip-mining algorithm for the WZ factorization with the parameter of this algorithm, namely `n/BLOCK_SIZE`. We use the compiler clause `__assume` which tells the compiler that a given condition is fulfilled – here, we declare that `ii` and `jj` are multiples of the `BLOCK_SIZE`.

The number of floating-point operations for the WZ factorization algorithm in both versions (basic and optimized for strip mining technique) is the same and equals $\frac{2}{3}n^3 + O(n^2)$ [9]. However, the algorithm (in both implementations) is rather memory-bound than compute-bound – that is, the amount of computations is relatively small compared to the amount of reads from and writes to memory. Namely in Figs. 2 and 3, we can see that in the inner-most loop (which has the most iterations), there are 5 memory reads/writes for every 4 floating-point operations. It is less cache-friendly and can impact both the speedup and energy savings.

$$\begin{aligned}
&\mathbf{W}_{1*} = (1, \underbrace{0,\dots,0}_{n-1}) \\
&\mathbf{W}_{i*} = (w_{i1},\dots,w_{i,i-1},1,\underbrace{0,\dots,0}_{n-2i+1},w_{i,n-i+2},\dots,w_{in}) && \text{for } i = 2,\dots,\frac{n}{2}, \\
&\mathbf{W}_{i*} = (w_{i1},\dots,w_{i,n-i},\underbrace{0,\dots,0}_{2i-n-1},1,w_{i,i+1},\dots,w_{in}) && \text{for } i = \frac{n}{2}+1,\dots,n-1, \\
&\mathbf{W}_{n*} = (\underbrace{0,\dots,0}_{n-1},1) \\
&\mathbf{Z}_{i*} = (\underbrace{0,\dots,0}_{i-1},z_{ii},\dots,z_{i,n-i+1},0,\dots,0) && \text{for } i = 1,\dots,\frac{n}{2}, \\
&\mathbf{Z}_{i*} = (\underbrace{0,\dots,0}_{n-i},z_{i,n-i+1},\dots,z_{ii},0,\dots,0) && \text{for } i = \frac{n}{2}+1,\dots,n.
\end{aligned}$$

Fig. 1. The output of the WZ factorization – rows of the matrices **W** and **Z**

```
for(k = 0; k < n/2-1; k++)
{
  p = n-k-1;
  akk = a[k][k];
  akp = a[k][p];
  apk = a[p][k];
  app = a[p][p];
  detinv = 1 / (apk*akp - akk*app);
  #pragma omp parallel for
  for(i = k+1; i < p; i++)
  {
    w[i][k] = (apk*a[i][p] - app*a[i][k])
              * detinv;
    w[i][p] = (akp*a[i][k] - akk*a[i][p])
              * detinv;
    start = RDTTNM(k+1, BLOCK_SIZE);
    for(jj = start; jj < p; jj += BLOCK_SIZE)
    {
      __assume(jj % BLOCK_SIZE == 0);
      #pragma omp simd
      for(j = jj; j < jj+BLOCK_SIZE; ++j)
        a[i][j] += - w[i][k]*a[k][j]
                   - w[i][p]*a[p][j];
    }
  }
}
```

Fig. 3. Strip-mining in the basic algorithm – pseudocode

4. METHODOLOGY

Similarly to [3], we consider two variants of the WZ factorization algorithm: the basic version and block algorithms employing strip-mining. The dataset for our assessment comprises a randomly generated square matrix containing $n \times n$ double-precision values, where n equals 32 768. Consequently, our test dataset encompasses 1 073 741 824 cells, equivalent to 8 GB of data. All algorithm versions adhere to a row-wise layout and are coded in C++, with vectorization and parallel processing.

Our experimental setup comprises three computing platforms: two featuring Intel Xeon processors and one featuring an AMD EPYC processor. One platform, identical to that used in [3], is equipped with a modern Intel Xeon Gold multi-core processor. The second platform utilizes an Intel Xeon Platinum processor. The third platform is equipped with a contemporary AMD EPYC multi-core processor. Detailed information about our computing platforms is presented in Table 1. It collects information about the clock frequency, number of cores, cache size, and thermal design power (TDP) of the Intel Xeon Gold, Intel Xeon Platinum, and AMD EPYC processors. These details are crucial because they affect the performance and energy consumption of the WZ factoring algorithm. Differences in the number of cores and clock speed affect, for example, parallel processing capabilities, while TDP affects energy efficiency in different workloads, which is crucial for focusing the study on performance and energy consumption.

Intel Xeon Platinum offers higher performance, more cores, better memory support, and more advanced features compared to Intel Xeon Gold. This makes Platinum more efficient for parallel computations. The higher core count allows for better load balancing, especially for larger problems, which reduces processing time. EPYC processors have a very large core count and are optimized for high parallelism. AMD EPYC 9654, with its more cores and higher maximum frequency (up to 3.7 GHz), performs better for computations that can effectively use a large number of cores.

Due to their higher TDP (thermal design power; although both manufacturers understand this concept slightly differently), Intel Xeon Platinum and AMD EPYC processors can use more power than Intel Xeon Gold processors. This is a result of their increased performance capabilities, larger number of cores, and more advanced architectural features, which require more energy to operate at peak efficiency. The higher TDP indicates that these processors are designed to handle greater workloads but at the cost of increased power consumption. Choosing between them depends on balancing your performance needs with energy efficiency, as higher TDP processors are better suited for more demanding tasks but may result in higher operational costs due to increased power usage.

We have many C++ code compilers available for Intel Xeon and AMD EPYC processors. The role of the compiler is a key element in the effective use of the hardware potential of the system on which specific software runs. An effective compiler should allow programmers to concentrate on building the code rather than worrying about the limitations of the compiler. Its ability to generate an optimal binary should cover even the most abstract high-level code. Unfortunately, finding a compiler that meets these criteria is often a challenge. Not all compilers are able to produce optimal code. In some cases, they can generate different sets of low-level instructions for the same piece of high-level code.

For our tests we chose three of available for our processor compilers, namely: GCC, Intel ICC and Intel OneAPI.

In this research, we select compiler options so that different compilers can be compared with corresponding options. Therefore, we give up various types of manual optimizations through detailed options, because they can be different for different compilers and work differently (like the ICC compiler options used in [3]: `-ipo -no-prec-div -fp-model fast-2`).

Table 1
Specification of computing platforms

Processor	Clock frequency	Cores	Cache	TDP
Intel Xeon Gold 5218R	800 MHz – 2.1 GHz	2 x 20 cores	L1i: 32KB, L1d: 32KB, L2: 1024KB, L3: 28MB	125 W
Intel Xeon Platinum 8358	800 MHz – 2.6 GHz	2 x 32 cores	L1i: 32KB, L1d: 48KB, L2: 1280KB, L3: 48MB	270 W
AMD EPYC 9654	400 MHz – 3.7 GHz	96 cores	L1i: 32KB, L1d: 32KB, L2: 1024KB, L3: 32MB	320 W

The following software was used during the tests along with the following compiler options:

- operating system: CentOS 7.5
- kernel: Linux 3.10.0
- GCC/G++ compiler v. 13.1.1 with the following compiler options:
  `-fopenmp -O3`
- Intel ICC compiler v. 2021.5.0 with the following compiler options:
  `-qoenmp -O3`
- Intel OneAPI DPC++/C++ compiler v. 2022.0.0 with the following compiler options:
  `-qoenmp -O3`

The G++ compiler (GCC), part of the GNU Compiler Collection and developed by the Free Software Foundation [21], is an open-source compiler known for its capability to produce binaries for diverse target architectures. It is extensively accessible on Unix-like operating systems. The G++ compiler utilized in this project offers default backing for the C++17 standard and features support for up to the C++23 standard. Additionally, it incorporates the OpenMP 5.0 standard.

The Intel C++ compiler (ICC) was crafted by the Intel Corporation, specializing in optimization for Intel processor architectures. It is tailored to accommodate the latest generation of Intel processors, encompassing support for C++2a and preceding standards. In the context of this paper, the Intel C++ compiler employed embraces the entirety of the OpenMP 4.5 standard, along with a subset of functionalities from OpenMP 5.0 [22]. This compiler is open-source.

The Intel OneAPI DPC++/C++ Compiler [23] is the specific compiler provided by Intel as part of the OneAPI platform. This compiler supports DPC++ and standard C++. It is optimized to support heterogeneous platforms, which means it allows programming on different types of processors such as CPU, GPU, FPGA and others. The compiler used in this article supports C++2a and preceding standards and covers the entire OpenMP 4.5 standard along with a subset of the functionality from OpenMP 5.0. This compiler is open-source.

Intel ICC and OneAPI typically offer better results on Intel processors because they are optimized to take advantage of Intel-specific architecture features such as AVX-512, advanced cache management, and dynamically adjusting execution parameters to the processor architecture. The -O3 option allows for more aggressive optimization, which further improves performance. They may not be as well optimized for AMD architectures. GCC may perform better on AMD processors than Intel compilers because it is more broadly optimized for different architectures and can work better with AMD-specific instruction sets such as AVX2, which are standard on AMD architectures.

We used the RAPL (Running Average Power Limit) interface [24] to measure the power and energy consumption of CPU-level components. We access RAPL energy meters via machine-specific registers (MSR). Counters are 32-bit registers that indicate the amount of energy used since the processor was started, they are updated approximately once every 1 ms (or 1000 Hz). Since its introduction, RAPL has been widely used in energy measurement and modeling. The results presented in the work [24] suggest that RAPL can be a very useful tool for measuring and monitoring energy consumption on multicore computers without the need to implement complicated power meters. The experience of the authors of the works [10, 18, 25] with RAPL confirms the results from the literature. RAPL is able to measure the energy consumption of a complex scientific application with acceptable accuracy and detail.

We carry out 5 iterations of each version of the algorithm for each tested frequency, and then average the results to obtain a statistically correct result. As the results from [10] show, HT does not provide any speedup benefits for the tested versions of the WZ factorization algorithm. During the tests, we use all hardware in terms of the number of processors and test for the number of threads of 40 on Intel Xeon Gold, 64 on Intel Xeon Platinum, and 96 on AMD EPYC, respectively, without HT. In this paper, we only consider the energy consumed by the processor, we ignore the energy consumed by memory because it is small and does not change significantly.

Using the technique of dynamic voltage frequency scaling, we adjusted the clock frequencies through CPU frequency scaling. By default, the `intel_pstate` driver is used to control the performance of processors on GNU/Linux systems. In our case, we did not obtain a satisfactory clock frequency forcing effect and we used the `acpi_cpufreq` driver. By default, the `acpi_cpufreq` driver follows governor `conservative`, which increases or decreases the clock frequency depending on the load on the core by selecting one of several available frequencies from the minimum to the maximum supported by the processor. Each core is independently adjustable. Using the `cpupower` program, we changed the minimum and maximum values of the processor frequency limit at a given level. The frequency setting has been made for all cores of the whole machine. We used the commands:

```
cpupower frequency-set -d 1400000
cpupower frequency-set -u 1400000
```

for setting the minimum and maximum frequency limits at 1.4 GHz. Using this setting means that the clock speed on all cores does not exceed 1.4 GHz, although this setting may result in a drop in clock speeds depending on the load on the cores.

We conducted our tests on each platform for 8 selected frequencies from the minimum to the maximum supported by the processors, making changes as evenly as possible and taking into account the limitations of the allowable frequency settings for each processor. On Intel Xeon platforms, only some clock frequencies can be set, the AMD EPYC platform is much more flexible in this respect, enabling an almost smooth change of the clock frequency. For Intel Xeon Gold the frequencies for which we performed the tests were: 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0 and 2.1, for Intel Xeon Platinum they were: 0.8, 1.1, 1.4, 1.6, 1.8, 2.0, 2.3 and 2.6, and for AMD EPYC they were: 0.4, 0.9, 1.4, 1.9, 2.4, 2.9, 3.4 and 3.7.

5. THE PERFORMANCE AND ENERGY CONSUMPTION FOR `BASIC` WZ FACTORIZATION ALGORITHM

Similarly to [3], we start by measuring the runtime of the `basic` version of the WZ factorization algorithm for different clock frequencies. We perform tests for three compilers (ICC, GCC and OneAPI) and three different platforms like in Table 1. The test results are shown in Fig. 4.

As expected, the results presented in Fig. 4 indicate that the Intel Xeon Platinum platform provides better overall performance than the Intel Xeon Gold platform, with the greatest time and energy savings achieved on the AMD EPYC platform.

Analysis of the impact of compiler and clock frequency on performance (Fig. 4) revealed irregular variations in execution times for all three compilers when changing the frequency. The largest variations were observed on the Intel Xeon Gold platform, especially for the GCC compiler (20% increase in execution time when reducing the frequency from 2.1 GHz to 0.8 GHz). For the Intel Xeon Platinum and AMD EPYC platforms, these differences were much smaller (maximum 8% and 2%, respectively).

In terms of execution time, the GCC compiler performed worse than ICC and OneAPI on both Intel Xeon platforms, especially at lower frequencies. These differences for Intel Xeon

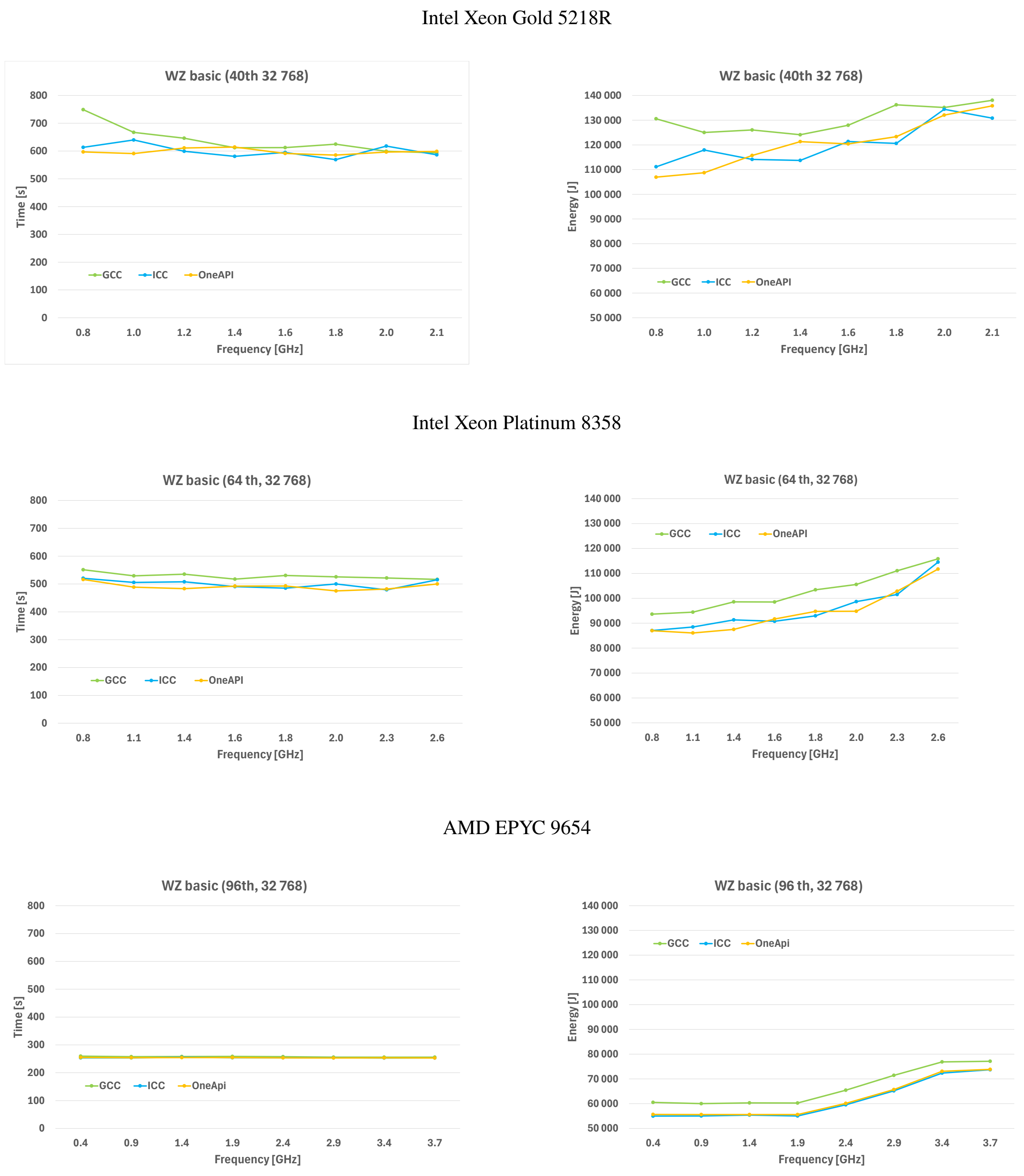


Fig. 4. Runtime and energy consumption of `basic` for data size 32 768 on three computing platforms

Gold reached 18% and 20%, respectively and about 6% for Intel Xeon Platinum at 0.8 GHz. On the AMD EPYC platform, the differences between the compilers were minor.

The energy consumption analysis showed that the GCC compiler was the least energy efficient on all platforms, with differences of up to 18% on 0.8 GHz clock frequency to the detriment of GCC on the Intel Xeon Gold. For the Intel Xeon Platinum and AMD EPYC platforms, these differences were 11% on 1.8 GHz clock frequency and 9% on 0.8 GHz clock frequency, respectively. Furthermore, the increase in clock frequency led to an increase in energy consumption on all platforms, regardless of the compiler used.

The energy consumption profile depending on the clock frequency shows significant differences between the Intel and AMD platforms. While on the Intel platforms, the increase in clock frequency translates linearly into an increase in energy consumption which is consistent with typical CPU power consumption patterns, where higher frequencies require more power, on the AMD platform we observe an unusual phenomenon. Energy consumption remains at a constant, minimum level for frequencies from 0.4 to 1.9 GHz, and increases only occur above this value to a frequency of 3.4 GHz after which it stays at the same high level. We observe these results regardless of the compiler used. The AMD EPYC platform stable power consumption at lower frequencies may be due to its ability to manage power more efficiently within the cores.

Table 2 presents the optimal clock frequency settings for the tested algorithm and indicates the most efficient and energy-efficient compiler for this settings. The best results in terms of both performance and energy efficiency on all platforms were achieved by Intel compilers. On both Intel Xeon platforms, OneApi performed better, while for AMD EPYC platform, a better result was achieved using the ICC compiler. The last two columns of Table 2 describe the percentage time loss and energy gain relative to the change observed when reducing the clock frequency from the maximum for a given platform to the value at which we observed the best energy result on each platform, i.e. 0.8 GHz for the Intel Xeon platforms and 1.9 GHz for the AMD EPYC platform, respectively.

6. THE PERFORMANCE AND ENERGY CONSUMPTION FOR `BASIC-SM` VERSIONS OF WZ FACTORIZATION

In this part, similarly to [3], we will present test results for block versions of the WZ factorization algorithm with strip-mining (abbreviated `sm`). We consider three block sizes: 128, 256, 512, so we have the following versions: `basic-sm-128`, `basic-sm-256`, `basic-sm-512`. We omitted the tests of block 64, which performed the worst in the tests, the results of which are presented in [3]. Our goal is to answer the question how optimization of `sm` and additional clock frequency scaling for the `sm` version affect performance and energy consumption depending on the compiler used.

6.1. Analysis across platforms and block sizes

Intel Xeon and AMD EPYC processors differ in terms of cache configuration (L1, L2, L3) and the number of cores, which affects how data is stored and processed. Different block sizes can optimize the use of these resources to varying degrees. In this section, we examine which block size performs better on each of the computing platforms.

Figure 5 presents the results for all three platforms and various block sizes using the ICC compiler. The results for the remaining compilers confirm the general trends observed in Fig. 5 for each platform, with a detailed comparison between the compilers to be discussed in the next section.

Figure 5 shows that the Intel Xeon Platinum platform generally provides better performance than the Intel Xeon Gold, while the greatest time and energy savings were achieved on the

Table 2
Energy efficiency for `basic` (32768)

Compiler	Runtime [s]	Total energy [J]	Performance [Gflops/s]	Energy efficiency [Gflops/J]	Time loss	Energy gain
Intel Xeon Gold 5218R for frequency 0.8 GHz						
GCC	748.93	130 569.40	31.32	0.180	20.8%	5.4%
ICC	613.19	111 111.65	38.25	0.211	4.4%	15.1%
OneAPI	**597.08**	**106 933.86**	**39.28**	**0.219**	**–0.3%**	**21.3%**
Intel Xeon Platinum 8358 for frequency 0.8 GHz						
GCC	551.30	93 636.21	42.55	0.251	6.3%	19.2%
ICC	520.62	87 052.95	45.05	0.269	1.0%	24.0%
OneAPI	**516.31**	**86 985.09**	**45.43**	**0.270**	**3.2%**	**22.1%**
AMD EPYC 9654 for frequency 1.9 GHz						
GCC	256.62	60 275.65	90.70	0.389	1.0%	21.9%
ICC	**253.81**	**55 021.94**	**92.42**	**0.426**	**0.2%**	**25.4%**
OneAPI	254.43	55 597.97	92.19	0.422	0.5%	24.7%

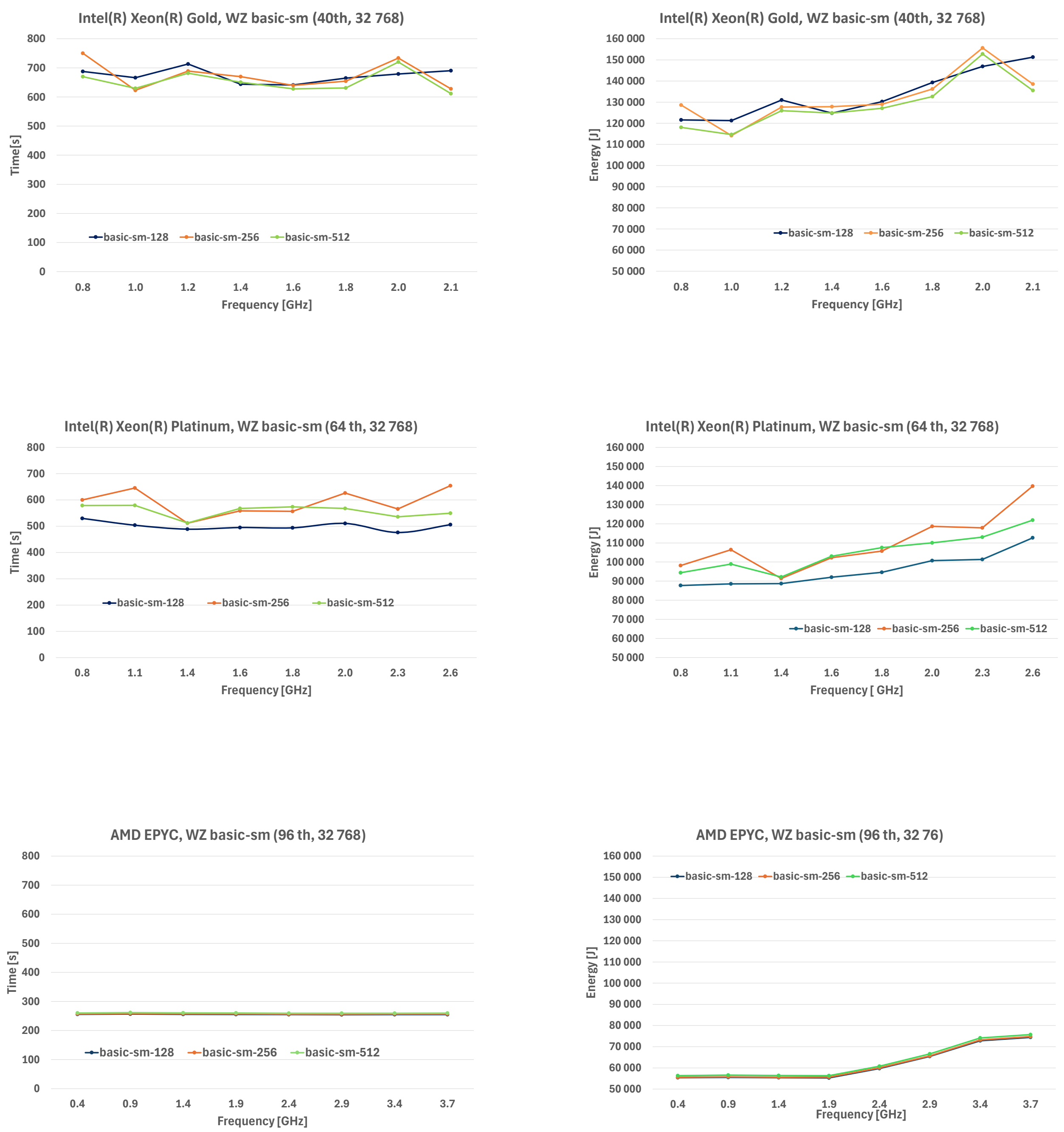


Fig. 5. Runtime and energy consumption of `basic-sm` for ICC compiler on three computing platforms

AMD EPYC platform, consistent with earlier observations for the basic version of the WZ factorization algorithm.

As before, we observe fluctuations in execution times with changes in clock frequency, which are more pronounced on the Intel Xeon platforms and significantly smaller on the AMD EPYC platform. The energy consumption profile with varying clock frequency also confirms previous observations—on the Intel Xeon platforms, there is a regular increase in energy consumption, whereas on the AMD EPYC platform, it remains stable at a minimal level for frequencies from 0.4 GHz to 1.9 GHz, with an increase beyond this range up to 3.4 GHz, where it then stabilizes at a high level.

Analyzing the results in Fig. 5, we can observe slightly better performance and energy efficiency for a block size of 512 on the Intel Xeon Gold platform and for a block size of 128 on the other platforms (see Table 3). The tests showed that this trend is also consistent for the other two compilers.

The results of tests conducted on the Intel Xeon Gold platform in the context of WZ factorization with strip-mining indicate that the optimal block size depends on the clock frequency. How-

Table 3
Percentage advantage in terms of energy savings for the best block in the context of WZ factorization using strip-mining, across platforms and compilers

Platform (compared blocks)	GCC	ICC	OneAPI
Intel Xeon Gold (512 vs 128 and 256)	0%–10%	0%–10%	0%–7%
Intel Xeon Platinum (128 vs 256 and 512)	5%–20%	3%–19%	6%–16%
AMD EPYC (128 vs 256 and 512)	0.1%–2%	0.4%–1.9%	0.8%–1.4%

ever, for most clock settings, a block size of 512 proves to be the most efficient in terms of performance and energy consumption, regardless of the compiler used (first row in Table 3). The advantage of the 512 block over other blocks does not exceed 10%. In situations where this advantage is zero (Table 3), it indicates that for some frequency settings, there is no noticeable advantage, and there may even be a slight decrease. Nevertheless, for most frequency settings, the 512 block demonstrates an advantage. Therefore, further comparisons of compiler efficiency on this platform will be limited to the analysis of results for the 512 block.

On the Intel Xeon Platinum and AMD EPYC platforms, test results showed better performance and energy efficiency for the 128 block, regardless of clock frequency, with the advantage on the Intel Xeon Platinum platform reaching up to 20% (second row in Table 3). On the AMD EPYC platform, the differences between blocks are minimal, not exceeding 2%, but the 128 block still appears to be more optimal (third row in Table 3). Consequently, further analyses on these platforms will focus on the 128 block.

The Intel Xeon Gold processor, with fewer cores and smaller cache compared to Intel Xeon Platinum and AMD EPYC, benefits more from larger block sizes (512). Larger blocks make better use of the available processing resources and cache, minimizing the overhead associated with context switching and improving performance. This approach allows for longer operations on a single core, reducing the costs of memory management and synchronization between cores.

In contrast, Intel Xeon Platinum and AMD EPYC, with significantly more cores and larger, more advanced cache structures, perform better with smaller block sizes (128). Smaller blocks allow for more efficient parallel data processing across multiple cores, improving overall performance by enabling each core to operate more effectively. Larger blocks on these platforms, however, could overwhelm the cache or lead to delays in data access between cores.

Thus, for the Intel Xeon Gold processor, larger blocks are more beneficial as they make better use of the limited cores and cache. On the other hand, Intel Xeon Platinum and AMD EPYC perform more efficiently with smaller blocks, as tasks are distributed more evenly across cores, minimizing memory access delays and enhancing parallelism.

6.2. Platform-specific analysis of compiler impact on WZ factorization with strip-mining

In this section, we will analyze the impact of the compiler on the time and energy optimization of the WZ factorization algorithm using strip-mining on various platforms. To this end, we focus on analyzing the basic-sm version of the WZ factorization algorithm for the best block sizes for a given platform according to the results in Table 3.

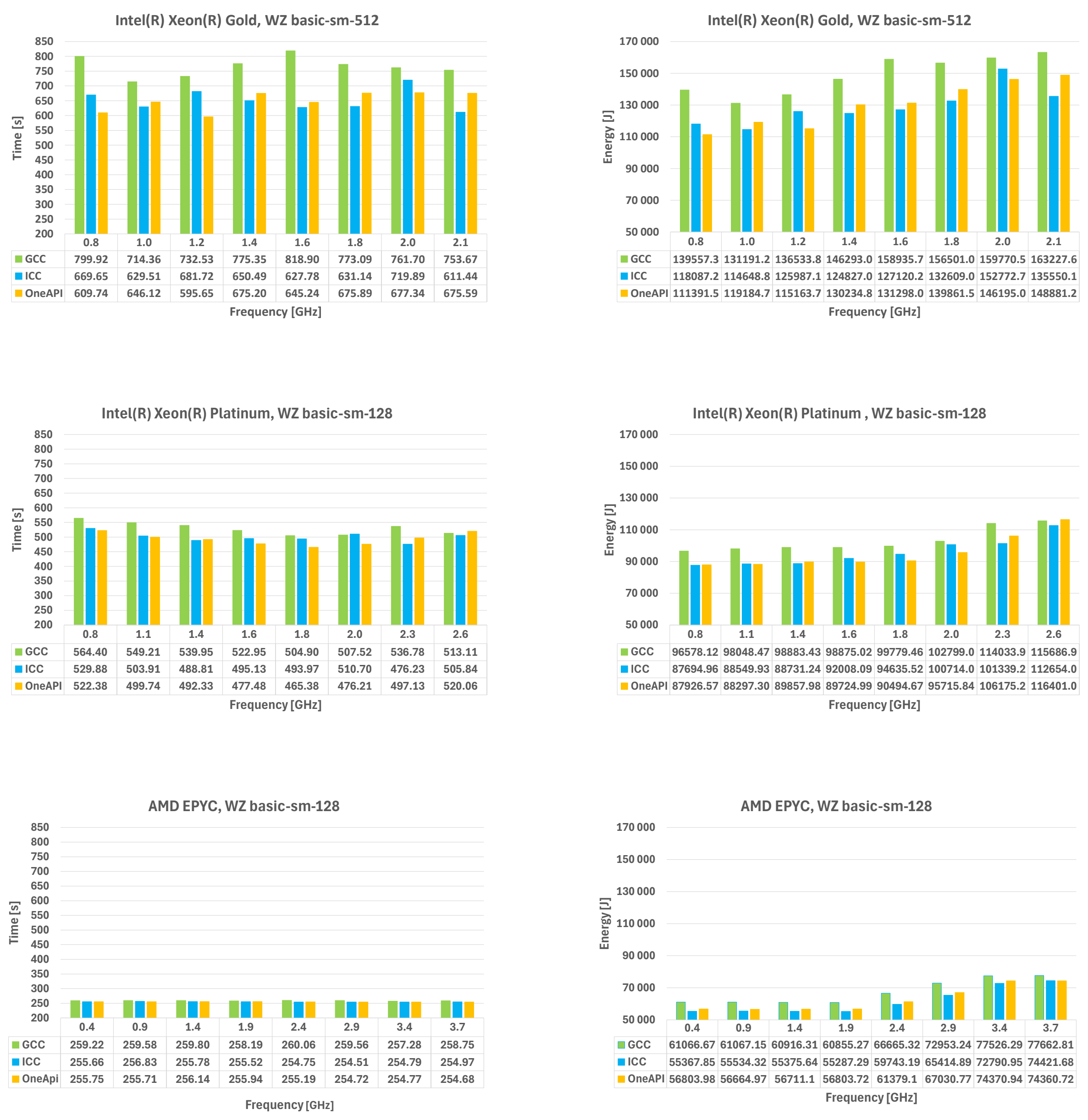
Figure 6 presents the runtime (left column) and energy consumption (right column) of selected basic-sm versions of the WZ factorization algorithm across different platforms, from top to bottom: Intel Xeon Gold, Intel Xeon Platinum, and AMD EPYC. The green bars represent the runtime and energy consumption for the GCC compiler, the blue bars for the ICC compiler, and the orange bars for the OneAPI compiler. All charts use the same scale, which highlights the difference in bar heights, demonstrating the performance advantage of the more powerful machines. The scale does not start at 0 to better highlight differences between individual compilers.

Let us now note the differences between the compilers used. In Fig. 6, we observe that the GCC compiler performs worse on Intel platforms compared to the ICC and OneAPI compilers, which are specifically designed for these platforms. Additionally, GCC also underperforms on the AMD platform, despite having a broader spectrum than the Intel-dedicated ICC and OneAPI compilers. GCC was the least efficient compiler on all platforms. This is probably because GCC, while versatile, is not as well-tuned for specific hardware as ICC and OneAPI are for Intel processors.

The ICC and OneAPI compilers perform well across all three tested platforms. However, identifying the superior compiler reveals a dependency on clock frequency settings. More significant differences between them are observed on the Intel Xeon platforms than on the AMD EPYC. On the Intel Xeon Gold platform, the ICC compiler outperforms OneAPI in 5 out of 8 considered frequencies; for instance, at 2.1 GHz, the advantage is about 9% in energy and in time. However, at 1.2 GHz, OneAPI surpasses ICC with a 13% and 9% advantage in time and energy, respectively. Similar frequency-dependent differences are observed on the Intel Xeon Platinum platform, where OneAPI more frequently has the upper hand (in 4 out of 8 frequencies). For example, at 2.0 GHz, OneAPI shows a 5% and 7% advantage in time and energy, respectively, whereas at 2.3 GHz, ICC surpasses OneAPI by approximately 4% in both time and energy.

On the AMD EPYC platform, ICC generally outperforms OneAPI at most tested frequencies, but the differences are minimal, not exceeding 1% and 3% in time and energy consumption, respectively.

Table 4 provides a summary of the WZ factorization algorithms with strip-mining, along with the most optimal compiler for each and the frequency at which the best energy consumption



	0.8	1.0	1.2	1.4	1.6	1.8	2.0	2.1
🟩 GCC	799.92	714.36	732.53	775.35	818.90	773.09	761.70	753.67
🟦 ICC	669.65	629.51	681.72	650.49	627.78	631.14	719.89	611.44
🟨 OneAPI	609.74	646.12	595.65	675.20	645.24	675.89	677.34	675.59

	0.8	1.0	1.2	1.4	1.6	1.8	2.0	2.1
🟩 GCC	139557.3	131191.2	136533.8	146293.0	158935.7	156501.0	159770.5	163227.6
🟦 ICC	118087.2	114648.8	125987.1	124827.0	127120.2	132609.0	152772.7	135550.1
🟨 OneAPI	111391.5	119184.7	115163.7	130234.8	131298.0	139861.5	146195.0	148881.2

	0.8	1.1	1.4	1.6	1.8	2.0	2.3	2.6
🟩 GCC	564.40	549.21	539.95	522.95	504.90	507.52	536.78	513.11
🟦 ICC	529.88	503.91	488.81	495.13	493.97	510.70	476.23	505.84
🟨 OneAPI	522.38	499.74	492.33	477.48	465.38	476.21	497.13	520.06

	0.8	1.1	1.4	1.6	1.8	2.0	2.3	2.6
🟩 GCC	96578.12	98048.47	98883.43	98875.02	99779.46	102799.0	114033.9	115686.9
🟦 ICC	87694.96	88549.93	88731.24	92008.09	94635.52	100714.0	101339.2	112654.0
🟨 OneAPI	87926.57	88297.30	89857.98	89724.99	90494.67	95715.84	106175.2	116401.0

	0.4	0.9	1.4	1.9	2.4	2.9	3.4	3.7
🟩 GCC	259.22	259.58	259.80	258.19	260.06	259.56	257.28	258.75
🟦 ICC	255.66	256.83	255.78	255.52	254.75	254.51	254.79	254.97
🟨 OneApi	255.75	255.71	256.14	255.94	255.19	254.72	254.77	254.68

	0.4	0.9	1.4	1.9	2.4	2.9	3.4	3.7
🟩 GCC	61066.67	61067.15	60916.31	60855.27	66665.32	72953.24	77526.29	77662.81
🟦 ICC	55367.85	55534.32	55375.64	55287.29	59743.19	65414.89	72790.95	74421.68
🟨 OneAPI	56803.98	56664.97	56711.1	56803.72	61379.1	67030.77	74370.94	74360.72

Fig. 6. Runtime and energy consumption of across platforms and compilers for selected block size

Table 4
Best compiler choice for performance and energy efficiency for selected platform and block size in WZ Factorization with strip-mining

Platform/algorithm	Compiler	Clock settings [GHz]	Runtime [s]	Total energy [J]	Performance [Gflops/s]	Energy efficiency [Gflops/J]	Time loss	Energy gain
Intel Xeon Gold/basic-sm-512	ICC	1.0	666.18	121274.95	35.21	0.193	2.9%	15.4%
Intel Xeon Platinum/basic-sm-128	ICC	0.8	529.88	87694.96	44.27	0.267	4.5%	22.2%
AMD EPYC/basic-sm-128	ICC	1.9	255.52	55287.29	91.80	0.424	0.2%	25.7%

result was achieved. Across all three platforms, the best results were obtained using the ICC compiler. The last two columns of Table 4 describe the percentage time loss and energy gain relative to the change observed when reducing the clock frequency from the maximum for a given platform to the value indicated in the third column of the table.

7. CONCLUSIONS

The subject of our research was to investigate the impact of the choice of C/C++ compilers when scaling the clock frequency using the DVFS technique on the performance and energy efficiency of the WZ factorization algorithm in two versions: basic and optimized using strip-mining. We conducted this analysis on three different computing platforms: Intel Xeon Gold, Intel Xeon Platinum and AMD EPYC, to identify the optimal configurations for each hardware environment. The goal of this study was to determine which C/C++ compilers, for different hardware configurations, provide the best performance in terms of execution time and energy consumption.

In terms of computing platforms, the best performance results were obtained for the AMD EPYC processor with the largest number of cores and the widest clock range, despite the largest TDP that this processor has. In terms of energy efficiency, the advantage is about 35% over Intel Xeon Platinum, which in turn shows an advantage of about 20% over Intel Xeon Gold.

Our tests show that Intel ICC and OneAPI compilers outperform GCC in terms of execution time optimization and energy efficiency (Fig. 7). Note the slightly different scales on the X-axis of the graphs representing the different platforms. The performance differences between the compilers were more pronounced on Intel platforms, where ICC and OneAPI showed a significant

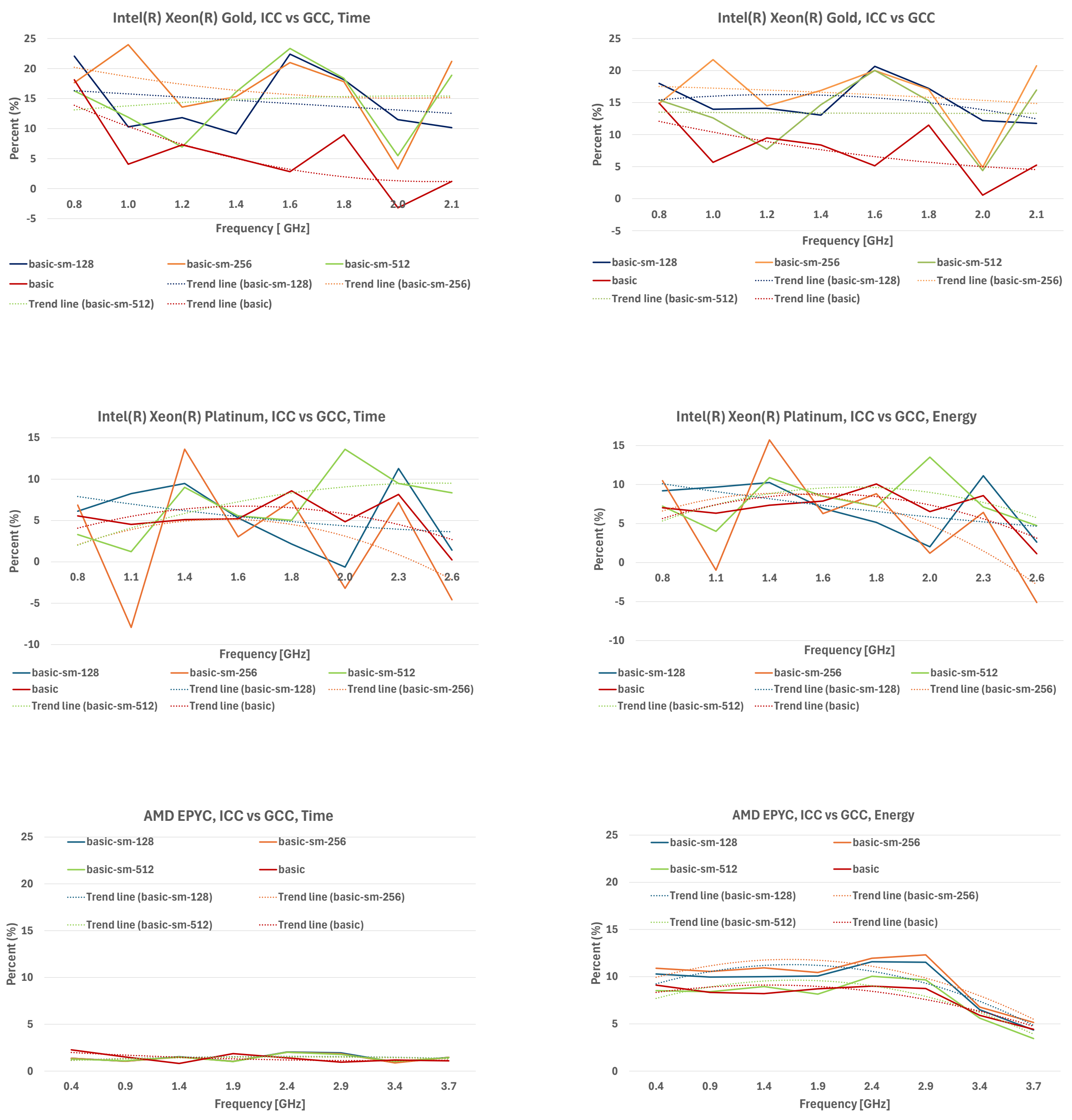


Fig. 7. Percentage advantage of Intel compiler (ICC) over GCC for WZ factorization algorithm with strip-mining optimization. The left column concerns the runtime of the energy consumption law

advantage over GCC. However, there were exceptional cases where GCC performed better in terms of execution time on Intel Xeon platforms. One such case was observed on Intel Xeon Gold at 2.0 GHz, where GCC outperformed the Intel compiler for the basic WZ factoring algorithm by 3% in execution time. On Intel Xeon Platinum, these cases were more frequent (Fig. 7 – middle line), where GCC outperformed in execution time for the three frequencies tested, although the advantage did not exceed 8% (Fig. 7). These frequencies – 1.1 GHz, 2.0 GHz, and 2.6 GHz – showed a GCC advantage for the algorithm optimized for bandwidth mining with smaller block sizes. This advantage also held when taking into account energy efficiency, but did not exceed 5% and is observed at 1.1 GHz and 2.6 GHz only for a block size of 256.

On the AMD EPYC platform, GCC performed relatively better compared to Intel platforms, although ICC and OneAPI still consistently delivered better results, with a smaller advantage of about 2% in execution time and up to 13% in power consumption.

Both ICC and OneAPI performed well on all platforms tested, but their relative performance depended on the clock speed settings. On Intel Xeon platforms, ICC more often outperformed OneAPI at higher frequencies, while OneAPI showed better performance at lower frequencies. On AMD EPYC processors, ICC maintained a small advantage with minimal differences (Fig. 6).

Figure 7 illustrates the percentage advantage of ICC over GCC. We can see that this advantage decreases with increasing hardware capabilities – more cores and higher clock speeds lead to decreasing performance differences, as shown by the trend lines (dashed lines – represent polynomial trend lines) that decrease with increasing clock speed (Fig. 7). It appears that on more advanced hardware, where computations are more parallel (due to more cores) and clock speeds are higher, the impact of specific compiler optimizations becomes less significant on overall performance. This suggests that on modern hardware, differences in compiler choice may be less critical, and hardware quality and capabilities play a more dominant role. However, as our tests show, especially when energy efficiency is a priority, it can still be beneficial to reduce the clock speed (to 0.8–1.0 GHz for Intel Xeon platforms and 1.9 GHz for AMD EPYC). As shown in Tables 2 and 4, choosing the right compiler also plays a significant role.

Our tests covered two versions of the WZ factoring algorithm, the basic version and the one with strip-mining optimization. The tests showed (see Table 2 and Table 4) that the basic version of the algorithm performed slightly better with a minimal advantage on AMD EPYC and Intel Xeon Platinum platforms and with a slightly larger advantage of about 10% and 12%, respectively, in terms of performance and energy efficiency on the Intel Xeon Gold platform.

In our upcoming research, we aim to explore the nuanced correlations between compiler selections and how they influence the efficiency and energy consumption of algorithms like the WZ factorization, along with pivotal ones such as Cholesky, LU, and QR decomposition in dense linear algebra. Specifically, we will scrutinize how these influences may vary across diverse hardware architectures, encompassing traditional central processing units (CPUs), graphics processing units (GPUs), hybrid multiprocessor systems, and emerging architectures like RISC-V.

Data from all conducted experiments are available in the public repository at https://github.com/mdpiekarz/time-energy-compiler-choice-article.

REFERENCES

[1] A. Paszkiewicz, C. Ćwikła, M. Bolanowski, M. Ganzha, M. Paprzycki, and M. Hodoň, "Multifunctional clustering based on the leach algorithm for edge-cloud continuum ecosystem," *Bull. Pol. Acad. Sci. Tech. Sci.*, vol. 72, no. 1, p. e147919, 2024, doi: 10.24425/bpasts.2023.147919.

[2] M. Łoś, M. Woźniak, and M. Paszynski, "Varying coefficients in parallel shared-memory variational splitting solvers for non-stationary Maxwell equations," *Bull. Pol. Acad. Sci. Tech. Sci.*, vol. 72, no. 3, p. e149179, 2024, doi: 10.24425/bpasts.2024.149179.

[3] B. Bylina, J. Bylina, and M. Piekarz, "Impact of processor frequency scaling on performance and energy consumption for wz factorization on multicore architecture," *Ann. Comput. Sci. Inf. Syst.*, vol. 35, p. 377–383, 2023, doi: 10.15439/2023F6213.

[4] D. Evans and M. Hatzopoulos, "A parallel linear system solver," *Int. J. Comput. Math.s*, vol. 7, no. 3, pp. 227–238, 1979, doi: 10.1080/00207167908803174.

[5] H.K. Olayiwola Babarinsa, "Quadrant interlocking factorization of hourglass matrix," *AIP Conf. Proc.*, vol. 1974, no. 1, p. 030009, 06, 2018, doi: 10.1063/1.5041653.

[6] H.K. Olayiwola Babarinsa and A.Z. Hailiza Kamarulhaili, "Quadrant interlocking factorization algorithm of hourglass matrix from nonsingular matrix," *Thai J. Math.*, vol. 19, no. 4, p. 1461–1476, Dec. 2021.

[7] G. Meurant, *Direct and Iterative Methods for Linear Systems*. Gérard Meurant: Paris, France, 2023. [Online]. Available: https://gerard-meurant.fr/book_2023.pdf

[8] B. Bylina and J. Bylina, "Nested loop transformations on multi- and many-core computers with shared memory," in *Selected Topics in Applied Computer Science*. Lublin: Maria Curie-Skłodowska University Press, 2021, vol. I, pp. 167–186. [Online]. Available: http://stacs.matrix.umcs.pl/v01/stacs_v01.pdf

[9] B. Bylina and J. Bylina, "The parallel tiled wz factorization algorithm for multicore architectures," *Int. J. Appl. Math. Comput. Sci.*, vol. 29, pp. 407–419, 2019.

[10] J. Bylina, B. Bylina, and M. Piekarz, "Influence of loop transformations on performance and energy consumption of the multithreded wz factorization," in *Preproc. 17th Conference on Computer Science and Intelligence Systems*, 2022, p. 479–488, doi: 10.15439/2022F251.

[11] T. Kaczorek, "Transformations of the matrices of linear systems to their canonical form with desired eigenvalues," *Bull. Pol. Acad. Sci. Tech. Sci.*, vol. 71, no. 6, p. e147342, 2023, doi: 10.24425/bpasts.2023.147342.

[12] "GREEN500," https://www.top500.org/lists/green500/, 2022.

[13] J.V. Lima, I. Raïs, L. Lefevre, and T. Gautier, "Performance and energy analysis of openmp runtime systems with dense linear algebra algorithms," in *2017 International Symposium on Computer Architecture and High Performance Computing*

Workshops (SBAC-PADW), 2017, pp. 7–12, doi: 10.1109/SBAC-PADW.2017.10.

[14] M. Mirka, G. Devic, F. Bruguier, G. Sassatelli, and A. Gamatié, "Automatic energy-efficiency monitoring of openmp workloads," in *2019 14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2019, pp. 43–50, doi: 10.1109/ReCoSoC48741.2019.9034988.

[15] M.A. Shahneous Bari, A.M. Malik, A. Qawasmeh, and B. Chapman, "Performance and energy impact of openmp runtime configurations on power constrained systems," *Sustain. Comput.-Informatics Syst.*, vol. 23, pp. 1–12, 2019.

[16] J.V.F. Lima, I. Raïs, L. Lefèvre, and T. Gautier, "Performance and energy analysis of OpenMP runtime systems with dense linear algebra algorithms," *Int. J. High Perform. Comput. Appl.*, vol. 33, no. 3, pp. 431–443, 2019, doi: 10.1177/1094342018792079.

[17] J. Dongarra, H. Ltaief, P. Luszczek, and V.M. Weaver, "Energy footprint of advanced dense numerical linear algebra using tile algorithms on multicore architectures," in *2012 Second International Conference on Cloud and Green Computing*, 2012, pp. 274–281, doi: 10.1109/CGC.2012.113.

[18] L. Szustak, R. Wyrzykowski, T. Olas, and V. Mele, "Correlation of performance optimizations and energy consumption for stencil-based application on Intel Xeon scalable processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 11, pp. 2582–2593, 2020, doi: 10.1109/TPDS.2020.2996314.

[19] T. Jakobs and G. Rünger, "Examining energy efficiency of vectorization techniques using a Gaussian elimination," in *2018 International Conference on High Performance Computing Simulation (HPCS)*, 2018, pp. 268–275, doi: 10.1109/HPCS. 2018. 00054.

[20] K. Halbiniak, R. Wyrzykowski, L. Szustak, A. Kulawik, N. Meyer, and P. Gepner, "Performance exploration of various C/C++ compilers for AMD EPYC processors in numerical modeling of solidification," *Adv. Eng. Softw.*, vol. 166, pp. 1–14, 2022, doi: 10.1016/j.advengsoft.2021.103078.

[21] GNU Compiler Collection, "GNU Compiler Collection," https://gcc.gnu.org/, 2023.

[22] Intel Corporation, "Intel Developer Zone," https://www.intel.com/content/www/us/en/resources-documentation/developer.html

[23] Intel Corporation, "Intel oneAPI Toolkits," https://www.intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html

[24] K. Khan, M. Hirki, T. Niemi, J. Nurminen, and Z. Ou, "RAPL in action: Experiences in using RAPL for power measurements," *ACM Trans. Modeling Perform. Eval. Comput. Syst.*, vol. 3, 01 2018, doi: 10.1145/3177754.

[25] B. Bylina, J. Potiopa, M. Klisowski, and J. Bylina, "The impact of vectorization and parallelization of the slope algorithm on performance and energy efficiency on multi-core architecture," *Ann. Comput. Sci. Inf. Syst.*, vol. 25, pp. 2283–2290, 2021.