DOI: 10.15439/2023F6213
ISSN 2300-5963 ACSIS, Vol. 35

# Impact of processor frequency scaling on performance and energy consumption for WZ factorization on multicore architecture

Beata Bylina, Jaroslaw Bylina, Monika Piekarz
0000-0002-1327-9747, 0000-0002-0319-2525, 0000-0002-3457-9335
Institute of Computer Science, Marie Curie-Sklodowska University
Pl. M. Curie-Skłodowskiej 5
Lublin, 20-031, Poland
Email: {beata.bylina, jaroslaw.bylina, monika.piekarz}@mail.umcs.pl

***Abstract*—With the growing demand for computing power, new multicore architectures have emerged to provide better performance. Reducing their energy consumption is one of the main challenges in achieving high performance computing. Current research trends develop new software and hardware techniques to achieve the best performance and energy compromise. In this work, we investigate the effect of processor frequency scaling using Dynamic Voltage Frequency Scaling on performance and energy consumption for the WZ factorization. This factorization is implemented both without optimization techniques and with strip mining. This technique involves transforming the program loop to improve program performance. Based on time and energy tests, we have shown that for the WZ factorization algorithm, regardless of the presence of manual optimization, it pays to reduce the frequency to save energy without losing performance. The conclusion can be extended to analogous algorithms — also having a high ratio of memory access to computational operations.**

***Index Terms*—processor frequency scaling, performance, energy, WZ factorization**

## I. Introduction

MULTICORE architectures are now common in all computing environments, from portable handheld devices to HPC computing platforms and supercomputers. The advent of multicore architectures has increased application performance by allowing them to run at a higher level of parallelism. This opened a new era for High Performance Computing (HPC). Of course, the increase in efficiency is closely related to the increase in energy consumption. Computing energy is currently a serious problem with dire environmental and economic consequences, and its mitigation is extremely important.

Top500 [1] is a website that has been updating the top 500 supercomputers list for performance in the LINPACK [2] benchmark since 1993. GREEN500 [3] is a complementary list to part of the TOP500 list, which since 2007 has ranked the top 500 supercomputers in terms of energy efficiency. The discrepancy between the energy-saving supercomputers and the fastest supercomputers may be seen from their respective positions in both lists. The relationship between performance and energy consumption is unclear and depends on many factors.

Energy efficiency can be achieved on the following two levels: hardware [3] and software [4], [5], [6], [7], [8], [9], [10]. The first approach is based on innovation in computer hardware, represented by microarchitecture and advances in the design of integrated circuits. Based on the software, the second approach can be divided into two categories: optimization of energy consumption at the operating system level and optimization of energy consumption at the application level. The approach to optimizing power consumption at the operating system level focuses on minimizing the power consumption of the entire node through the use of techniques such as clock and power gating [11], dynamic voltage, and frequency scaling (DVFS) [12], [13]. In contrast, application-level optimization methods use application-level parameters and models to maximize the energy efficiency of the application.

Some papers have analyzed the energy impact of numerical linear algebra algorithms that do not change the code but only control software parameters (e.g. block size) accordingly. ATLAS (Automatically Tuned Linear Algebra Software) [14] is an example of the automatic tuning of a library to the architecture of the machine for reducing execution time. The focus of the paper [15] is to propose a method for tuning the ATLAS library, whereby it is possible to replace the execution time tuning process by tuning the energy consumption. In that work, the performance and the number of MFlops/J as well as the execution time and energy consumption were investigated for single and double precision for different array sizes.

In the article [16], different techniques related to algorithm transformation were used to reduce static and dynamic energy consumption on multicore machines with shared memory. In particular, one of the most popular matrix factorizations, namely the LU factorization, was considered. In the LU factorization code, the most energetically expensive instructions were extracted, and then, by performing a code transformation, an attempt was made to reduce their number. That work investigated the effect of the number of threads on dynamic and total energy consumption, performance, and also the number of

MFlops/W for different matrix sizes at a fixed number of threads.

The authors of [17], [18], present algorithms for solving systems of equations, trying to improve their performance, in particular in parallel. Improvement in performance was obtained by appropriate transformation of the underlying algorithm using looping tiling and appropriate data structures. There are currently not too many studies in the literature on the analysis of both efficiency and energy consumption in the context of loop transformations.

In our work, we study a numerical algorithm (the WZ factorization) in which loops are transformed. This algorithm concerns numerical linear algebra, particularly solving systems of equations on multi-core architectures using OpenMP in constant performance and energy consumption.

In this paper, we will focus on the combination of two approaches at the software level, namely the DVFS technique (at the operating system level) and the optimization of the program algorithm (at the application level) on one of the latest multicore architectures. The main idea is to capture the actual performance and energy consumption of a multi-threaded computing application when it is executed on a multicore computing platform by changing the clock frequency.

The LU factorization is a well-known factorization used to solve the linear systems. From the very nature, this factorization is sequential. Throughout recent years, parallel implementations of LU decomposition have been implemented on various modern computers. In particular, the researchers have taken into account the improvement of its parallel performance. The WZ factorization is designed straight into parallel computers of SIMD type according to Flynn classification and it seems to be a potentially attractive alternative to Gaussian elimination or Cholesky factorization for parallel computations, especially for SIMD computers. The advantage of the WZ factorization is that it simultaneously evaluates two columns or two rows instead of one column or one row as it happens with the LU factorization. The WZ factorization has a fewer number of iterations of the loop but more computations in each iteration in comparison with the LU factorization. For the WZ factorization, we can achieve higher parallelism. The algorithms with higher parallelism are desirable for multicore architectures.

In work [19], we presented the detailed implementation of multi-threaded WZ factorization with OpenMP [20] on multicore architecture using various nested loop transformation strategies to program optimization. In this work, we investigate the effect of CPU frequency scaling on performance and energy consumption for the WZ factorization in two selected versions, namely basic and strip-mining (optimized). We selected these versions for testing based on the results of the work [21]. Strip-mining is a loop transformation technique to improve memory performance. Additionally, we are examining the influence of the parameter value of the strip-mining — block size — for the WZ factorization in the strip-mining version on energy efficiency.

The rest of the paper is organized as follows. Section 2 introduces the DVFS technique. Section 3 describes the WZ factorization algorithm using OpenMP programming models in two versions. Section 4 presents the numerical experimental evaluation of the impact of processor frequency scaling on performance and energy consumption for WZ factorization on multicore architecture. Lastly, Section 5 concludes the paper.

## II. DVFS — PROCESSOR FREQUENCY SCALING

Nowadays, computers incorporate various energy management techniques that support the reduction of energy consumption. Examples are Dynamic Voltage Frequency Scaling (DVFS) or, for example, the use of special instructions and specialized coprocessors. DVFS [12], [13] is a technique that reduces the clock frequency and voltage level of various computing node components (CPU, DRAM, etc.) at the cost of some performance degradation. Today, DVFS is widely supported by energy-saving and efficient processors supplied by different vendors under different names (eg SpeedStep for Intel [22] and PowerNow processors or Cool'n'Quiet for AMD processors [23]). DVFS can reduce the power consumption of a CMOS chip such as modern processors by reducing the frequency at which it operates as shown in the formula:

$$P = CfV^2 + P_{static}$$

where $C$ is the capacitance of the transistor gates, $f$ is the operating frequency, $V$ is the supply voltage, and $P_{static}$ is a static power which is mainly due to various leakage currents. The voltage required for stable operation is determined by the clock frequency of the circuit and may be reduced if the frequency is also reduced. This can result in a significant reduction in energy consumption due to the compound $V^2$ shown above.

However, dynamic power $CfV^2$ alone is not the total power of the system. Due to the static power consumption and the asymptotic execution time, it has been shown that the power consumption of the software shows a convex energy behavior, i.e. there is an optimal processor frequency at which energy consumption is minimized [24].

## III. WZ FACTORIZATION

We present shortly the WZ factorization [25], [26]. We transform a square and nonsingular matrix $\mathbf{A}$ into a product of two matrices, namely $\mathbf{WZ}$. The matrix $\mathbf{W}$ is a matrix of the form of a butterfly with units on its main diagonal, the matrix $\mathbf{Z}$ is a matrix of the form of an hourglass. Both the matrices are complements of each other in the sense of the structure of non-trivial elements (one has non-trivial elements in places where the other has zeros/units — and vice versa). The forms of these matrices can be seen in Figure 1.

We chose this numerical algorithm here because it's quite complicated and difficult to optimize by the compiler. Figure 2 presents a parallel basic algorithm for the WZ factorization for an even size of the matrix (we only consider even sizes — without loss of generality).

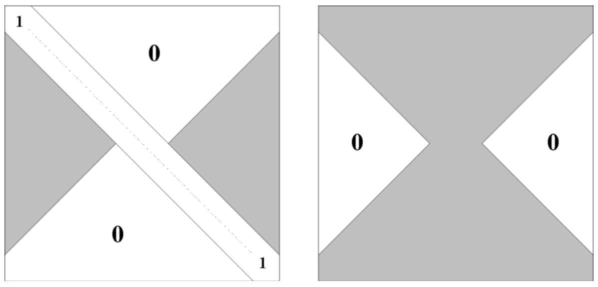


Fig. 1: The output of the WZ factorization — forms of the matrices **W** (left) and **Z** (right).

```
for(k = 0; k < n/2-1; k++) {
    // the following four lines are omitted
    // in the next versions of the algorithms
    // (thay are always the same)
    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 simd
        for(j = k+1; j < p; j++)
            a[i][j] = a[i][j]
                      - w[i][k]*a[k][j]
                      - w[i][p]*a[p][j];
    }
}
```

Fig. 2: The parallel basic algorithm for the WZ factorization — pseudocode.

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 feature multicore architectures. A good example is the use in the code of loop optimization 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` is 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).

In Figure 3, we present a parallel 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 is:

$$\frac{2}{3}n^3 + O(n^2)$$

and number of memory access is:

$$\frac{7}{6}n^3 = O(n^2)$$

what gives the ratio of memory access to computations — $\frac{7}{4}$. This means that we need almost two memory accesses for one floating point operation to perform our algorithm.

```
for(k = 0; k < n/2-1; k++) {
    . . .
#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 simd
            for(j = jj; j < jj+BLOCK_SIZE;
                        ++j)
                a[i][j] = a[i][j]
                          - w[i][k]*a[k][j]
                          - w[i][p]*a[p][j];
        }
    }
}
```

Fig. 3: Parallel strip-mining in the basic algorithm — pseudocode.

## IV. Numerical Experiments

### A. Methodology

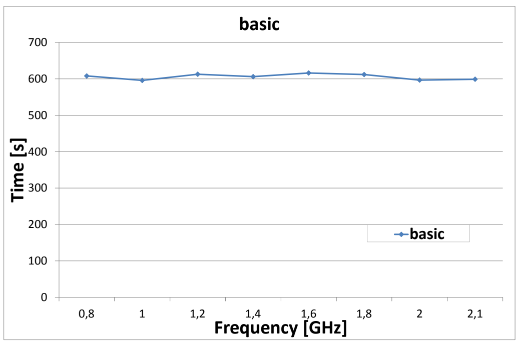
We test two types of versions of the WZ factorization algorithm: the basic algorithm and block algorithms with strip-mining. Our test dataset is a random square matrix with dimensions of $n \times n$ double-precision values, $n = 32768$. So our test dataset is 1073741824 cells, which is 8GB of data. All versions of the algorithm match the row-wise layout and are implemented in C++ with vectorization and parallelism. The following block sizes were checked during the tests: 64, 128, 256, 512.

Our experimental setup includes the following computing platform equipped with a multicore processor with the following parameters:

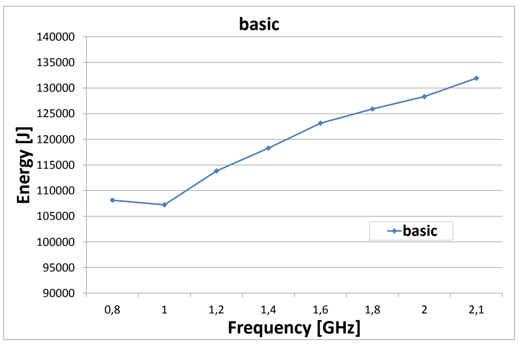
- processor: Intel(R) Xeon(R) Gold 5218R
- CPU @ 2.10GHz HT (2x20 cores)
- Cache: L1: 32KB, L2:1024KB, L3: 28MB

The following software was installed during tests:

- operating system: CentOS 7.5;
- kernel: Linux 3.10.0;



(a) Runtime



(b) Energy consumption

Fig. 4: Runtime and energy consumption of `basic` for data size 32768

TABLE I: Energy efficiency for `basic` (32768)

| Frequency [GHz] | Time [s] | Total energy [J] | Performance [Gflops] | Energy efficiency [Gflops/J] |
|---|---|---|---|---|
| 0.8 | 607.88 | 108132.066 | 38.59 | 0.217 |
| 1.0 | **595.58** | **107218.43** | **39.38** | **0.219** |
| 1.2 | 612.59 | 113827.77 | 38.29 | 0.206 |
| 1.4 | 605.92 | 118292.45 | 38.71 | 0.198 |
| 1.6 | 616.01 | 123146.44 | 38.08 | 0.190 |
| 1.8 | 611.79 | 125920.48 | 38.34 | 0.186 |
| 2.0 | 596.57 | 128343.30 | 39.32 | 0.183 |
| 2.1 | 598.88 | 131927.20 | 39.17 | 0.178 |

- icc compiler v. 2021.5.0 with the following compiler options:
  ```
  -qoenmp -O3 -ipo -no-prec-div
  -fp-model fast=2
  ```

We used the RAPL (Running Average Power Limit) interface [27] 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 [27] 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 work [28] and our experience [21], [29] 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 shown in [21], running HT for the tested versions of the WZ factorization algorithm has no benefit in speeding up the calculations so we run all versions without HT on 40 threads.

We made changes to the clock frequencies using CPU frequency scaling. The `intel_pstate` driver is used by default to control the performance of Intel processors on GNU/Linux systems. With this driver, we did not get a satisfactory effect of forcing the clock frequency, so we used the `acpi_cpufreq` driver, which by default follows the `conservative` governor. Governor `conservative` increases or decreases the clock frequency depending on the core load, selecting one of several available frequencies from the minimum to the maximum supported by the editor. For the Intel Xeon Gold 5218R processors we use, the permissible frequencies range from 0.8 GHz to 2.1 GHz. Using the `cpupower` program, we change the minimum and maximum values of the CPU frequency limit to a given level. The frequency setting was done for all cores of both installed processors with the commands:

```
cpupower frequency-set -d 1800000
cpupower frequency-set -u 1800000
```

for setting the minimum and maximum frequency limit values to 1.8 GHz.

We conducted our tests for frequencies ranging from 0.8 GHz to 2.1 GHz, making changes every 0.2 GHz, 0.8 GHz is the lowest frequency to which we could lower the clock.

### B. *The performance and energy consumption for `basic` WZ factorization algorithm*

First, we measure the runtime of the `basic` version of the WZ factorization algorithm. The test results are presented in Figure 4.

In Figure 4 on the left diagram we can see that the algorithm runtime is similar regardless of the clock frequency. The difference between the shortest operating time (frequency 1.0

GHz) and the longest (frequency 1.6 GHz) is 3% (20 seconds). In Figure 4 on the right diagram we can see that the energy consumption increases with increasing clock frequency. For a higher frequency, we have a greater instantaneous power consumption, hence, with a similar runtime, we have a greater energy consumption. We have the lowest energy consumption for the frequency of 1.0 GHz. Lots of energy were consumed at the highest frequency. Lowering the clock frequency to 1.0 GHz saves about 19% of the energy consumed here.

In Table I, we have a summary of the runtime, total energy, performance, and energy efficiency for the `basic` algorithm. We can see that both the best performance and the energy efficiency of the algorithm will be achieved at the frequency of 1.0 GHz. Thus, there are benefits to lowering the clock frequency.

### *C. The performance and energy consumption for* `basic-sm` *versions of WZ factorization algorithm*

Next, we test block versions of the WZ factorization algorithm with strip-mining (abbreviated as `sm`). We consider four block sizes: 64, 128, 256, 512 so we have the following versions: `basic-sm-64`, `basic-sm-128`, `basic-sm-256`, `basic-sm-512`. Our goal is to answer the question of whether the `sm` optimization will affect the performance and energy consumption and whether the additional clock frequency scaling for the `sm` version will also have a positive effect on the reduction of energy consumption without loss of performance.

The test results are presented in Figure 5. Here we see a similar situation as in the case of the `basic` version, the operating time of the tested versions of the algorithm slightly fluctuates for different frequencies (left diagram in Figure 5), while the energy consumption increases with increasing frequency (right diagram in Figure 5). We can also observe that the lowest energy consumption for each of the tested blocks was for the lowest tested frequency: 0.8 GHz. In addition, we can notice that, regardless of the frequency, block 64 was the weakest, both in terms of operating time and energy consumed. The best, however, are blocks 256 and 512.

In Table II and Table III, we have a summary of the runtime, total energy, performance and energy efficiency for the `basic-sm-256` and `basic-sm-512` versions of algorithm respectively.

In this case, other than for the `basic` version of the algorithm, the best performance and the best energy efficiency are not achieved for the same frequency. It is the same for both blocks, the best performance was obtained for the 1.4 GHz frequency and the best efficiency for 0.8 GHz. Here, too, we can infer that lowering the frequency (in the case of our algorithm to 0.8 GHz) results in less energy consumption. We save 21% of energy consumption without losing time for block 256 and save 19% of energy consumption lose about 2% (11 seconds) of time for block 512 compared to 2.1 GHz. If we look at high frequencies (2.0 and 2.1 GHz), we get slightly better results in terms of efficiency and energy efficiency for block 512 compared to block 256. However, if we want to reduce energy consumption without losing time, a bit better results are observed for block 256. The Figure 6 shows a juxtaposition of tests for the `basic` and `baisc-sm -256` versions. Lowering the clock to 0.8 GHz we can see that strip-mining will pay off regardless of the clock frequency. Meanwhile, if we use the clock frequency reduction mechanism, we can get 21% for `basic-sm-256` without wasting time, and for `basic` equal to 19% without wasting time, but for `basic-sm-256` it pays to lower the clock frequency to 0.8 GHz, and for `basic` it is enough to 1.0 GHz.

## V. Conclusion

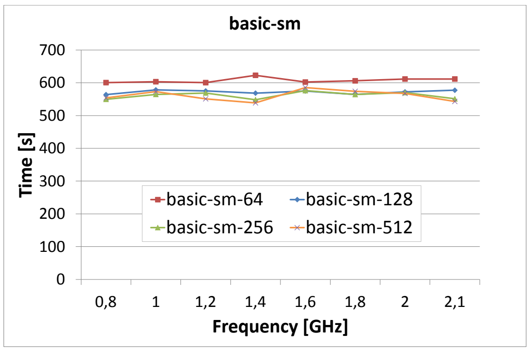
In this paper, we focused on the combination of two approaches, namely the DVFS technique and the optimization of the strip-mining loop. Our goal was to answer the question of whether traditional methods of strip-mining loop optimization in combination with the DVFS technique will reduce energy consumption without major losses on performance. Measurements were made on a 2nd Generation Intel Xeon Scalable Processors using the Intel RAPL interface.

There are a lot of memory references in our algorithm, so frequency scaling does not significantly affect performance, as our tests showed. The reduction of the frequency to the level of 0.8–1.0 GHz did not cause a decrease in performance in the case of the `basic` version, Table I. We improved memory access using strip-mining transformation, which resulted in a performance increase of about 9%. Although in the case of the `basic-sm` versions lowering frequency we lose a bit of performance (less than 2%, Table II), we can still lower the frequency while maintaining better performance than for the basic version.

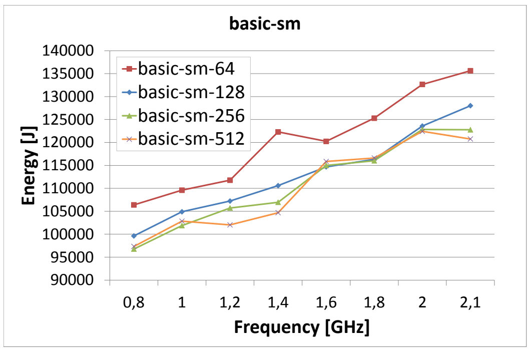
Our tests also showed two facts. First, the first version of the `basic` will use more energy than the `basic-sm`, which means that the strip-mining transformation will pay off. Second, with this algorithm, the frequency scaling affects the energy consumption. By reducing the frequency to 0.8 GHz, we can reduce the power consumption for `basic-sm-256` by 21%, Table II.

The conclusion from our tests is 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. The frequency that works best during experiments is the smallest that can be tested here, i.e. 0.8 GHz.

In the future, we plan to extend our tests by a wide one a range of architectures, including graphics cards. Moreover, we will evaluate the performance and energy consumption impact of various execution systems for OpenMP loop configurations and transformations for WZ and the three decomposition main kernels in dense linear algebra algorithms (Cholesky, LU and QR).



(a) Runtime



(b) Energy consumption

Fig. 5: Runtime and energy consumption of `basic-sm` for data size 32768

TABLE II: Energy efficiency for `basic-sm-256` (32768)

| Frequency [GHz] | Time [s] | Total energy [J] | Performance [Gflops] | Energy efficiency [Gflops/J] |
|---|---|---|---|---|
| 0.8 | 549.62 | **96781.44** | 42.68 | **0.242** |
| 1.0 | 564.49 | 101886.49 | 41.55 | 0.230 |
| 1.2 | 568.83 | 105726.79 | 41.24 | 0.222 |
| 1.4 | **548.34** | 106968.62 | **42.78** | 0.219 |
| 1.6 | 576.02 | 114994.40 | 40.72 | 0.204 |
| 1.8 | 564.11 | 116015.06 | 41.58 | 0.202 |
| 2.0 | 570.03 | 122833.96 | 41.15 | 0.191 |
| 2.1 | 551.03 | 122763.00 | 42.57 | 0.191 |

TABLE III: Energy efficiency for `basic-sm-512` (32768)

| Frequency [GHz] | Time [s] | Total energy [J] | Performance [Gflops] | Energy efficiency [Gflops/J] |
|---|---|---|---|---|
| 0.8 | 553.72 | **97352.90** | 42.36 | **0.240** |
| 1.0 | 573.04 | 102853.71 | 40.93 | 0.228 |
| 1.2 | 550.89 | 102039.32 | 42.58 | 0.230 |
| 1.4 | **538.57** | 104694.09 | **43.55** | 0.224 |
| 1.6 | 585.31 | 115855.60 | 40.07 | 0.202 |
| 1.8 | 574.01 | 116610.11 | 40.86 | 0.201 |
| 2.0 | 567.50 | 122428.81 | 41.33 | 0.192 |
| 2.1 | 542.94 | 120763.71 | 43.20 | 0.194 |

## References

[1] "Top500," https://www.top500.org/, 2022.

[2] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, *LINPACK users' guide*. : SIAM, 1979.

[3] "Green500," https://www.top500.org/lists/green500/, 2022.

[4] 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.

[5] 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.

[6] M. A. Shahneous Bari, M. M. Abid, A. Qawasmeh, and B. Chapman, "Performance and energy impact of openmp runtime configurations on power constrained systems," *Sustainable Computing: Informatics and Systems*, vol. 23, pp. 1–12, 2019.

[7] J. V. F. Lima, I. Raïs, L. Lefèvre, and T. Gautier, "Performance and energy analysis of OpenMP runtime systems with dense linear algebra algorithms," *The International Journal of High Performance Computing Applications*, vol. 33, no. 3, pp. 431–443, 2019.

[8] 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.

[9] 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 Transactions on Parallel and Distributed Systems*, vol. 31, no. 11, pp. 2582–2593, 2020.

[10] 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.

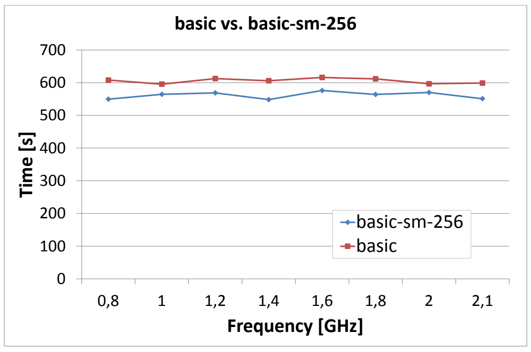
[11] A. Shahid, S. Arif, M. Qadri, and S. Munawar, "Power optimization using clock gating and power gating: A review," in *Innovative Research and Applications in Next-Generation High Performance Computing*, Q. F. Hassan, Ed. : IGI Global, 2016.

[12] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," *1st OSDI*, pp. 13–23, 1994.

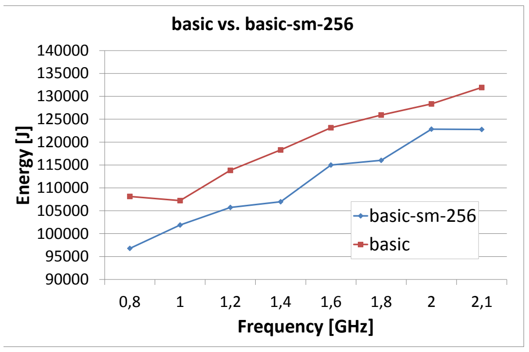
[13] W. A. and F. Bellosa, "Process cruise control - event-driven clock scaling for dynamic power management," *CASES*, 2002.

[14] R. C. Whaley, A. Petitet, and J. J. Dongarra, "Automated empirical optimizations of software and the ATLAS project," *Parallel Comput.*, vol. 27, no. 1-2, pp. 3–35, 2001. [Online]. Available: https://doi.org/10.1016/S0167-8191(00)00087-9

[15] T. Jakobs, J. Lang, G. Rünger, and P. Stocker, "Tuning linear algebra for energy efficiency on multicore machines by adapting the ATLAS library," *Future Gener. Comput. Syst.*, vol. 82, pp. 555–564, 2018.



(a) Runtime



(b) Energy consumption

Fig. 6: Runtime and energy consumption of `basic` and `basic-sm-256` for data size 32768

[Online]. Available: https://doi.org/10.1016/j.future.2017.03.009

[16] E. Garcia, J. Arteaga, R. S. Pavel, and G. R. Gao, "Optimizing the lu factorization for energy efficiency on a many-core architecture," in *International Workshop on Languages and Compilers for Parallel Computing*, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:489258

[17] S. Donfack, J. Dongarra, M. Faverge, M. Gates, J. Kurzak, P. Luszczek, and I. Yamazaki, "A survey of recent developments in parallel implementations of Gaussian elimination," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 5, pp. 1292–1309, 2015.

[18] J. J. Dongarra, M. Faverge, H. Ltaief, and P. Luszczek, "Achieving numerical accuracy and high performance using recursive tile LU factorization," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 6, pp. 1408–1431, 2013.

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

[20] R. Chandra, L. Dagum, D. Kohr, D. Maydan, R. Menon, and J. McDonald, *Parallel Programming in OpenMP*. San Francisco: Morgan Kaufmann Publishers, 2001.

[21] J. Bylina, B. Bylina, and M. Piekarz, "Influence of loop transformations on performance and energy consumption of the multithreded wz factorization," *Preproceedings of the of the 17th Conference on Computer Science and Intelligence Systems*, pp. 479–488, 2022, https://annals-csis.org/proceedings/2022/pliks/251.pdf.

[22] E. Rotem, A. Mendelson, A. Naveh, and M. Moffie, "Analysis of the enhanced intel® speedstep® technology of the pentium® m processor," https://www.cs.virginia.edu/~skadron/tacs/rotem_slides.pdf, 2004.

[23] "Amd powernow! technology dynamically manages powerand performance," https://www.amd.com/system/files/TechDocs/24404a.pdf, 2000.

[24] K. De Vogeleer, G. Memmi, P. Jouvelot, and F. Coelho, "The energy/frequency convexity rule:modeling and experimental validation onmobile devices," *PPAM'2013*, 2014.

[25] D. Evans and M. Hatzopoulos, "A parallel linear system solver," *International Journal of Computer Mathematics*, vol. 7, no. 3, pp. 227–238, 1979.

[26] P. Yalamov and D. Evans, "The wz matrix factorisation method," *Parallel Computing*, vol. 21, no. 7, pp. 1111–1120, 1995.

[27] K. Khan, M. Hirki, T. Niemi, J. Nurminen, and Z. Ou, "RAPL in action: Experiences in using RAPL for power measurements," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 3, 2018.

[28] 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 Transactions on Parallel and Distributed Systems*, vol. 31, no. 11, pp. 2582–2593, 2020.

[29] 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," *Annals of Computer Science and Information Systems*, vol. 25, pp. 2283–290, 2021.