Proceedings of the 18th Conference on Computer Science and Intelligence Systems pp. 895–900
DOI: 10.15439/2023F3506
ISSN 2300-5963 ACSIS, Vol. 35

The scalability in terms of the time and the energy for several matrix factorizations on a multicore machine

Beata Bylina
0000-0002-1327-9747
Maria Curie-Skłodowska University in Lublin
Pl. M. Curie-Skłodowskiej 5, 20-031 Lublin, Poland
Email: beata.bylina@mail.umcs.pl

Monika Piekarz
0000-0002-3457-9335
Maria Curie-Skłodowska University in Lublin
Pl. M. Curie-Skłodowskiej 5, 20-031 Lublin, Poland
Email: monika.piekarz@mail.umcs.pl

***Abstract*—Scalability is an important aspect related to time and energy savings on modern multicore architectures. In this paper, we investigate and analyze scalability in terms of time and energy. We compare the execution time and consumption energy of the LU factorization (without pivoting) and Cholesky, both with Math Kernel Library (MKL) on a multicore machine. In order to save the energy of these multithreaded factorizations, the dynamic voltage and frequency scaling (DVFS) technique was used. This technique allows the clock frequency to be scaled without changing the implementation. An experimental scalability evaluation was performed on an Intel Xeon Gold multicore machine, depending on the number of threads and the clock frequency. Our test results show that scalability in terms of the execution time expressed by the Speedup metric has values close to a linear function with an increase in the number of threads. In contrast, scalability in terms of the energy consumed expressed by the Greenup metric has values close to a logarithmic function with an increase in the number of threads. Both kinds of scalability depend on the clock frequency settings and the number of threads.**

I. Introduction

SCALABILITY is one of the main requirements to be taken into account when implementing parallel software on multicore machines, in particular for numerical algorithms involving many matrix calculations. The scalability feature allows an increasing number of threads to be used on a multicore machine in the hope that both time and energy efficiency will increase rather than degrade. The classical approach to scalability in parallel processing focuses on performance in terms of runtime. In this work, we want to study scalability in terms of two criteria, both the running time of the numerical algorithm and the energy consumption. The importance and need to consider multiple criteria in relation to scalability in parallel processing is shown in the work [8].

A distinction is made between two basic concepts related to scalability: scalability in the strong sense and scalability in the weak sense. In this paper we will only study scalability in the strong sense, that is, for a given problem size we will increase the number of threads. We focus on strong scalability because the parallelism available on modern machines will continue to increase.

An in-depth understanding of scalability in terms of execution time and energy consumption and the correlation between the two can allow the design of specific optimizations to reduce runtime and energy consumption for applications in different domains. In particular, it is important to study applications that make deliberate use of cache. Such applications usually come from the field of numerical linear algebra and involve matrix computations. Linear algebra is an important component of many numerical algorithms for various scientific and engineering problems. Over the years, BLAS (Basic Linear Algebra Subroutines) [6] has become the standard interface for linear algebra operations. One of the most popular BLAS packages is Math Kernel Library (MKL) [1]. The MKL library also contains implementations of matrix factorizations such as the LU factorization and the Cholesky factorization. The implementations of all factorizations are based on the BLAS library. The classical approach implemented in the MKL library for parallel matrix factorizations in cache-based systems uses fixed-size blocks that fit in the cache to evenly distribute the workload between threads. Currently, the MKL library tends to optimize runtime and does not take into account energy consumption savings. It is a well-known fact that reducing computation time usually implies energy savings, and is not the only reason for energy saving. Therefore, in order to improve the saving of energy of the algorithms from the MKL library without changing their implementation on multicore architectures, this work uses the dynamic voltage and frequency scaling technique DVFS [9].

The main contributions of this paper:

- a thorough empirical study of the runtime and energy consumption of multithreaded matrix factorizations (LU and Cholesky) concerning changing clock frequency and a selected number of threads;
- a scalability study using Speedup and Greenup metrics for varying numbers of threads for different clock frequencies;

The remainder of this article is organized as follows. In Section II, we discuss metrics such as Speedup and Greenup used to measure scalability in terms of the time and energy of parallel applications on multicore machines. In Section III we briefly review the LU and Cholesky algorithms. In Section IV, we present the test methodologies and experimental evaluation. Finally, in Section V, we conclude and make suggestions for future work.

II. Metrics

Energy consumption is the product of runtime and power consumed. Energy can be saved in various ways, e.g. by shortening runtime, reducing power consumption, or both, extending time but reducing power consumption more or vice versa.

The Speedup metric is known in the literature and used to analyze performance in parallel programming between different code implementations. It is assumed that we have two implementations of the algorithm, one non-optimized (basic) code running in time T_B and the other optimized code running in time T_O. Speedup is defined as follows:

$$Speedup = \frac{T_B}{T_O}$$

In [2] Greenup is defined analogously to Speedup only in terms of energy consumption:

$$Greenup = \frac{E_B}{E_O}$$

where E_B is the total energy consumption of the non-optimized code and E_O is the total energy consumption of the optimized code.

III. Algorithms

We will briefly introduce the LU and Cholesky algorithms used to solve systems of linear equations. The LU factorization transform square nonsingular matrix A into a product of two matrices:

$$A = LU$$

where L and U are lower and upper triangular matrices respectively.

The Cholesky factorization is defined only for A being Hermitian and positive-definite and has a form:

$$A = LL^T$$

where L is a lower triangular matrix. In this article, we investigate the LAPACK [3] implementation of the LU factorization from MKL library, namely `dgetrfnpi` (LU) [5], `dpotrf` (Cholesky) routines. These implementations are based on BLAS and arise from the use of a multithreaded BLAS.

The total number of floating-point operations (add, multiply, divide) for the LU factorizations is equal approximately $\frac{2}{3}n^3$. The number of floating point comparisons for the LU factorization is equals 0. The number of floating-point operations in the Cholesky factorization is $\frac{1}{3}n^3$. The number n is the size of the factoring matrix A.

IV. Numerical experiment – methodology and results analysis

A. Methodology

We tested two versions of matrix factorization: LU and Cholesky. We tested all algorithms without parallelization (1 thread) and in parallelized versions for 10, 20, 30, and 40 threads.

TABLE I: LU factorization at 1.7GHz

Threads/ Frequency	Time[s]	Energy[J]	Performance	Efficiency
1/1.7	889.07	57216.78	26.426	0.411
10/1.7	90.72	10560.03	258.989	2.225
20/1.7	46.35	7724.50	506.902	3.042
30/1.7	32.65	6716.32	719.517	3.498
40/1.7	26.53	6169.10	885.598	**3.808**

TABLE II: Cholesky factorization at 2.0GHz

Threads/ Frequency	Time[s]	Energy[J]	Performance	Efficiency
1/2.0	403.73	26795.84	29.098	0.438
10/2.0	40.35	4570.90	291.150	2.570
20/2.0	21.24	3653.45	553.140	3.215
30/2.0	14.67	3375.74	801.009	**3.480**
40/2.0	13.26	3412.81	886.171	3.442

Our test dataset consists of a square matrix filled with double-precision values. The matrix has dimensions of nxn, where $n = 32786$. In other words, our test dataset comprises 1073741824 cells, amounting to a total data size of 8 GB. For all algorithm versions, we have adhered to a row-wise data arrangement. These algorithms have been implemented in C++, incorporating vectorization and parallel processing techniques.

In our experimental configuration, we utilized a computing platform featuring a contemporary multicore Intel(R) Xeon(R) Gold 5218R processor, boasting 40 cores and a clock speed of 2.1 GHz. Our system was powered by the Linux 4.18.0 kernel and ran on the AlmaLinux 8.4 operating system, with the Intel ICC version 2021.5.0 compiler.

The Linux kernel facilitates CPU performance scaling through the `CPUFreq` subsystem, comprising three layers: core, scaling drivers, and governors. The core of `CPUFreq` offers a universal code infrastructure and user interfaces for all platforms supporting CPU performance scaling. It establishes the foundational framework for the other components. Scaling drivers communicate with hardware, supplying scale managers with data on available P-states (or P-state ranges in some cases) and accessing platform-specific hardware interfaces to modify processor P-states as directed by scale masters. Governors execute algorithms for estimating the necessary CPU capacity, typically each manager employing a single, optionally customized scaling algorithm.

The default scaling driver and governor are automatically chosen, but advanced configurations can still utilize userspace tools like `cpupower`, `acpid`, laptop mode tools, or desktop GUI tools.

To modify clock frequencies, we employed `CPUfreq` with the `acpi_cpufreq` driver. By default, this driver follows the

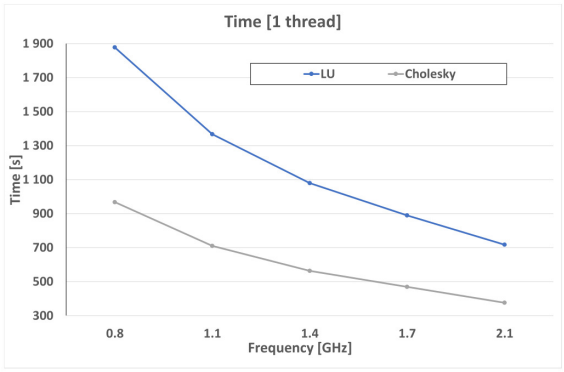

Fig. 1: Time execution for LU and Cholesky for 1 thread – left; Energy consumption of LU and Cholesky for 1 thread – right.

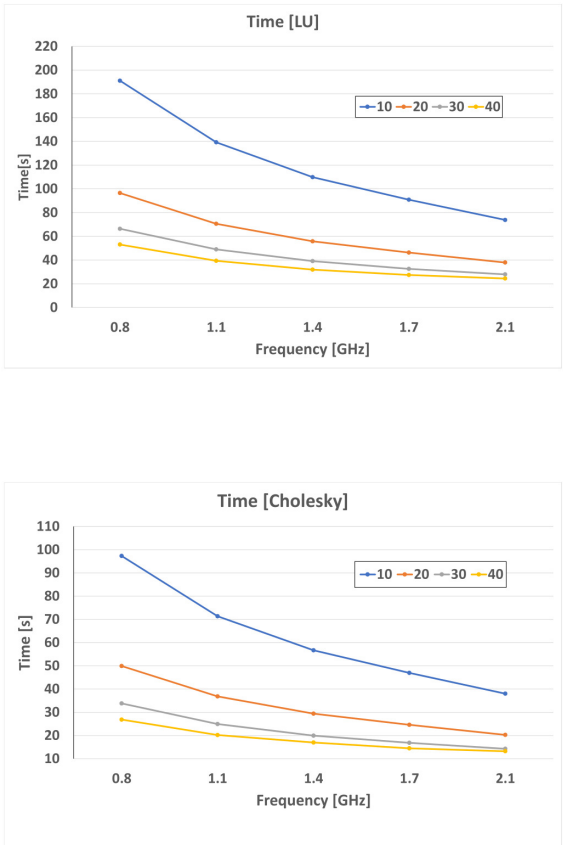

Fig. 2: Time execution for LU and Cholesky – left; Energy consumption of LU and Cholesky – right.

TABLE III: The most energy saving versions of algorithms

version of algorithm	energy consumption [J]	time [s]	waste of time [%]	energy saving [%]
LU ǀ 40 threads ǀ 1,7 GHz	6169.10	26.53	7.9	1.6
Cholesky ǀ 30 threads ǀ 2.0 GHz	3375.74	14.67	10.1	1.3

"conservative" governor, adjusting clock frequencies based on core load, selecting from available frequencies ranging from the minimum to the maximum supported by the processor.

We utilize the `cpupower` program to adjust the processor frequency limit's minimum and maximum values at a specific level, using the following commands:

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

for setting the minimum and maximum frequency limit values to 1.4 GHz. Executing these commands automatically switches the governor to `userspace`, enabling the configuration of a specific frequency. This frequency adjustment applies uniformly to all cores.

We do tests for the frequencies (P-states) available on our platform from 0.8 GHz to 2.1 GHz with step 0.1 GHz. We test first the following frequencies: 2.1 GHz, 1.7 GHz, 1.4 GHz, 1.1 GHz, and 0.8 GHz.

To assess the impact of algorithm optimizations on energy usage, we relied on data collected via the RAPL (Running Average Power Limit) interface, specifically designed for Intel processors. RAPL utilizes machine-specific records to continually monitor and regulate real-time energy consumption. In multi-socket systems, RAPL provides individual results for each socket or package, while also offering separate measurements for the memory modules (DRAM) linked to each socket. Starting with Haswell processors featuring fully integrated voltage regulators, RAPL's measurement accuracy has notably improved and meets acceptable standards [7]. Throughout our tests, we conducted measurements at 1-second intervals, considering the combined energy consumption of all sockets and their associated memory modules for analysis.

B. Time and energy consumption

In Fig. 1, we present the runtime and energy usage of individual algorithms when using a single thread. In Fig. 2, we display the same for parallel versions, i.e., for 10, 20, 30, and 40 threads. Notably, in the case of a single thread, the Cholesky factorization outperforms the LU factorization in terms of both time and energy consumption. This disproportion in performance is evident in Fig. 2, where we have different the y-axis scales to accommodate the dissimilarities.

In Fig. 2, we observe that reducing the clock frequency leads to an increase in runtime across all scenarios, while increasing the number of threads consistently reduces runtime. Thus, for our architecture, utilizing 40 threads at a frequency of 2.1 GHz proves to be the optimal choice in terms of time efficiency.

However, when considering energy consumption, a lower clock frequency, such as 1.7 GHz, can be advantageous in certain instances. This reduction in energy usage is evident for non-parallelized algorithm versions (approximately 19%) and for parallelized versions across all cases with 10 threads (3% for LU and 9% for Cholesky). Additionally, for 20 threads, a decrease in energy consumption is noticeable when employing 1.7 GHz with the Cholesky factorization (5%) and even for 40 threads with the LU factorization (1.6%). Lowering the clock frequency beyond 1.4 GHz does not yield any significant energy benefits.

Furthermore, we observe that energy consumption decreases as the number of threads used for calculations increases, with one exception: the Cholesky algorithm at 2.1 GHz. In this particular case, the algorithm is 1.2% more energy-efficient at 30 threads compared to 40 threads. A similar situation is observed at 2.0 GHz and 1.9 GHz (Fig. 3).

In response to the observed energy reduction when transitioning to a clock frequency of 1.7 GHz, we conducted additional experiments to explore the behavior of other frequencies within the range of 1.4 GHz to 2.1 GHz. The outcomes are depicted in Fig. 3. Subsequent tests indeed validated the presence of a localized energy consumption minimum at the 1.7 GHz frequency even for the LU algorithm executed with 40 threads.

The Table I and Table II show the test results for the frequencies at which we observe decreases in energy consumption for LU and Cholesky factorizations, respectively. The highest efficiency is achieved with LU at 1.7 GHz (Table I) running on 40 threads and in the case of Cholesky at 2.0 GHz (Table II) on 30 threads. Table III displays a compilation of algorithm versions and clock frequencies that resulted in the lowest energy consumption across both factorizations. In the table, the first column outlines the algorithm and the chosen configurations, the second column presents energy consumption in Joules, and the third column indicates the algorithm's runtime. The fourth and fifth columns reveal the percentage increase in runtime and the percentage reduction in energy consumption, respectively, relative to the configuration that achieved the shortest runtime — which, for both factorizations, was 40 threads and a 2.1 GHz clock frequency. Traditionally, optimizing for both time and energy efficiency involves increasing the number of threads and elevating the clock frequency. However, the two factorizations examined here demonstrate exceptions to this rule. If prioritizing energy savings over runtime, alternative thread and clock settings can be considered. Our tests have identified that the most energy-efficient configuration is achieved with 40 threads and a reduced clock frequency of 1,7 GHz for LU, while for Cholesky, it is with 30 threads and a clock frequency reduced to 2.0 GHz.

C. Speedup and Greenup

The figures in Fig. 4 illustrate the Speedup (left column) and Greenup (right column) values for different clock frequencies,

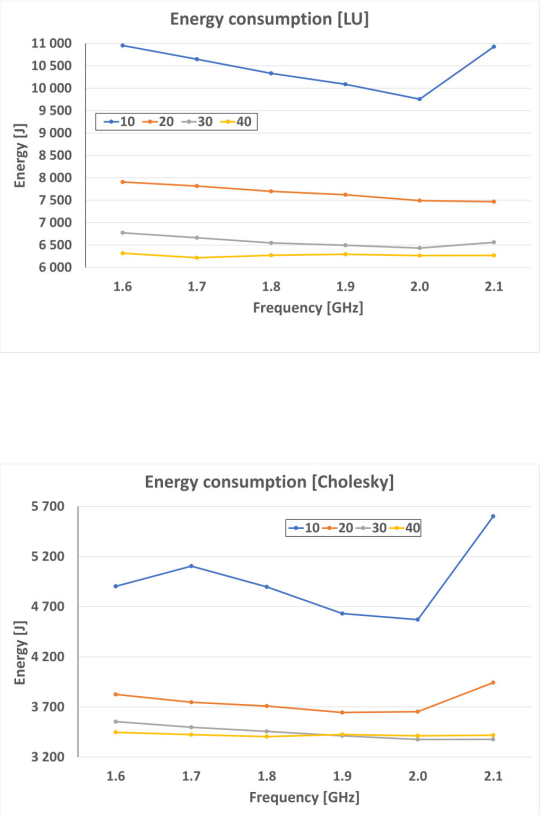

Fig. 3: Energy consumption of LU and Cholesky for frequencies from 1.6 GHz to 2.1 GHz.

derived from our experimental measurements. Each chart corresponds to one algorithm, either LU or Cholesky. The x-axis represents the number of active threads, while the y-axis displays the Speedup or Greenup values. To enhance reference, we've indicated the maximum expected Speedup (linear with the number of threads p) and Greenup (logarithmic with the number of threads p) with a dashed black line in the charts.

Across all two algorithms, it's evident that as the number of threads increases, irrespective of the clock frequency, the runtime improvement outpaces the reduction in energy consumption (time decreases more rapidly than energy consumption). For LU, Speedup approaches linearity for all frequencies, with deviations from the maximum expected value increasing as the thread count rises. Regarding Speedup, the 0.8 GHz frequency yields the most favorable results, while 2.1 GHz performs the poorest.

In general, the Greenup plot deviates further from the maximum expected value compared to Speedup, confirming our observations. For the algorithms we tested, similar to Speedup, the highest Greenup values are achieved at 0.8 GHz, while the lowest values, differing from the Speedup scenario, occur at 2.0 GHz. Notably, the 2.1 GHz frequency, which yielded the lowest Speedup values, still results in relatively high Greenup values (as seen in the purple line in the chart).

In our architecture, we observed the relationship:

$$Greenup \leq \alpha \log_2(Speedup)$$

where $\alpha > \beta$, and in our specific case, β falls within the interval $(2.84; 2.85)$. This leads to a research question: What is the value of α for other architectures?

V. Conclusion

This study explores scalability in relation to execution time and energy consumption for two matrix factorizations (LU and Cholesky) derived from the MKL library. To minimize energy consumption in these factorizations, we employed the DVFS technique. This approach allowed us to adjust clock frequency settings at the operating system level without modifying the implementation code.

We examined the impact of two parameters, clock frequency, and the number of threads, on execution time and energy consumption on a multicore machine. Execution time consistently decreases when using the highest clock frequency and the maximum number of threads for both factorizations. However, the same cannot be said for energy savings, as it varies based on the number of threads and clock frequency less regularly (see Table III).

Speedup and Greenup values increase with an expanding number of threads and typically decrease with a lower clock frequency. Experimental results reveal that Speedup values consistently surpass Greenup values, sometimes reaching up to 74% higher for specific combinations of clock frequency and thread count.

A deeper analysis of the research results, extended by tests of the LU factorization algorithm with pivoting and a study of the correlation between operation time and energy consumption using the Powerup and EDP metrics, is presented in [4]. Future research will address poor scalability by examining its impact on execution time and energy consumption in various multi-core machines and applications. Poor scalability entails keeping the problem size per processor constant while adding more computational units. Additionally, a key aspect to investigate is the correlation between strong and weak scalability regarding energy consumption.

References

[1] Intel Math Kernel Library, 2014. http://software.intel.com/en-us/articles/intel-mkl/.

[2] S. Abdulsalam, Z. Zong, Q. Gu, and Q. Meikang. Using the greenup, powerup, and speedup metrics to evaluate software energy efficiency. In *2015 Sixth International Green and Sustainable Computing Conference (IGSC)*, pages 1–8, 2015. 10.1109/IGCC.2015.7393699.

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK users' Guide. Society for Industrial and Applied Mathematics*. SIAM, 1999. 10.1137/1.9780898719604.

[4] B. Bylina and M. Piekarz. Time–energy correlation for multithreaded matrix factorizations. *Energies*, 16, 08 2023. 10.3390/en16176290.

[5] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997. 10.1137/1.9781611971446.

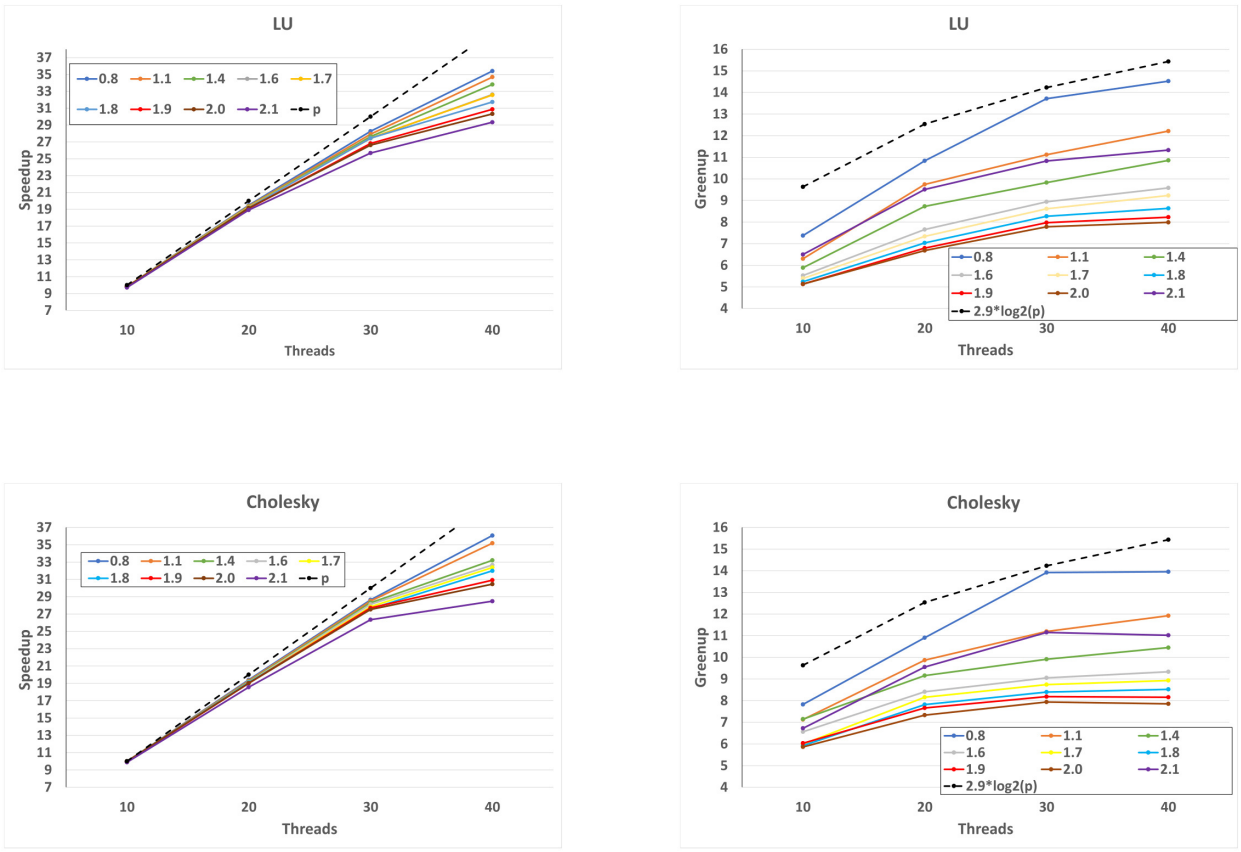

Fig. 4: Speedup for LU and Cholesky – left; Greenup of LU and Cholesky – right (p denotes number of threads).

[6] J. Dongarra, J. DuCroz, I. S. Duff, and S. Hammarling. A set of level-3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Software*, 16:1–28, 1990. 10.1145/77626.79170.

[7] 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)*, 3, 01 2018. 10.1145/3177754.

[8] Y. Ngoko and D. Trystram. Scalability in parallel processing. In S. K. Prasad, A. Gupta, A. L. Rosenberg, A. Sussman, and C. C. Weems, editors, *Topics in Parallel and Distributed Computing, Enhancing the Undergraduate Curriculum: Performance, Concurrency, and Programming on Modern Platforms*, pages 79–109. Springer, 2018. 10.1007/978-3-319-93109-8_4.

[9] M. Weiser, B. Welch, A.J. Demers, and S. Shenker. Scheduling for reduced cpu energy. *1st OSDI*, pages 13–23, 11 1994.