# Energy Efficiency Analysis of Compiler Optimizations on the SPEC CPU 2017 Benchmark Suite

Norbert Schmitt
University of Würzburg
Chair of Software Engineering
norbert.schmitt@uni-wuerzburg.de

James Bucek
Hewlett Packard Enterprise
SPEC CPU Committee Vice-Chair
james.bucek@hpe.com

Klaus-Dieter Lange
Hewlett Packard Enterprise
SPECpower Committee Chair
klaus.lange@hpe.com

Samuel Kounev
University of Würzburg
Chair of Software Engineering
samuel.kounev@uni-wuerzburg.de

## ABSTRACT

The growth of cloud services leads to more and more data centers that are increasingly larger and consume considerable amounts of power. To increase energy efficiency, both the actual server equipment and the software themselves must become more energy-efficient. It is the software that controls the hardware to a considerable degree. In this work-in-progress paper, we present a first analysis of how compiler optimizations can influence energy efficiency. We base our analysis on workloads of the SPEC CPU 2017 benchmark. With 43 benchmarks from different domains, including integer and floating-point heavy computations executed on a state-of-the-art server system for cloud applications, SPEC CPU 2017 offers a representative selection of workloads.

## CCS CONCEPTS

• **Hardware → Enterprise level and data centers power issues**; • **Software and its engineering → Software development techniques**.

## KEYWORDS

energy efficiency, compiler optimizations, SPEC CPU 2017, performance, benchmark

**ACM Reference Format:**
Norbert Schmitt, James Bucek, Klaus-Dieter Lange, and Samuel Kounev. 2020. Energy Efficiency Analysis of Compiler Optimizations on the SPEC CPU 2017 Benchmark Suite. In *ACM/SPEC International Conference on Performance Engineering Companion (ICPE '20 Companion), April 20–24, 2020, Edmonton, AB, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3375555.3383759

## 1 INTRODUCTION

Cloud computing is a significant line of business with high growth rates, leading to growing data centers. According to a study of the NRDC from 2014, data centers need large amounts of energy with an estimated 140 billion kilowatt-hours annually by 2020 [18]. Yet, cloud data centers can be made more efficient. For instance, by intelligently placing or consolidating the workload [11] or minimizing the number of resources to satisfy demands through auto-scaling. The server hardware has also become more efficient with dynamic voltage and frequency scaling (DVFS) and different C-states.

While the server hardware consumes the energy, it is the software that, in most parts, controls how the server behaves, indirectly controlling how much energy the equipment uses. Initial measurements on storage servers show that the running workload influences the power consumed by a server [13]. Nevertheless, energy efficiency is sometimes seen as identical to the energy efficiency of the hardware. Yet, Capra et al. show that two different ERP software systems have little performance differences (+5%) but deviate widely in energy consumption (+50%) [3]. For a fair comparison of software, it must perform the same task but with a different binary. One option to achieve this is by using compiler optimizations to improve performance or binary size without changing functionality.

The main contribution of this work is a first work-in-progress analysis of compiler optimizations on the SPEC CPU 2017 benchmarks. With 43 benchmarks from different domains, organized in four suites with integer and floating-point heavy computations, it has a large variety of workloads that we see as representative. Hebbar et al. showed that different compilers, GNU and Intel compiler, do influence the energy efficiency measured by the SPEC CPU 2017 on a desktop system through better utilization of the hardware [6]. In contrast, we take a look at which factors, application domain, or programming language, could be susceptible to compiler optimizations in terms of a change in energy efficiency. This work will act as a basis for future work on which compiler optimizations impact different applications running on modern state-of-the-art servers.

The remainder of this paper is structured as follows: First, we review related work in Section 2. Afterward, we describe the SPEC CPU 2017 in Section 3, followed by our analysis in Section 4. Finally, the paper is wrapped up in Section 5.

## 2 RELATED WORK

Compilers usually target performance instead of energy efficiency with the available optimizations. Hence, to optimize for energy efficiency, the correct optimization settings must be known [16].

*ICPE '20 Companion, April 20–24, 2020, Edmonton, AB, Canada*

ACM ISBN 978-1-4503-7109-4/20/04…$15.00
https://doi.org/10.1145/3375555.3383759

The tool Socrates from [4] not only tries to find suitable compiler optimizations and can change the number of used threads, but also weaves in code for tuning the software to different targets, like power consumption or performance. Multiple binaries are compiled and profiled to then optimize at runtime. Nobre et al. [15] is changing the order of performance optimizations during compilation. They have shown that while an increase in energy efficiency is possible, the sequence of optimization is not independent of the application. By looking at compiler optimizations for multi-dimensional signal processing and video applications on embedded devices, Kandemir et al. [12] targeted six specific loop optimizations. Based on five applications and simulation, they have achieved mixed results, and most optimizations increased the power consumed in the embedded system. Hsu et al. [10] insert instructions to control the Dynamic Voltage Scaling (DVS). The approach selects program parts suitable for running with lower voltage and frequency without degrading performance above a user-selectable value. They show in their work that energy savings of up to 28% are possible at a performance degradation of 5%, increasing energy efficiency. Gheorghita et al. argue that iterative compilation, generating different versions of source code with transformation patterns, can be used to optimize for energy efficiency for mobile embedded systems [5]. Trying to find the Pareto optimal set of compiler flags for performance, Hoste and Eeckhout used an evolutionary algorithm to solve the multi-objective optimization problem that is adaptable to energy efficiency [9]. Martins et al. used clustering to find the best order of optimization settings for performance on a soft microprocessor inside an FPGA [14]. In our work, expert developers, instead of heuristics and machine learning, aim to maximize performance through carefully picked optimization settings. Our analysis is also broader by using a large variety of workloads instead of limiting itself to specific use cases. Most works in optimizing energy efficiency with compiler options aim at embedded systems, such as Pallister et al. [16]. We use a state-of-the-art server for cloud data centers, a domain in which software energy efficiency is not the main focus during development.

## 3 THE SPEC CPU 2017 BENCHMARK SUITE

The SPEC CPU 2017 benchmark suite is a compute-intensive benchmark using real-world benchmarks of different code and problem sizes. It is designed to stress a system's processing unit, memory, and compiler [2]. SPEC CPU 2017 is developed and maintained by the SPEC OSG with defined submission, review, and publication procedures [1]. It provides specified run and reporting rules. The report contains not only the results of a run but also the necessary information to reproduce the results again. The report, therefore, includes the system under test (SUT), the SUT's operating system and configuration, and compiler flags for the benchmarks, as well as additional information, increasing reproducibility. SPEC CPU 2017 benchmarks come from a wide variety of application domains, ranging from pathfinding algorithms and compression to complex workloads such as artificial intelligence and simulation. It has a wide variety of implementation languages, consisting of compiled machine-independent languages, C, C++, and Fortran. With the variety of languages, different programming paradigms are covered, from functional programming and procedural programming to object-oriented programming. The wide range of LOC from only 1000 lines up to over 1.5 million shows a broad spectrum of programs in terms of source code size. This large variety makes the SPEC CPU 2017 relevant for our analysis to cover different real-world workloads that can be influenced by compiler optimizations.

The 43 benchmarks are organized in four suites. The SPECspeed suites, Integer, and Floating–Point, use a time-based metric of a single- or multi-threaded run and measures the time required to run one task at a time. The SPECrate suites, also Integer and Floating–Point, use a throughput metric, a work per unit of time, in which multithreading is not allowed according to the run rules. Without multithreading, the tester running the benchmark still can decide on the number of copies of each benchmark task that runs. The number of copies is normally the number of logical processors. Each suite reports a base and a peak result. Peak and base generally use different compiler settings. In the *base* run, for each language, C, C++, and Fortran, or a combination of those, a set of optimizations flags must be selected. All benchmarks using this language or combination must use the same settings. In a *peak* run, the compiler settings are different for each of the benchmarks listed. An overview of the benchmarks and suites is shown in [7].

The general setup for a SPEC CPU 2017 run, including power measurements, is outlined in Figure 1. It consists of the SUT running the benchmark and a controller system. The ambient temperature of the SUT is measured as well (but not reported) to achieve reliable and accurate power measurements as the temperature can influence the power consumption of the SUT. Both the power analyzer and the temperature sensor are connected to the SPEC PTDaemon tool [17], collecting the measurement data, and performing validity checks (e.g., temperature does not exceed operational limits of the SUT). PTDaemon must support and list the measurement devices obtaining data as an accepted measurement device [2, 8].

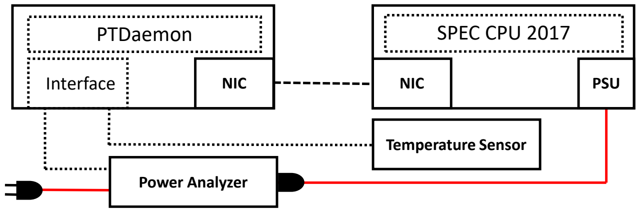


**Figure 1: General setup for SPEC CPU 2017 with power and temperature measurements.**

## 4 ANALYSIS

In our analysis, we first describe our SUT and setup for the SPEC CPU 2017 benchmark runs. We then compare the *base* and *peak* benchmark runs for the SPECrate suites described in Section 3.

### 4.1 System Under Test and Setup

We use an HPE ProLiant DL385 Gen10 server as the state-of-the-art SUT. The SUT is equipped with two AMD EPYC 7702 CPUs with a total of 128 cores and 256 threads. Each is configured with a 2.00 GHz base and 3.35 GHz maximum clock. The memory setup is 1 TB, organized in 16 modules with 64 GB each. The OS is a SUSE Linux

Enterprise Server 15 SP1 with kernel version 4.12.14-195-default. Binaries are generated with the AOCC 2.0.0 compiler. The optimization flags for the official result that we are using can be found in the SPEC CPU 2017 result database[1], together with additional information on the SUT and test environment. The binaries are cross-compiled on a Fedora 26. According to the SPEC CPU 2017 run and reporting rules, each benchmark was executed three times in *base* and *peak* configurations.

Each system's energy consumption consists of two parts, the static energy, and the dynamic energy. The dynamic energy is due to the software running. The static energy for our case is equal to the idle power times the time-to-result of the benchmark. For energy efficiency, only the dynamic energy is considered as the static energy is not affected by the running software. Hence, we subtract the idle energy from the total energy consumption. We define energy efficiency as the ratio of work to the energy consumed. For the throughput metric of the SPECrate suites, the amount of work is the number of copies for a benchmark run. We average all three runs to compute the energy efficiency. It expresses the number of copies executed per 1000 Joule.

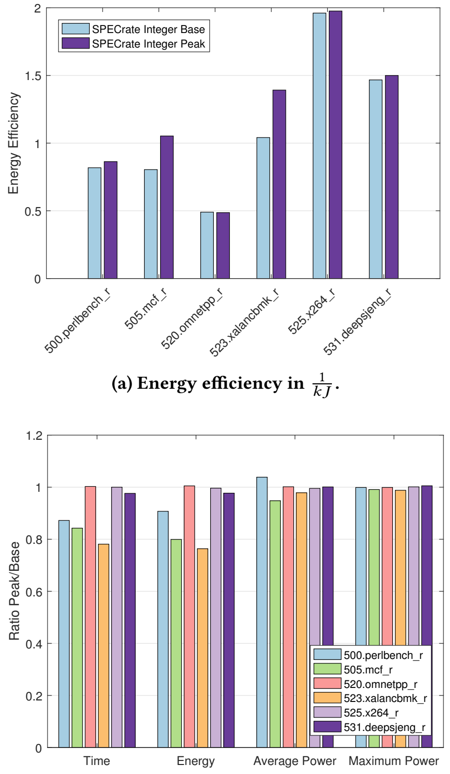
For our analysis, we also group the benchmarks into four application domains. The four categories are *language transformation*, *artificial intelligence*, *modeling and simulation*, and *others* that do not fit in the mentioned categories. The grouping allows us to analyze if a particular application domain is susceptible to compiler optimizations. Each group consists of at least three benchmarks.

Benchmarks that reported *base* as *peak* values are omitted because they used identical compiler settings. For the Integer suite, 502.gcc_r, 541.leela_r, 548.exchange_r and 557.xz_r, and for the Floating–Point suite, the 507.cactuBSSN_r, 519.lbm_r and 527.cam4_r benchmarks are excluded.

## 4.2 Energy Efficiency Analysis

Figure 2a shows the energy efficiency for the evaluated benchmarks in $\frac{1}{kJ}$. While the energy efficiency for the 500.perlbench_r, 505.mcf_r, 523.xalancbmk_r and 531.deepsjeng_r benchmarks exhibit a visible increase, others (520.omnetpp_r, 525.x264_r and 531.deepsjeng_r) do not. The performance for the 500.perlbench_r has 15% less runtime (899.33s for the *base* run compared to 784.33s for the *peak* run) but only an increase in the energy efficiency of 5.5% due to an increase in average power. The higher average power partially negates the benefit in energy efficiency that results from a faster time-to-result shown in Figure 2b. The Floating–Point suite results also show similar behavior but are not shown for brevity.

We take a look at the benchmarks that display an improvement in energy efficiency and their implementation language. From 23 benchmarks in the Floating-Point and Integer suites that ran, 7 are excluded due to reporting *peak* as *base* values (see Section 4.1). From the remaining 16 benchmarks, 12 achieved a higher energy efficiency in the *peak* runs, listed in Table 1. Of those 12, only one is written in Fortran. We, therefore, suspect that C or C-like languages can be optimized better. We ran Fisher's exact test (see Table 2) with the null hypothesis $H_0$ that both C-like and functional languages are equally likely to show better energy efficiency. We selected Fisher's exact test in favor of the $\chi^2$ test due to the small number of observations. We must reject $H_0$ in favor of the alternative hypothesis at the 5% significance level with $p = 1.57 \cdot 10^{-2}$. $H_0$ cannot be rejected at the 1% level.



**(a) Energy efficiency in $\frac{1}{kJ}$.**

**(b) Relative comparison of *base* and *peak* results.**

**Figure 2: SPECrate Integer suite *base* and *peak* comparison.**

**Table 1: Implementation language of efficient benchmarks. Three benchmarks are implemented in two languages and are counting towards each implementation language.**

| Implementation | Benchmarks | | |
|---|---|---|---|
| Language | More Efficient | Total | Percentage |
| C | 8 | 8 | 100% |
| C++ | 6 | 7 | 85.7% |
| Fortran | 1 | 4 | 25.0% |

This outcome could have three reasons: 1) it is due to the compiler that allows fewer optimizations for Fortran programs, 2) it is the functional programming paradigm that provides an already energy-efficient programming style, or 3) the results are outliers. In all cases, additional observations should be made in the future to verify or reject our findings that the C-like languages are more susceptible to compiler optimizations in terms of energy efficiency.

We also test if the application area from which the benchmarks are taken can play a role in increasing energy efficiency. As mentioned in Section 4.1, we group each benchmark in one of four application domains. Table 3 lists the percentage of how many

[1] https://www.spec.org/cgi-bin/osgresults?conf=cpu2017

**Table 2: Fisher's exact test contingency table.**

| | Improved $eff$ | | |
|---|---|---|---|
| Language | Yes | No | Sum |
| C-like | 14 | 1 | 15 |
| Functional | 1 | 3 | 4 |
| Sum | 15 | 4 | 19 |

**Table 3: App. domain and ratio of efficient benchmarks.**

| Application | Benchmarks | | |
|---|---|---|---|
| Domain | More Efficient | Total | Percentage |
| Language Transf. | 2 | 2 | 100% |
| Modelling and Sim. | 3 | 7 | 42.8% |
| Artificial Intelligence | 1 | 1 | 100% |
| Others | 6 | 6 | 100% |

benchmarks that reported a higher energy efficiency in *peak* came from which application area. The results suggest that modeling and simulation workloads cannot be optimized well by a compiler. Yet, it must be taken into consideration that the application areas are not equally distributed.

Conclusively, the impact of the application area needs additional measurements to be able to reach a clear verdict. The relation of implementation language to energy efficiency already shows promising results towards improving the energy efficiency for C-like languages, but we also recommend additional measurements.

## 5 CONCLUSION AND FUTURE WORK

Conserving energy is an important aspect of data centers that needs to be addressed and is achievable in two different ways, either by hardware or by software. We selected the SPEC CPU 2017 performance benchmark on a state-of-the-art server, to see if and what characteristics of software plays a role in using compiler optimizations for energy efficiency. SPEC CPU 2017 allowed us to compare compiler optimizations tuned by experts to specific applications and the particular programming language and evaluate their impact. While the application domain did not show an obvious connection to energy efficiency, we would argue that further measurements on different application areas should be made to reach an answer. Other options for improving the analysis and evaluation of the application domain could be an altered grouping of benchmarks. The proposed groups could be too skewed towards modeling and simulation, the largest group of all four. Investigating the programming language showed more promising results. A first test concluded that C and C-like languages are more prone to achieve better energy efficiency and performance through compiler optimizations. Still, we suggest additional measurements that also could include other functional languages. We propose to identify if different compiler optimizations impact different programming paradigms, languages, and application domains in contrasting ways. Finally, we further aim to research if an increase in energy efficiency through software optimizations can be hardware independent.

## ACKNOWLEDGEMENT

The authors wish to acknowledge current and past members of the SPECpower Committee, SPEC CPU Committee, and SPEC Research Power Working Group who have contributed to the design, development, testing, and overall success of SPEC CPU 2017. SPEC, SPECrate, SPECspeed, SPEC CPU, SPECpower, and PTDaemon are registered trademarks of the Standard Performance Evaluation Corporation. All rights reserved; see spec.org as of 12/12/2020.

## REFERENCES

[1] Walter Bays and Klaus-Dieter Lange. 2012. SPEC: Driving Better Benchmarks. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE '12)*. ACM, New York, NY, USA, 249–250. https://doi.org/10.1145/2188286.2188327

[2] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. 2018. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18)*. ACM, New York, NY, USA, 41–42. https://doi.org/10.1145/3185768.3185771

[3] Eugenio Capra, Chiara Francalanci, and Sandra A. Slaughter. 2012. Is software "green"? Application development environments and energy efficiency in open source applications. *Information and Software Technology* 54, 1 (2012), 60 – 71. https://doi.org/10.1016/j.infsof.2011.07.005

[4] D. Gadioli, R. Nobre, P. Pinto, E. Vitali, A. H. Ashouri, G. Palermo, J. Cardoso, and C. Silvano. 2018. SOCRATES - A seamless online compiler and system runtime autotuning framework for energy-aware applications. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1143–1146. https://doi.org/10.23919/DATE.2018.8342183

[5] Stefan Valentin Gheorghita, Henk Corporaal, and Twan Basten. 2005. Iterative compilation for energy reduction. *Journal of Embedded Computing* 1, 4 (2005), 509–520.

[6] Ranjan Hebbar SR and Aleksandar Milenković. 2019. SPEC CPU2017: Performance, Event, and Energy Characterization on the Core i7-8700K. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*. ACM, 111–118.

[7] John Henning. 2019. *SPEC CPU 2017® benchmarks*. Retrieved September 26, 2019 from https://www.spec.org/cpu2017/Docs/overview.html#benchmarks

[8] John Henning. 2019. *SPEC CPU 2017® run and reporting rules*. Retrieved October 3, 2019 from https://www.spec.org/cpu2017/Docs/runrules.html

[9] Kenneth Hoste and Lieven Eeckhout. 2008. Cole: compiler optimization level exploration. In *Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization*. ACM, 165–174.

[10] Chung-Hsing Hsu and Ulrich Kremer. 2003. The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI '03)*. ACM, New York, NY, USA, 38–48. https://doi.org/10.1145/781131.781137

[11] Yichao Jin, Yonggang Wen, and Qinghua Chen. 2012. Energy Efficiency and Server Virtualization in Data Centers: An Empirical Investigation. In *2012 IEEE Conference on Computer Communications Workshops*. 133–138.

[12] M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye. 2000. Influence of Compiler Optimizations on System Power. In *Proceedings of the 37th Annual Design Automation Conference (DAC '00)*. ACM, New York, NY, USA, 304–307. https://doi.org/10.1145/337292.337425

[13] Klaus-Dieter Lange. 2009. The Next Frontier for Power/Performance Benchmarking: Energy Efficiency of Storage Subsystems. In *Proceedings of the 2009 SPEC Benchmark Workshop on Computer Performance Evaluation and Benchmarking*. Springer-Verlag, Berlin, Heidelberg, 97–101. https://doi.org/10.1007/978-3-540-93799-9_6

[14] Luiz GA Martins, Ricardo Nobre, Joao MP Cardoso, Alexandre CB Delbem, and Eduardo Marques. 2016. Clustering-based selection for the exploration of compiler optimization sequences. *ACM Transactions on Architecture and Code Optimization (TACO)* 13, 1 (2016), 8.

[15] Ricardo Nobre, Luís Reis, and João M. P. Cardoso. 2018. Compiler Phase Ordering as an Orthogonal Approach for Reducing Energy Consumption. *CoRR* abs/1807.00638 (2018). arXiv:1807.00638 http://arxiv.org/abs/1807.00638

[16] James Pallister, Simon J Hollis, and Jeremy Bennett. 2013. Identifying compiler options to minimize energy consumption for embedded platforms. *Comput. J.* 58, 1 (2013), 95–109.

[17] SPEC. 2012. *SPEC PTDaemon Design Document*. Retrieved October 7, 2019 from https://spec.org/power/docs/SPEC-PTDaemon_Design.pdf

[18] J. Whitney and P. Delforge. 2014. Data center efficiency assessment. https://www.nrdc.org/sites/default/files/data-center-efficiency-assessment-IP.pdf.