Comparative Analysis of Compiler Efficiency: Energy Consumption Metrics in High-Performance Computing Domains

Erick Damasceno[1], Fellipe Queiroz[1], Luan Siqueira[1], Thiago Rodrigues[1], Marcos Amaris [1]

[1]Universidade Federal do Pará
Programa de Pós-Graduação em Computação Aplicada - PPCA
Tucuruí - Pará
{erick.silva, fellipe.queiroz, luan.siqueira,
thiago.rodrigues}@tucurui.ufpa.br, amaris@ufpa.br

Resumo. *Este estudo apresenta uma análise comparativa abrangente da eficiência dos compiladores, com destaque para as métricas de consumo de energia em diversos domínios da computação de alto desempenho. Através de uma avaliação rigorosa do desempenho do GCC, do Clang e do ICC, a investigação visa elucidar quais os compiladores que se destacam em áreas específicas, fornecendo assim informações valiosas para a seleção estratégica destas ferramentas com base nos requisitos únicos de várias tarefas computacionais. Os resultados revelam que, entre a energia total consumida durante os cálculos, o GCC foi responsável por 33,23%, o Clang por 36,01% e o ICC por 30,76%. Notavelmente, o ICC demonstrou uma eficiência energética superior, sendo 7,43% mais eficiente que o GCC, enquanto o Clang foi 8,35% menos eficiente. Esses resultados ressaltam a importância crítica de selecionar o compilador apropriado para otimizar a eficiência energética em ambientes de computação de alto desempenho.*

Abstract. *This study presents a comprehensive comparative analysis of compiler efficiency, with a focus on energy consumption metrics across diverse domains of high-performance computing. By rigorously evaluating the performance of GCC, Clang, and ICC, the research aims to elucidate which compilers excel in specific areas, thereby providing valuable insights for the strategic selection of these tools based on the unique requirements of various computational tasks. The findings reveal that, among the total energy consumed during the computations, GCC accounted for 33.23%, Clang for 36.01%, and ICC for 30.76%. Notably, ICC demonstrated superior energy efficiency, being 7.43% more efficient than GCC, while Clang was 8.35% less efficient. These results underscore the critical importance of selecting the appropriate compiler to optimize energy efficiency in high-performance computing environments.*

1. Introduction

Parallel and High-Performance Computing (HPC) is not solely dependent on hardware architectures and algorithm design methods but also on the programming languages and, critically, the compilers used. A compiler is a tool that translates programs written in high-level programming languages into low-level versions suitable for processing by assemblers and linkers. Compilers play a pivotal role in this process by translating high-level

programs into machine-level instructions, directly influencing the efficiency and behavior of applications [Daniel and Page 2009].

In the context of programming for parallel computers, the language C is often the imperative language, complemented by various parallel frameworks as OpenMP and Message Passing Interface, as pointed out by [Hatcher et al. 1991]. With the rapid advancement of high-performance software and hardware, there has been an increasing need to accurately measure the efficiency of these platforms. Benchmarking tools have been developed for this purpose, providing a standardized method to test and evaluate system performance. These tools are essential for assessing a system's ability to solve complex problems and perform realistic simulations.

In the current era, energy efficiency and environmental impact have become paramount concerns. This study seeks to evaluate the performance of various benchmarks within the context of energy consumption and execution time. Specifically, several benchmarks from the Rodinia suite were tested across three C compilers: GCC [2.1.2], CLANG [2.1.3], and ICC [2.1.4]. The goal was to compare their performance and energy efficiency, providing insights into how compiler choice can influence the overall effectiveness of HPC systems.

The main objective of this work is to analyze and compare the energy efficiency and execution time of High-Performance Computing applications using various C compilers and the OpenMP library on multicore machines. For this, we must conduct a thorough collection of energy consumption data for each selected benchmark, with three separate measurements for each, using different compilers. This approach will enable a detailed comparative analysis of how compiler choice affects energy consumption. We will perform a comparative evaluation of benchmark efficiency with each compiler to determine how each performs under various scenarios, revealing insights into their relative performance across different demands and benchmark characteristics. We will examine the correlation between the residual size of each binary generated after compilation and its impact on final performance. This analysis will assess the effect of compilation overhead on benchmark efficiency and overall performance, contributing to a deeper understanding of the factors influencing system performance.

This document is divided into 6 sections, as follows: Section 2 demonstrates the essential concepts of this research. Section 3 presents some relevant papers for understanding this work. Section 4 presents the methodology. Section 5 explains the results and finally Section 6 presents the conclusions and future work.

2. Theoretical Background

2.1. C Programming Language

In simpler terms, the C language is a versatile programming language initially developed for the UNIX operating system, and is not considered a high-level language. It has emerged as a valuable tool for real-time and computationally intensive tasks. Although there was historical preference for other languages, such as FORTRAN, for digital signal processing, the shift to C is deemed inevitable due to its superior advantages [Embree et al. 1991].

2.1.1. Open Multi-Processing

According to [Kiessling 2009], the OpenMP (Open Multi-Processing) Application Programming Interface (API), which was initially released in 1997, has become a standard for writing parallel applications with shared memory in C, C++, and Fortran. OpenMP offers the advantage of being easily integrated into existing serial code, enabling incremental parallelization. Additionally, it is widely used, highly portable, and particularly well-suited for multicore architectures, which are becoming increasingly common in everyday desktop computers.

2.1.2. GCC - GNU C Compiler

The GNU C Compiler (GCC), developed by Richard Stallman, originated within the context of the GNU Project in 1984. The project aimed to create a free software operating system similar to UNIX. Funded through donations to the Free Software Foundation, GCC has evolved from a C compiler into a versatile, multilingual tool supporting over 30 architectures and various programming languages, including Fortran, ADA, Java, and Objective-C [Gough and Stallman 2004]. This expansion has solidified GCC's role as a cornerstone in the development of open-source software, offering a robust and flexible compiler infrastructure that underpins numerous modern computing applications.

2.1.3. LLVM and CLANG

LLVM (Low-Level Virtual Machine) is a compiler framework designed to support the ongoing analysis and transformation of programs throughout their lifecycle. It provides a common code representation in the form of Static Single Assignment (SSA), coupled with a simple, language-independent type system, an instruction set for typed address arithmetic, and an efficient mechanism for implementing exception handling [Lattner and Adve 2004]. LLVM's modular design allows for the implementation of a wide range of optimizations and analyses, making it a powerful tool for both academic research and industrial applications.

Clang, serving as the C/C++ front-end for LLVM, parses source code, checks for errors, and generates an Abstract Syntax Tree (AST), which is subsequently converted into LLVM Intermediate Representation (LLVM IR). This IR undergoes various optimizations before being translated into machine code. Clang's library-based architecture enables the flexible combination of front-end components to meet diverse needs, simplifying the integration process for new developers and promoting innovation in compiler technology [Mishra et al. 2020]. Its design philosophy emphasizes modularity, performance, and compatibility, making Clang a popular choice for modern software development.

2.1.4. ICC - Intel C Compiler

The Intel C Compiler (ICC) is a suite of compilers for C and C++, developed by Intel Corporation, with support for Linux, Windows, and macOS. Available at no cost to academic institutions, ICC is tailored for x86 architecture, enabling efficient compilation and

performance optimization. It is distinguished by its comprehensive support for OpenMP 4.0 and its capability for automatic parallelization, which is particularly advantageous in symmetric multiprocessing environments. Beyond C and C++, ICC also supports Fortran, thereby extending its applicability to a broader range of scientific and engineering domains [Machado et al. 2017]. The compiler's advanced optimization techniques, including vectorization and loop unrolling, are crucial for achieving high performance on Intel processors.

2.2. Rodinia Benchmark Suite

The Rodinia Benchmark Suite is a widely used collection of benchmarks designed to evaluate the performance of parallel computing systems, encompassing multicore CPUs, GPUs, and multi-node environments. Developed by the University of Virginia, the suite includes a diverse array of benchmarks across domains such as physical simulations, image processing, and data analysis. Utilizing the Berkeley Dwarf Taxonomy, Rodinia provides a comprehensive set of benchmarks to assess the efficiency of parallel and distributed architectures, as well as to explore non-traditional memory hierarchies and optimization layers. Its primary objectives are to deliver a diverse benchmark suite, facilitate performance measurement, and advance research in parallel computing [Che et al. 2010]. Rodinia's benchmarks are instrumental in identifying bottlenecks and guiding the design of next-generation computing systems.

2.3. Profiling, Power API and Joule It

Power API is a programming interface that offers access to information and control over energy consumption and management in computing systems. It allows developers to interact with hardware components such as CPUs, GPUs, and DRAM to monitor and optimize energy usage. Power API provides detailed real-time data, which is essential for managing the power consumption of applications and components, particularly in mobile devices, servers, and data centers where energy efficiency is critical [Inria, University of Lille 2024]. Joule It complements this by offering fine-grained energy profiling capabilities, enabling developers to identify and mitigate energy hotspots within their code, thereby contributing to the development of greener, more sustainable computing technologies.

2.4. *Green Computing*

Green Computing, refers to the efficient and environmentally responsible use of computers and their resources. This approach encompasses various practices and strategies. With the widespread adoption of computational technologies, there has been an increasing demand for greater computing power, which, in turn, has led to higher energy consumption and, consequently, greater carbon dioxide (CO2) emissions into the environment. The environmental issues caused by increased CO2 emissions and the financial costs associated with energy consumption have driven research aimed at developing mechanisms and technologies for more efficient energy use. These mechanisms and technologies are collectively referred to as Green Computing [Williams and Curtis 2008].

3. Related Works

This section aims to present references to related works that provided the foundation for the development of this research. By exploring prior works and relevant studies, we

seek to establish a solid theoretical and contextual foundation for the present work. The literature review was conducted comprehensively, considering research and publications that address similar or directly related topics to the objectives of this investigation. We discussed the main contributions found in the literature, highlighting concepts, methodologies, and results that influenced the formulation of research questions, the definition of methodological approaches, and the interpretation of obtained results.

Modeling Power and Energy Usage of HPC Kernels [Tiwari et al. 2012] underscores the significance of computationally intensive kernels in the realm of High-Performance Computing (HPC) applications, as these kernels account for a substantial portion of execution time. The development of power and energy models for Central Processing Units (CPUs) and Dynamic Random-Access Memory (DIMMs) involves training Artificial Neural Networks (ANNs) on three widely used HPC kernels. These neural networks are trained with empirical data collected from the target architecture. The models use kernel-specific optimization parameters and hardware adjustments as inputs for predicting power consumption rates and energy usage of system components. The results show an average absolute error rate of less than 5.5% for three major kernels—matrix multiplication (MM), stencil computation, and LU factorization.

Seven Pillars to Achieve Energy Efficiency in High-Performance Computing Data Centers [Hussain et al. 2019] highlights that energy efficiency in intensive computing environments, such as data centers and HPC systems, has become a perennial concern due to its crucial relevance today. Energy-efficient design and ecological measures represent central challenges in HPC environments. However, current research focuses on practical methods for measuring energy utilization aimed at promoting Green Computing without exceeding resources and compromising performance. This work provides a comprehensive analysis of issues, challenges, and solutions related to energy consumption in data centers and HPC systems, focusing on the period from 2010 to 2016.

The study categorizes existing problems in energy efficiency faced by data centers, providing a broad view of the models adopted by each approach. The work has a dual contribution. Firstly, through this categorization, it seeks to offer a concise view of the underlying energy efficiency model adopted by each approach. Secondly, it proposes a seven-pillar framework for energy efficiency in HPC systems and data centers, representing an original contribution to the field.

Performance Assessment of OpenMP Constructs and Benchmarks Using Modern Compilers and MultiCore CPUs [Gawrych and Czarnul 2023] fits into the context of contemporary advances in Computer System Architecture, especially considering the increasing number of cores, cache memory expansion, and evolving architectures, along with the ongoing improvement of compilers. The demand for accurate assessments of workloads, frequently executed and representative, becomes imperative in this ever-changing scenario.

The primary metric explored in this article is execution speed, as the computational power of modern CPUs is predominantly derived from the efficient use of multiple cores. The experiments cover a variety of codes, including batch normalization, convolution, linear functions, matrix multiplication, prime number testing, and wave equation. These operations were executed with different compilers, such as GNU gcc, LLVM clang, icx, and icc, across four distinct systems, consisting of 1 or 2 sockets: namely, 1 x Intel

Core i7-5960X, 1 x Intel Core i9-9940X, 2 x Intel Xeon Platinum 8280L, and 2 x Intel Xeon Gold 6130.

By discussing and analyzing the performance achieved in different hardware and compiler configurations, this work contributes to a deeper understanding of the impact of architectural and software choices on computational efficiency.

SIMD Programming Using Intel Vector Extensions [Amiri and Shahbahrami 2020] explores the context of Single Instruction, Multiple Data (SIMD) extensions, one of the most significant capabilities of modern General-Purpose Processors (GPPs), aimed at enhancing application performance with minimal hardware modifications. Each GPP vendor, such as HP, Sun, Intel, and AMD, presents its own Instruction Set Architecture (ISA) and SIMD microarchitecture with distinct perspectives.

The goals of this work are threefold. Firstly, it offers a review of SIMD technology in general and Intel's SIMD extensions in particular. Secondly, it compares SIMD features of Intel technologies such as MMX, SSEs, AVX, and FMA in terms of ISA, vector width, and SIMD programming tools. Finally, it compares the performance of different auto-vectorizers and IPM approaches using Intel C++ (ICC), GNU Compiler Collection (GCC), and Low Level Virtual Machine (LLVM) by mapping and implementing some representative multimedia kernels in AVX and AVX2 extensions.

GCC vs. ICC Comparison Using PARSEC Benchmarks [Almomany et al. 2014] aligns with the goal of evaluating the impact of various compiler optimizations on program performance by using two renowned compiler suites: GNU C Compiler and Intel's C/C++ Compiler with PARSEC benchmarks. Compiler optimization involves adjusting a compiler's output to minimize or maximize specific attributes of an executable program. Optimization is often achieved through the activation of optimization flags.

This research explores the potential to enhance program performance through better utilization of existing architectural features, notably through compiler optimizations. The careful activation of these optimizations not only has the potential to improve program performance but also to reduce the need for costly updates and, consequently, the development costs of the system.

Assessing the effectiveness of compiler optimizations is a crucial component in the quest for more efficient and cost-effective programs. The choice between GNU C Compiler and Intel's C/C++ Compiler, as well as the specific selection of optimization flags, becomes fundamental to achieving results that not only benefit program performance but also optimize existing architectural resources.

By investigating the impact of these optimizations on specific benchmarks, this work contributes to a deeper understanding of how compiler optimization choices can significantly influence program performance in practical environments. This comparative analysis between widely used compiler suites and the use of recognized benchmarks adds valuable perspective to the existing body of knowledge, providing relevant insights for developing efficient and economically viable software.

mdspan in C++: A Case Study in the Integration of Performance Portable Features into International Language Standards [Hollman et al. 2019] highlights the

ubiquitous presence of multidimensional arrays in High-Performance Computing (HPC), underscoring their importance. However, their absence in the C++ language represents a recognized and enduring limitation for HPC use. This work refers to the design and implementation of *mdspan*, a multidimensional span proposal for inclusion in the C++23 standard. This proposal is heavily inspired by the work done in the Kokkos project, a high-performance C++ programming model used by various HPC institutions to prepare their codebases for exascale-class supercomputing systems.

The text describes the final design of *mdspan* after a five-year process to achieve consensus in the C++ community. It specifically explores how the design addresses some fundamental challenges of performance-portable programming, highlighting customization points that enable seamless extension to areas not currently covered by the C++ standard but critically important in modern heterogeneous computing.

A significant aspect emphasized in the text is the adaptation of the design to the needs of performance-portable programming, in addition to providing a high-quality implementation in its current form. It also includes various benchmarks to demonstrate the zero-overhead nature of the proposed modern design.

4. Methodology

4.1. Hardware Components

The system used in this study is powered by an Intel Core i7 5500U CPU, with a base frequency of 2.4 GHz, which can increase up to 3.0 GHz in turbo mode. This processor has two physical cores and four threads, and is built with the 14 nm lithography of the Haswell microarchitecture. The CPU includes a total of 4 MB of cache distributed across three levels (L1: 64 KB, L2: 256 KB, L3: 4 MB). Additionally, the system is equipped with 10 GB of DDR3L RAM operating at 1600 MHz.

It is important to highlight that the choice of this hardware for the tests was due to its availability and the system's suitability for the study's objectives. Although it is not the latest processor, the Intel Core i7 5500U offers a suitable balance between performance and energy efficiency.

4.1.1. Software Components

The operating system utilized for this research is Linux Ubuntu version 22.04.1 LTS. Three general-purpose compilers were selected for the study:

- **GCC**: Version 11.4.0;
- **ICC**: Version 2021.10.0 (released on 2023-06-09);
- **CLANG**: Version 11.4.0.

The benchmarks chosen from the Rodinia suite 2.2 for this study include:

- ***LU Decomposition***: Implementation of the LU decomposition, a linear algebra algorithm used for matrix factorization into a lower triangular matrix and an upper triangular matrix;
- ***Back Propagation***: A benchmark related to neural networks, specifically back-propagation, which is a fundamental technique in the supervised training of neural networks;

- ***Streamcluster***: A clustering algorithm used to analyze large datasets, identifying patterns and grouping similar data points;
- ***LavaMD***: Implementation of the Molecular Dynamics method, a common benchmark for simulating the behavior of particles in a system.

4.2. Data Collection

Data collection was automated using a Bash script that runs the benchmark binaries in conjunction with the "jouleit.sh" script, which monitors energy consumption using intel RAPL and records the results in a CSV file. Modifications were made to the source code to remove the terminal output lines, ensuring accurate data collection. In addition, five sets of data were collected for each benchmark and averaged to obtain a safe margin to analyze.

The data collection method established a maximum input value that did not overload the system, serving as an upper load limit, and used powers of two for the input values. This maximum value was halved to obtain the initial value, which was then incremented by 128 in each iteration until the maximum value was reached, resulting in 128 samples. This method is mathematically expressed as:

$$\frac{V_{\text{max}} - V_{\text{min}}}{128} = \text{Value} \tag{1}$$

This approach proved effective for three of the four selected benchmarks. However, for the LavaMD benchmark, only 90 samples could be collected due to its unique characteristics, but these were still sufficient for the analysis.

The data collection configuration was as follows:

- ***Back Propagation*** ranged from 16,777,216 to 33,554,432, with increments of 131,072 per iteration;
- ***LavaMD*** ranged from 1 to 90, with increments of 1 per iteration;
- ***LU Decomposition*** ranged from 16,384 to 32,768, with increments of 128 per iteration;
- ***Streamcluster*** ranged from 2,097,152 to 4,194,304, with increments of 16,384 per iteration.

This process yielded twelve CSV datasets, exemplified in Table [1], each containing an additional column labeled *INPUT* to facilitate controlled analysis of the collected data. Additionally, data on the sizes of the compiled binaries were collected for potential correlation analysis. This information is exemplified in Table [2].

Table 1. Sample CSV Data Generated by the Collection Script

CORE	CPU	DRAM	DURATION	UNCORE	EXIT CODE	INPUT
73647760	91968454	10552097	9812865	2503	0	512
27366324	34201390	3997121	3636782	1587	0	516
74480400	92856879	10615390	9855222	2990	0	520
31772136	39662802	4590259	4200268	1648	0	524
...	...	...	...	...	...	...

Source: Own work.

Table 2. Sample Data on Binary File Sizes

File Name	Size (KB)
lud_omp	21.828125
backprop	49.8359375
lavaMD	20.734375
sc_omp	40.671875

Source: Own work.

4.3. Data Analysis

To conduct the data analysis, a thorough cleaning and preprocessing of the collected data were performed to correct potential errors and adjust variable types, including converting microjoules to joules and milliseconds to seconds. Performance analysis was visualized through comparative graphs and parameterization of the results, providing a clear and detailed view of the data presented in this research.

Additionally, a percentage gain analysis was conducted using the binary data to investigate the relationships between energy consumption, performance, and binary size.

5. Results

5.1. Performance

The analysis of the performance data reveals that the Intel C++ Compiler (ICC) exhibits the highest efficiency, with a performance percentage of 30.76%. In comparison, Clang and GCC demonstrate performance percentages of 36.01% and 33.23%, respectively. These findings are visually represented in Figure 1.

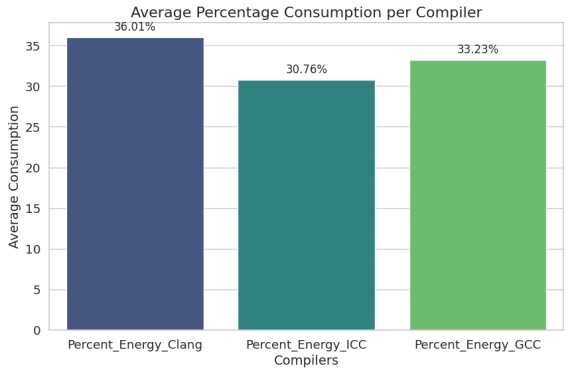

Figure 1. Percentage of Consumption by Compiler

In terms of the total time required to complete all benchmark iterations, ICC completed the process in 60 hours, 23 minutes, and 21 seconds. By contrast, GCC took 66 hours, 18 minutes, and 51 seconds, while Clang required 74 hours, 11 minutes, and 50 seconds. When considered in percentage terms, as illustrated in Figure 2, ICC accounted for 30.1% of the total time spent, GCC for 33%, and Clang for 36.9%.

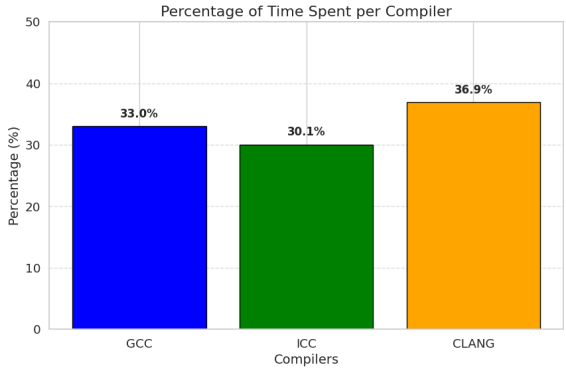

Figure 2. Time Spent by Compiler

5.2. Binary Size

Investigating the potential correlation between the size of generated binaries and compiler efficiency, it was observed that ICC tends to produce significantly larger binaries compared to GCC and Clang, as depicted in Figure 3.

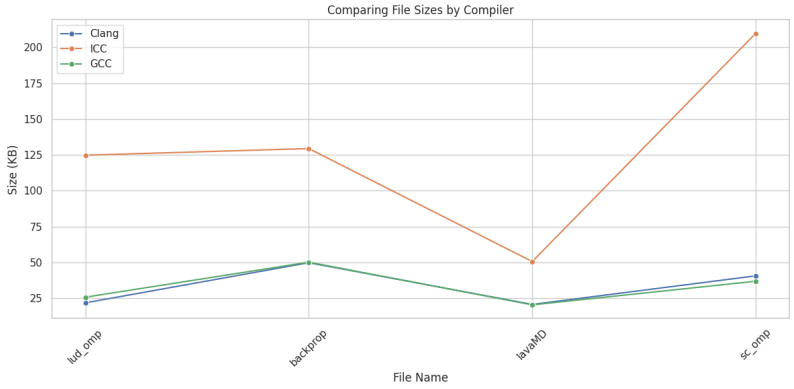

Figure 3. Binary Size (KB) by Compiler

When examining this comparison from a percentage perspective relative to GCC, ICC shows an average relative increase of 289.6% in binary size. This trend suggests that larger binaries might be associated with better performance, as indicated in Figure 4.

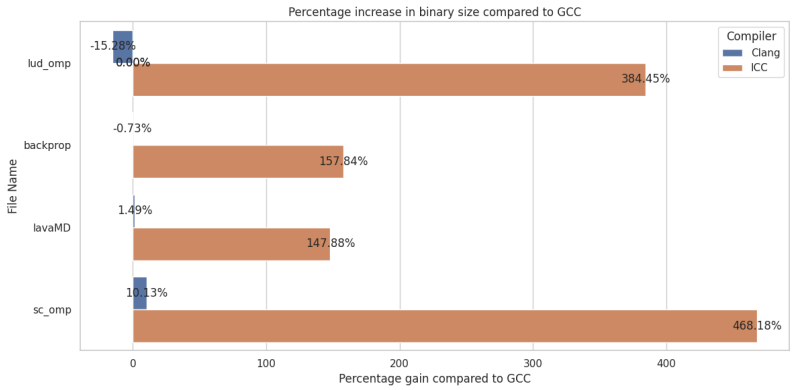

Figure 4. Percentage Increase in Binary Size (KB) Relative to GCC

6. Conclusions and Future Works

The results presented in this study indicate that the Intel C Compiler (ICC) exhibits superior efficiency in both energy consumption and total time required to complete the benchmarks. Specifically, ICC achieved a consumption percentage of 30.76% and a total execution time of 60 hours, 23 minutes, and 21 seconds. In comparison, the GNU Compiler Collection (GCC) and Clang demonstrated lower performance metrics. GCC consumed 33.23% of the total energy and required 66 hours, 18 minutes, and 51 seconds to complete the same benchmarks, while Clang exhibited the highest consumption at 36.01% and the longest execution time of 74 hours, 11 minutes, and 50 seconds. These findings underscore the advantages of ICC in terms of compilation efficiency and resource utilization.

Furthermore, it was observed that the binaries generated by ICC were significantly larger, with an average relative increase of 289.6% in binary size compared to those produced by GCC. This substantial increase in binary size may be correlated with the improved performance, suggesting that ICC employs optimization techniques that produce larger binaries but enhance execution efficiency. The trade-off between binary size and performance gain highlights the effectiveness of ICC's approach to optimizing compiled code, potentially offering a more favorable balance between resource consumption and execution time in high-performance computing scenarios.

In future studies, it is essential to utilize more advanced machines to continue this research. Additionally, exploring various hardware configurations is crucial, as different setups can yield valuable data for the evaluation conducted in this work, given that the selected benchmarks cover diverse areas. Furthermore, to gain deeper insights into the binaries produced by the compilers, reverse engineering the binaries to assembly language could provide additional information on how the generated size affects the energy consumption of these applications.

References

[Almomany et al. 2014] Almomany, A., Alquraan, A., and Balachandran, L. (2014). Gcc vs. icc comparison using parsec benchmarks. *IJITEE*, 4(7).

[Amiri and Shahbahrami 2020] Amiri, H. and Shahbahrami, A. (2020). Simd programming using intel vector extensions. *Journal of Parallel and Distributed Computing*, 135:83–100.

[Che et al. 2010] Che, S., Sheaffer, J. W., Boyer, M., Szafaryn, L. G., Wang, L., and Skadron, K. (2010). A characterization of the rodinia benchmark suite with comparison to contemporary cmp workloads. In *IEEE International Symposium on Workload Characterization (IISWC'10)*, pages 1–11.

[Daniel and Page 2009] Daniel and Page (2009). *Compilers*, pages 451–493. Springer London, London.

[Embree et al. 1991] Embree, P. M., Kimble, B., and Bartram, J. F. (1991). C language algorithms for digital signal processing.

[Gawrych and Czarnul 2023] Gawrych, B. and Czarnul, P. (2023). Performance assessment of openmp constructs and benchmarks using modern compilers and multi-core cpus. In *2023 18th Conference on Computer Science and Intelligence Systems (FedCSIS)*, pages 973–978. IEEE.

[Gough and Stallman 2004] Gough, B. J. and Stallman, R. (2004). *An Introduction to GCC*. Network Theory Limited.

[Hatcher et al. 1991] Hatcher, P. J., Quinn, M. J., Lapadula, A. J., Anderson, R. J., and Jones, R. R. (1991). Dataparallel c: A simd programming language for multicomputers. In *The Sixth Distributed Memory Computing Conference*, pages 91–92. IEEE CS.

[Hollman et al. 2019] Hollman, D. S., Lelbach, B. A., Edwards, H. C., Hoemmen, M., Sunderland, D., and Trott, C. R. (2019). mdspan in c++: A case study in the integration of performance portable features into international language standards. In *International Workshop on Performance, Portability and Productivity in HPC*, pages 60–70. IEEE.

[Hussain et al. 2019] Hussain, S. M., Wahid, A., Shah, M. A., Akhunzada, A., Khan, F., Amin, N. u., Arshad, S., and Ali, I. (2019). Seven pillars to achieve energy efficiency in high-performance computing data centers. *Recent Trends and Advances in Wireless and IoT-enabled Networks*, pages 93–105.

[Inria, University of Lille 2024] Inria, University of Lille (2024). Power api. Copyright © 2024 Inria, University of Lille. Made with Material for MkDocs.

[Kiessling 2009] Kiessling, A. (2009). An introduction to parallel programming with openmp. In *The University of Edinburgh, A Pedagogical Seminar (accessed 24 September 2020), URL: https://www. roe. ac. uk/ifa/postgrad/pedagogy/2009_kiessling. pdf*, volume 76.

[Lattner and Adve 2004] Lattner, C. and Adve, V. (2004). Llvm: a compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pages 75–86.

[Machado et al. 2017] Machado, R. S., Almeida, R. B., Jardim, A. D., Pernas, A. M., Yamin, A. C., and Cavalheiro, G. G. H. (2017). Comparing performance of c compilers optimizations on different multicore architectures. In *2017 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, pages 25–30.

[Mishra et al. 2020] Mishra, A., Malik, A. M., and Chapman, B. (2020). Extending the llvm/clang framework for openmp metadirective support. In *2020 IEEE/ACM 6th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC) and Workshop on Hierarchical Parallelism for Exascale Computing (HiPar)*, pages 33–44.

[Tiwari et al. 2012] Tiwari, A., Laurenzano, M. A., Carrington, L., and Snavely, A. (2012). Modeling power and energy usage of hpc kernels. In *IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 990–998.

[Williams and Curtis 2008] Williams, J. and Curtis, L. (2008). Green: The new computing coat of arms? *IT Professional Magazine*, 10(1):12.