Volume 12, No.5, May 2025

Journal of Global Research in Mathematical Archives

ISSN 2320 – 5822

RESEARCH PAPER

Available online at http://www.jgrma.com

A COMPARATIVE STUDY OF GCC, CLANG, AND ICC: WHICH C COMPILER EXCELS?

Mohit Sahu[1], Aklesh Kumar[2]

[1,2] Department of Computer Science and Applications
Mandsaur University, Mandsaur, Madhya Pradesh, India
mohit.sahu@meu.edu.in[1], aklesh.kumar@meu.edu.in[2]

Abstract: Picking a C compiler matters in software programming since it determines how well the application performs, where it can run, and how easy it is to take care of. This article looks at GCC (GNU Compiler Collection), Clang, and ICC (Intel C Compiler), which are three common C compilers. Factors such as optimizing performance, handling errors, being portable, user friendliness, and sticking to C standards are used to compare the languages. It is clear from the results that GCC does best in offering broad support for different platforms, a steady performance and strong community backing. The main strength of Clang is how easy and modular its diagnostics are and ICC excels at optimizing code for Intel architectures. The purpose of this document is to guide developers in selecting the appropriate compiler for their project.

Keywords: C compilers, GCC, Clang, ICC, Performance optimization, Diagnostics, Portability, Standards compliance.

1 INTRODUCTION

The creation of operating systems, embedded systems and high-performance applications owes much to C programming language. Software engineers have to make sure they have the correct compiler for the job. The compiler links the code programmers write to code the machine can read and it is key to deciding how well the software will run [1][2]. Three popular C compilers that are widely used in open-source and commercial circumstances are GCC (GNU Compiler Collection), Clang and ICC (Intel C Compiler) [3][4]. Each of these compilers has its strengths and weaknesses, making their comparison essential when deciding on the best option based on project needs [5][6]. This study compares these three compilers based on the following five key criteria:

- Performance optimization
- Error handling
- Portability
- Ease of use
- Compatibility with C standards

The goal is to help developers, researchers, and organizations make an informed choice about which compiler best suits their project needs.

2 EVALUATION CRITERIA

To provide an unbiased and comprehensive analysis, it has selected five primary areas of evaluation:

- **Performance Optimization:** The ability of the compiler to optimize code for different architectures, enabling efficient execution.
- **Error Handling:** The clarity, usefulness, and diagnostic value of the compiler during the development process.
- **Portability:** The ease with which the compiler can run on different platforms and operating systems.
- **Ease of Use:** The interface, documentation, and support community surrounding the compiler.
- **Compatibility:** The degree to which the compiler complies with official C language standards and introduces new or experimental features.

Each of these factors is weighed based on their practical impact on C programming and the developer's workflow.

3 METHODOLOGY

This study employs qualitative comparative analysis of GCC, Clang, and ICC. The three compilers were tested on actual C programs and codebases from various domains, such as system programming [7], embedded systems, and performance-critical applications.

- **Performance Testing:** The compilers were tested using multiple benchmarks to measure execution time, optimization effectiveness, and resource usage. The performance of compilers was studied on Intel-based and ARM-based hardware.
- **Error Handling Test:** Topics were covered by testing with code that showed errors and situations that cause bugs [8]. It examined if the diagnostics offered actual advice to help developers.
- **Portability Test:** The portability test involved developing and executing the same code on different kinds of operating systems (Linux, macOS, Windows) and hardware platforms.
- **Ease of Use:** Using the software eased was evaluated through its documentation, how it fits with development programs and how quick it is for someone new to learn.
- **Compatibility:** Specialists examined the adherence to C language standards in the compilers and checked if they include things such as C11 and C18, along with experimental extensions.

4. COMPARATIVE ANALYSIS

4.1 GCC (GNU Compiler Collection)

Developers often prefer GCC, mostly for open-source projects and when programming using various platforms. People say that Unity stands out with its user-friendly approach and broad support on different platforms.

- **Performance Optimization:** GCC includes features such as link-time optimization (LTO) and intraprocedural optimization, leading to better optimization across platforms [9].
- **Error Handling:** There are detailed and insightful error messages from GCC. Even if these messages can be wordy, the clear information helps developers solve errors more quickly [10].
- **Portability:** GCC works on Linux, macOS, Windows, and embedded systems. It represents one of the most versatile compilers available.
- **Ease of Use:** The GCC community is large and the documentation is detailed which allows anyone wanting to learn to use it. On the other hand, command-line interface may appear difficult for beginners.
- **Compatibility:** GCC includes major C standards and continuously includes up-to-date features. Years of sticking to the C standard mean that it is a safe choice for most projects.

4.2 Clang (LLVM-based Compiler)

Many developers have started using Clang because it is modular and aims to find and report programming errors [11][12].

- **Performance Optimization:** The optimization in Clang is impressive and it is usually as effective as GCC. Dart is also distinguished by its fast compilation while still maintaining good optimization [13].
- **Error Handling:** Debugging is easy in Clang because its error messages are simple and clear, especially suited for new developers [14].
- **Portability:** Clang is similar to GCC in being highly portable and operating on macOS, Linux and Windows. The use of it feels natural in IDEs like Xcode and Visual Studio Code.
- **Ease of Use:** MacOS developers usually prefer Clang due to how simple it is and how well it works with other development software.
- **Compatibility:** Clang uses standards and is built to easily accept and use modern and experimental additions to the C language.

4.3 ICC (Intel C Compiler)

ICC by Intel is a compiler that works best on its own processors and sees regular use in high-performance computing environments.

- **Performance Optimization:** ICC does better than GCC and Clang where Intel-specific tricks like vectorization and parallel processing are used. It is most suited for tasks in science and engineering.
- **Error Handling:** ICC gives useful error messages, but Clang's are more understandable. These documents are helpful, but they miss instructions for action.
- **Portability:** ICC cannot be used on all computers because it is designed mainly for Intel computers. It does not work for computers using non-Intel processors.
- **Ease of Use:** ICC can be easily used with Intel's popular development tools, though having a lot of knowledge about Intel makes it harder for other developers to use.
- **Compatibility:** ICC can use C standards, though it may not be compatible with brand-new or experimental language elements.

5. RESULTS AND DISCUSSION

- **GCC:** It gives the most reliable answer for most of the tasks done in C programming [15]. Because it works on so many platforms and is flexible, Rust is great for many types of development.
- **Clang:** Ideal for programmers concerned about having quick compilation results and comprehensive error messages [16]. It is especially good for building software in contemporary environments, and this includes macOS.
- **ICC:** Works extremely well on Intel processors, however, it is not very portable or compatible with everything [17]. Intel recommends it when raw performance in important applications is needed.

6. CONCLUSION

This research indicates that GCC excels in versatility because it maintains good performance, portability, and compatibility without making the tool difficult to use. Those creating apps for Intel equipment consider ICC a need because of its outstanding optimization for their processors. If diagnostics, fast compilation, and a current toolchain are top priorities, Clang is an excellent program for you. Which compiler to pick is influenced by the requirements of the project and the target platform.

REFERENCES

[1] K. Daungcharone, P. Panjaburee, and K. Thongkoo, "Using Digital Game as Compiler to Motivate C Programming Language Learning in Higher Education," in *2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, IEEE, 2017, pp. 533–538.

[2] P. Grigorenko, A. Saabas, and E. Tyugu, "COCOVILA - Compiler-compiler for visual languages," in *Electronic Notes in Theoretical Computer Science*, 2005. doi: 10.1016/j.entcs.2005.05.009.

[3] S. R. Thota, S. Arora, and S. Gupta, "Al-Driven Automated Software Documentation Generation for Enhanced Development Productivity," in *2024 International Conference on Data Science and Network Security (ICDSNS)*, IEEE, Jul. 2024, pp. 1–7. doi: 10.1109/ICDSNS62112.2024.10691221.

[4] N. I. V'yukova, V. A. Galatenko, and S. V. Samborskii, "Dynamic Program Analysis Tools in GCC and CLANG Compilers," *Program. Comput. Softw.*, vol. 46, no. 4, pp. 281–296, Jul. 2020, doi: 10.1134/S0361768820010089.

[5] Z. Gong *et al.*, "An empirical study of the effect of source-level loop transformations on compiler stability," *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, 2018, doi: 10.1145/3276496.

[6] V. S. Thokala, "A Comparative Study of Data Integrity and Redundancy in Distributed Databases for Web Applications," *Int. J. Res. Anal. Rev.*, vol. 8, no. 4, pp. 383–389, 2021.

[7] E. Damasceno, F. Queiroz, L. Siqueira, T. Rodrigues, and M. Amaris, "Comparative Analysis of Compiler Efficiency: Energy Consumption Metrics in High-Performance Computing Domains," in *Anais do XXV Simpósio em Sistemas Computacionais de Alto Desempenho (SSCAD 2024)*, Sociedade Brasileira de Computação, Oct. 2024, pp. 252–263. doi: 10.5753/sscad.2024.244771.

[8] J. G. Feng, Y. P. He, and Q. M. Tao, "Evaluation of Compilers' Capability of Automatic Vectorization Based on Source Code Analysis," *Sci. Program.*, vol. 2021, pp. 1–15, Nov. 2021, doi: 10.1155/2021/3264624.

[9] A. A. Moreira, G. Ottoni, and F. M. Quintão Pereira, "VESPA: Static profiling for binary optimization," *Proc. ACM Program. Lang.*, 2021, doi: 10.1145/3485521.

[10] H. S. Chandu, "A Review on CNC Machine Tool Materials and Their Impact on Machining Performance," *Int. J. Curr. Eng. Technol.*, vol. 14, no. 5, pp. 313–319, 2024.

[11] A. Litteken, Y.-C. Fan, D. Singh, M. Martonosi, and F. T. Chong, "An Updated LLVM-Based Quantum Research Compiler With Further OpenQASM Support," *Quantum Sci. Technol.*, vol. 5, no. 3, Jul. 2020, doi: 10.1088/2058-9565/ab8c2c.

[12] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A LLVM-based Python JIT Compiler," in *Proceedings of LLVM-HPC 2015: 2nd Workshop on the LLVM Compiler Infrastructure in HPC - Held in conjunction with SC 2015: The International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015. doi: 10.1145/2833157.2833162.

[13] B. C. Lopes and A. Rafael, *Getting Started with LLVM Core Libraries*. 2014.

[14] T. Sadasue and T. Isshiki, "LLVM-C2RTL: C/C++ Based System Level RTL Design Framework Using LLVM Compiler Infrastructure," *IPSJ Trans. Syst. LSI Des. Methodol.*, 2023, doi: 10.2197/ipsjtsldm.16.12.

[15] J. Kvalsvik, "Modified Condition/Decision Coverage in the GNU Compiler Collection," 2025.

[16] T. Vargas and W. Jose, "Enhancing the Clang Compiler with OpenMP Parallelism : A Novel Approach Using ClangIR," Universitat Politècnica de Catalunya (UPC), 2024.

[17] Intel Corporation, "Intel C++ Compiler 17.0 Developer Guide and Reference," 2016.

