УДК 004.432

АНАЛІЗ ШВИДКОДІЇ КОМПІЛЯТОРА МОВИ ПРОГРАМУВАННЯ GO

Орлов О.С. [1]; Петрова Р.В.[2], к.т.н., доц.
[1,2]Харківський національний університет радіоелектроніки
[1,2]Україна, Харків
[1]oleksii.orlov@nure.ua; [2]roksana.petrova@nure.ua

Анотація. *Робота присвячена аналізу швидкодії компілятора мови Go (gc) з метою підтвердження його практичних переваг у швидкості збірки. Дослідження поєднує теоретичний аналіз архітектури компілятора та практичне порівняння його продуктивності з компіляторами мов програмування C++ (clang++) та Rust (rustc).*
Ключові слова: *Go; компілятор; компонувальник; CI/CD; аналіз продуктивності.*

Вступ. У сучасному ландшафті розробки програмного забезпечення час компіляції залишається критичним фактором, що безпосередньо впливає на продуктивність розробників та ефективність процесів безперервної інтеграції та доставки (CI/CD). Тривалі цикли збірки, характерні для таких мов, як C++ або Java, стали однією з основних мотивацій для створення мови Go. Розробники Go ставили за мету не лише простоту синтаксису та ефективне виконання, але й надзвичайно швидку компіляцію. Актуальність даного дослідження полягає у необхідності верифікації та кількісного оцінювання успішності досягнення цієї мети в умовах сучасних програмних систем.

Мета дослідження. Метою даної роботи є проведення теоретичного та експериментального аналізу швидкодії стандартного компілятора мови Go (gc), виявлення ключових архітектурних рішень, що впливають на його продуктивність, та порівняння його показників із поширеними компіляторами інших системних мов програмування.

Основна частина. Мова Go вимагає явного декларування всіх імпортованих пакетів на початку кожного файлу. Компілятор не шукає файли у системних шляхах неявним чином. Це, у поєднанні з відсутністю заголовкових файлів (таких, як у C/C++), радикально спрощує та прискорює аналіз графа залежностей.

Одиницею компіляції є пакет (package). Компілятор Go компілює кожен пакет окремо і паралельно (якщо дозволяють залежності). Інформація про експортовані типи та функції зберігається у стислому бінарному форматі в самому об'єктному файлі (.a), що усуває потребу в окремих метаданих або заголовках.

Хоча gc вже не є суто однопрохідним, як на початку своєї історії, він зберіг ідею мінімізації надлишкової роботи. Він використовує специфічне проміжне подання (IR) і уникає багатьох складних, часозатратних оптимізацій, характерних для LLVM (на якому базуються clang та rustc), зосереджуючись на «достатньо хорошій» оптимізації за мінімальний час.

Go використовує власний, написаний на цій мові, компонувальник, який спеціально оптимізований для створення статично пов'язаних виконуваних файлів. Він працює значно швидше за традиційні компонувальники, такі як ld (GNU) або lld (LLVM), оскільки йому не потрібно розв'язувати складні задачі динамічного зв'язування.

На другому, експериментальному, етапі було проведено порівняльне тестування. Було прийняте рішення обрати три проекти з відкритим вихідним кодом: малий (утиліта командного рядка, близько 5 тис. рядків коду), середній (веб-сервер, близько 20 тис. рядків) та великий (open-source проєкт, близько 100 тис. рядків). Для кожного типу були знайдені конкретні проекти, написані на мовах Go[1;2;3], Rust[4;5;6] та C++[7;8;9].

Було проведено виміри двох показників:

- час «холодної» збірки (Cold Build): час повної компіляції проєкту з чистого стану (без кешу);
- час «гарячої» збірки (Hot Build): час інкрементальної перекомпіляції після внесення незначної зміни в один із файлів.

Вимірювання проводилися на стандартизованій машині (ОС Linux, 4-ядерний процесор, 16 ГБ оперативної пам'яті, SSD підключений до інтерфейсу SATA3 6GB/s) з використанням утиліти hyperfine для статистично значущих вимірів. Дані вимірювань наведені у табл. 1 та табл. 2.

Таблиця 1 — Порівняння середнього часу повної компіляції проектів

	Малий проект	**Середній проект**	**Великий проект**
Rust	40.9 с.	80.8 с.	117.9 с.
C++	94.2 с.	240 с.	401.5 с.
Go	11.3 с.	20.3 с.	41.7 с.

Таблиця 2 — Порівняння середнього часу інкрементальної компіляції проектів

	Малий проект	**Середній проект**	**Великий проект**
Rust	1.2 с.	5.3 с.	10 с.
C++	5.7 с.	12.6 с.	20.4 с.
Go	450 мс.	3.1 с.	7 с.

Результати експерименту показали, що компілятор Go демонструє значно кращі показники в обох сценаріях. Для «холодної» збірки середнього проєкту gc виявився в середньому у 4 рази швидшим за rustc (у режимі dev) та у 12 разів швидшим за clang++. В інкрементальних збірках перевага зберігалася, хоча й скорочувалася, особливо у порівнянні з rustc, який має досить ефективний механізм кешування. Перевага Go була особливо помітною на етапі компонування.

Висновки. Проведений аналіз підтверджує, що компілятор мови Go є найшвидшим у своїй галузі. Теоретичне дослідження виявило, що його швидкодія є результатом свідомих архітектурних рішень: простої моделі залежностей, ефективної паралельної компіляції пакетів та високо-оптимізованого власного компонувальника.

Експериментально доведено, що gc значно (в рази) випереджає компілятори C++ (Clang) та Rust (rustc) за часом повної збірки проєктів різного масштабу. Ця характеристика робить Go дуже ефективною мовою для середовищ, що вимагають швидкої ітерації, зокрема для розробки мікросервісів та впровадження CI/CD практик. Також, швидкість та легкість процесу компіляції впливає на кількість використаної електроенергії, що робить компілятор Go дуже ефективним з точки зору виділення тепла та вуглекислого газу під час обробки процесором команд компілятора.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Github.(2025). Малий проект мовою програмування Go. https://github.com/charmbracelet/gum
[2] Github. (2025). Середній проект мовою програмування Go. https://github.com/go-chi/chi
[3] Github. (2025). Великий проект мовою програмування Go. https://github.com/gohugoio/hugo
[4] Github. (2025). Малий проект мовою програмування Rust. https://github.com/sharkdp/hyperfine

[5] Github. (2025). Середній проект мовою програмування Rust. https://github.com/tokio-rs/axum
[6] Github. (2025). Великий проект мовою програмування Rust. https://github.com/tursodatabase/turso
[7] Github. (2024). Малий проект мовою програмування C++. https://github.com/LesterEvSe/AteNika
[8] Github. (2023). Середній проект мовою програмування C++. https://github.com/sprinfall/webcc
[9] Github. (2025). Великий проект мовою програмування C++. https://github.com/OpenTTD/OpenTTD

Orlov O.S., Petrova R.V.

Analysis of Go compiler performance

Abstract. *This work is dedicated to analysis of the performance of the Go compiler (gc) in order to confirm its practical advantages in terms of compilation speed. The study combines a theoretical analysis of the compiler architecture with a practical comparison of its performance with the C++ (clang++) and Rust (rustc) programming language compilers.*

Keywords: *Go; compiler; linker; CI/CD; performance analysis.*