Шуржунов Н.В., Алещенко А.С.

Оценка удовлетворительности быстродействия Kotlin/Native в сравнении с альтернативными языками в различных типах задач

Московский авиационный институт (национальный исследовательский университет) (Россия, Москва)

doi: 10.18411/trnio-05-2022-21

Аннотация

В статье анализируется быстродействие Kotlin/Native при выполнении типичных промышленных задач, а также степень готовности языка к подобного рода задачам. Для понимания относительного быстродействия скорость работы Kotlin/Native сравнивается со скоростью работы C++, Rust, а также Kotlin/JVM.

Ключевые слова: Kotlin, Kotlin/Native, быстродействие, низкоуровневые языки программирования, системные языки программирования, LLVM.

Abstract

The article analyzes Kotlin/Native performance in real-life commercial tasks, as well as answers the question if it is ready to be used in such way. In order to benchmark relative performance, Kotlin/Native execution speed compared with such of C++, Rust, and Kotlin/JVM.

Keywords: Kotlin, Kotlin/Native, performance, low-level programming languages, system programming languages, LLVM.

Различные языки программирования имеют свои преимущества и ниши применения. Некоторые, как в случае с Kotlin, расширяют нишу применения, добавляя новые функции языка. В таком случае наступает момент, когда необходимо понять, для каких задач подходят добавленные функции.

Язык Kotlin, кроме удобства разработки под JVM с двухсторонней совместимостью с Java, имеет возможность мультиплатформенной разработки. В этом случае программа разделяется на платформонезависимую логику и реализацию зависимого от платформы функционала.

В случае, если целевой платформой является конкретная платформа, к примеру десктопная ОС (Windows, Mac OS, Linux), или же мобильная ОС (iOS, Android) Kotlin собирается в полноценное нативное приложение [1]. Для сборки используется LLVM-бэкенд, для получения LLVM IR применяется Konan – собственный LLVM-фронтенд Kotlin. Также, Kotlin/Native поддерживает функциональную совместимость с языком C, что позволяет применять в разработке библиотеки с C-ABI. При этом в языке имеются конструкции, позволяющие увеличить в сравнении с C безопасность при работе с памятью, что положительно сказывается на безопасности конечного ПО [2].

Подобное решение выглядит удовлетворяющим требованиям мультиплатформенной разработки, однако нераскрытым остаётся вопрос актуальности Kotlin/Native при разработке ПО с требованиями к скорости работы [3]. Иными словами, вопрос способности Kotlin/Native работать столь же быстро, как низкоуровневые, системные языки, такие, как C, C++ [4], Rust [5].

Для сравнения скорости работы кроме Kotlin/Native были взяты ещё три языка:

- Kotlin/JVM (Основная реализация Kotlin, работающая на JVM)
- C++
- Rust

Языки C++ и Rust являются системными и низкоуровневыми, в данном случае представляя из себя эталон скорости работы. Kotlin/JVM используется для того, чтобы сравнить степень отклонения скорости работы Kotlin/Native от стандартной реализации под JVM [4].

Важной частью постановки задачи проверки скорости языков было условия приближённости тестовых задач к реальной разработке, а не синтетическим тестам. Также, код задач не был дополнительно оптимизирован, реализация была максимально приближена к обычной промышленной разработке.

В итоге были выбраны следующие три задачи:

– Расчёт 1 000 000 факториалов
– Поиск в файле строчек, в которые входит заданное слово
– Параллельный расчёт SHA-256 сум для строк на 4 ядрах, по 1 000 000 на каждое ядро

Расчёты произведены на нескольких разных компьютерах, после чего результаты усреднены для получения нейтральной картины.

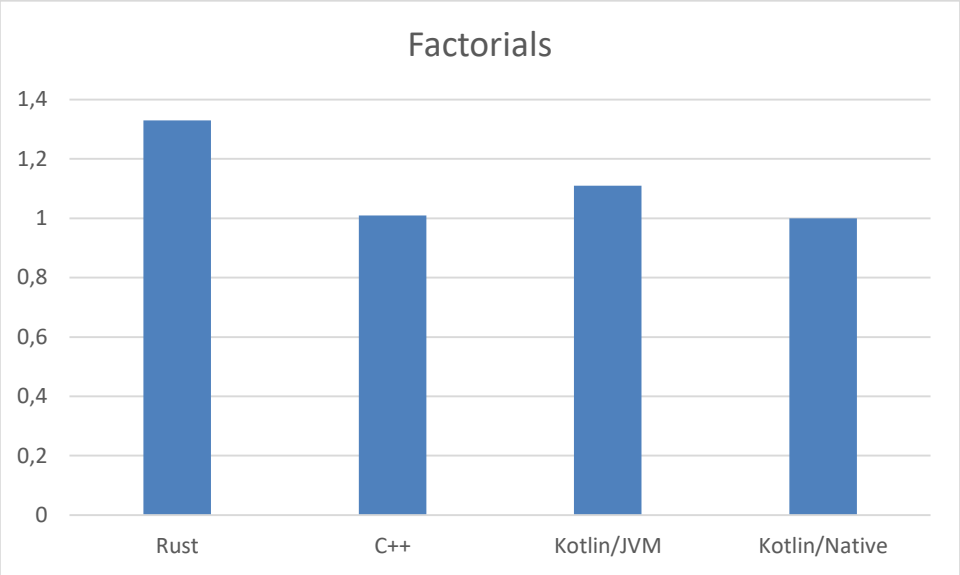

Рисунок 1 Сравнение скорости вычисления при расчёте факториалов.

При расчёте факториалов (Рис. 1), задаче, не затрагивающей I/O и параллельные вычисления, Kotlin/Native показал себя по скорости на равных с реализацией на C++.

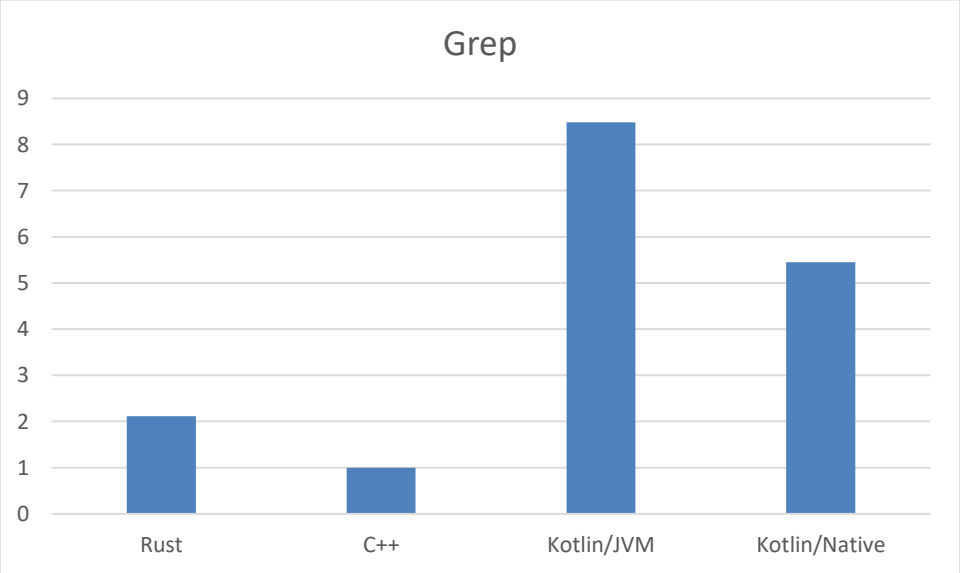

Рисунок 2 Сравнение скорости вычисления при поиске в файле строк, содержащих заданное слово.

В случае с задачей по поиску строк (Рис. 2), содержащих определённое слово, Kotlin/Native продемонстрировал скорость меньшую, чем системные языки, но большую, чем

JVM реализация. При этом важно отметить, что в отличие от C++, в Kotlin по умолчанию используются не ASCII строки.

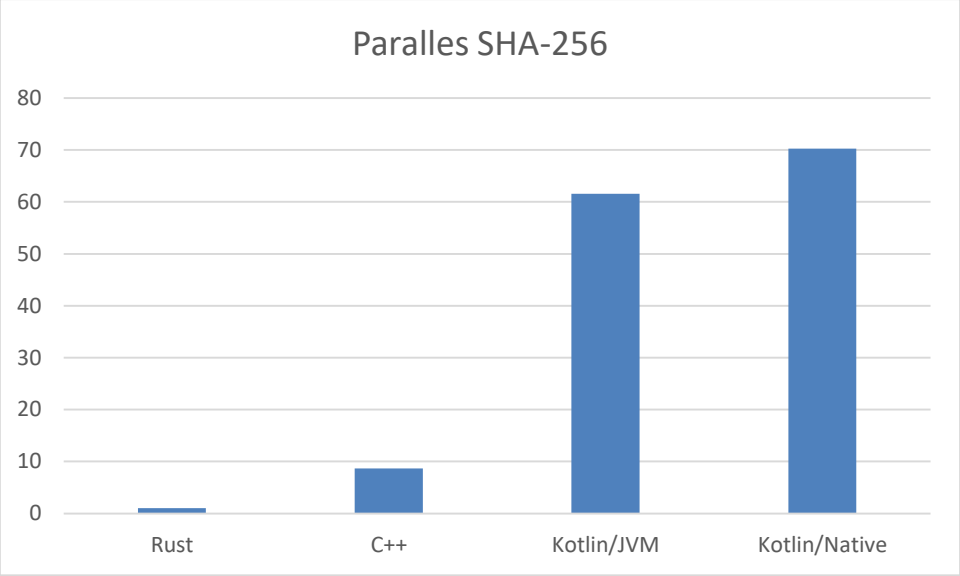

Рисунок 3 Сравнение скорости вычисления при параллельном вычислении SHA-256.

В параллельных вычислениях SHA-256 (Рис. 3) Kotlin/Native показал себя медленнее всего. Причиной тому может быть или скорость работы воркеров, при помощи которых производятся параллельные вычисления, или скорость работы при использовании сторонних библиотек для C при помощи функциональной совместимости.

Подводя итог, можно сказать, что сам по себе Kotlin/Native по скорости может быть сравним с низкоуровневыми языками, хотя в задачах, где активно применяется взаимодействие с системой, порой может сказываться отсутствие оптимизацией в связи с тем, что Kotlin/Native всё ещё находится на стадии альфа-версии.

Однако, с учётом возможностей мультиплатформенной разработки, применение возможно уже сейчас, если учитывать тип задач или же надеяться на будущие улучшения. Для обычных задач, где по скорости он сравним с Kotlin/JVM или даже быстрее, Kotlin/Native может стать хорошим решением, особенно там, где нет возможности применять JVM, например, на iOS.

Тем не менее, для задач, где скорость вычислений всё же важнее удобства разработки, всё же рекомендуется применять низкоуровневые языки.

***

1. Kotlin/Native for Native. Kotlin. Available online: [Электронный ресурс] https://kotlinlang.org/docs/reference/native-overview.html (дата обращения: 14.04.2022).
2. Gilad Bracha: Keynote address: towards secure systems programming languages. 2004, DOI: 10.1145/967900.967902
3. Alexander Nozik: Kotlin language for science and Kmath library. 2019, DOI: 10.1063/1.5130103
4. Luca Gherardi, Davide Brugali, Daniele Comotti: A Java vs. C++ Performance Evaluation: A 3D Modeling Benchmark. 2012, DOI: 10.1007/978-3-642-34327-8_17
5. Manuel Costanzo, Enzo Rucci, Marcelo Naiouf, Armando De Giusti: Performance vs Programming Effort between Rust and C on Multicore Architectures: Case Study in N-Body. 2021, [Электронный ресурс] https://www.researchgate.net/publication/353479028_Performance_vs_Programming_Effort_between_Rust_and_C_on_Multicore_Architectures_Case_Study_in_N-Body (дата обращения: 14.04.2022)