JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 30 (2024) 21–25
Received: 13 September 2023
Accepted: 18 October 2023

C++ and Kotlin performance on Android – a comparative analysis

Analiza porównawcza wydajności języków C++ i Kotlin na platformie Android

Grzegorz Zaręba*, Maciej Zarębski, Jakub Smołka

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
The article discusses the comparison of C++ and Kotlin programming languages in a mobile environment. The authors performed a series of tests based on five selected algorithms: n-bodies, the n^{th} term of the Fibonacci sequence, reading and writing a file, and bubble sort for both small and large sets of values. The tests were carried out in a way that allowed to determine the performance of the Kotlin language both when it uses the Just-in-Time compilation mechanism and when it is not used. The research was carried out both on a physical mobile device and emulators. Although the C++ language outclassed its rival in most of the tests performed, Kotlin showed more than three times faster performance when bubble sorting on a small (20,000 values) array.

Keywords: programming language comparison; C++; Kotlin; Just-In-Time Compilation

Streszczenie
Artykuł porównuje języki programowania C++ i Kotlin w środowisku mobilnym. Autorzy wykonali serie testów w oparciu o pięć wybranych algorytmów: n-ciał, n-ty wyraz ciągu Fibonacciego, odczyt i zapis do pliku oraz sortowanie bąbelkowe dla małych oraz dużych zbiorów. Testy wykonano w sposób pozwalający określić wydajność języka Kotlin zarówno kiedy wykorzystuje on mechanizm kompilacji Just-in-Time, jak również gdy nie jest on używany. Badania przeprowadzono zarówno na fizycznym urządzeniu mobilnym, jak również emulatorach. Jakkolwiek język C++ zdeklasował rywala w większości wykonanych testów, Kotlin wykazał się ponad trzykrotnie większą szybkością działania przy sortowaniu bąbelkowym na małej (20 tysięcy wyrazów) tablicy.

Słowa kluczowe: porównanie języków programowania; Kotlin; C++; Kompilacja Just-In-Time

*Corresponding author
Email address: **grzegorz.zareba@pollub.edu.pl** (G. Zaręba)

1. Wstęp

Telefony komórkowe dwukrotnie już zrewolucjonizowały życie codzienne nieomal każdego człowieka – po raz pierwszy poprzez umożliwienie mu dzwonienia z dowolnego miejsca objętego zasięgiem sieci GSM, następnie zaś wraz z wejściem na rynek smartfonów oddając mu do ręki de facto uniwersalny i mobilny komputer z dostępem do Internetu. Największą popularnością obecnie cieszą się urządzenia z systemem Android do których należy niemalże 35% rynku [1]. Nie są one jednak idealne, ze względu na oparcie systemu o wirtualną maszynę Java, później zaś dołączenie doń języka Kotlin. Języki wspomniane nie cieszą popularnością w miejscach, w których priorytetyzowana jest szybkość wykonania operacji – w wypadku urządzeń mobilnych mogą to być operacje niskopoziomowe tudzież skomplikowane algorytmy, chociażby związane z kryptografią. Z myślą o takich zastosowaniach, Google umożliwiło kompilowanie dla urządzeń Android kodu języka C oraz C++ poprzez narzędzie zwane NDK (Native Development Kit).

Praca niniejsza jest próbą porównania wydajności tych dwóch języków o teoretycznie zgoła odmiennym charakterze. Miarą testów przeprowadzanych naprzemiennie jest czas realizacji zadanego algorytmu. Pytaniem na które chcą odpowiedzieć autorzy, jest zasadność przechodzenia przez skomplikowany proces wytwarzania aplikacji dla systemu Android (lub ich części) w C++, mając jako alternatywę stworzony przez JetBrains język Kotlin.

2. Cel i zakres badań

Celem badań jest porównanie wydajności kodu napisanego w językach C++ oraz Kotlin, uruchamianego w ramach jednolitej aplikacji działającej na maszynie wirtualnej Javy, w środowisku systemu Android. Porównanie opiera się na wynikach testów wydajnościowych, polegających na pomiarze czasu wykonania poszczególnych algorytmów, zaimplementowanych przez autorów w obu badanych językach.

Zakres badań obejmuje:

- zapoznanie się z istniejącą literaturą naukową,
- zaimplementowanie aplikacji testowej oraz algorytmów,
- wykonanie pomiarów wydajności zaimplementowanych algorytmów,
- analizę wyników.

3. Przegląd literatury

Jedną z prac traktujących o wydajności aplikacji mobilnych napisanych w różnych językach programowania jest [2]. Autorzy zaimplementowali prostą aplikację w środowisku systemu Android 2.2 w dwóch technolo-

giach. Jedną z nich był język Java, a drugą - stos technologiczny HTML + CSS + JavaScript w połączeniu ze szkieletem programistycznym PhoneGap. Jedną z istotnych obserwacji, poczynionych przez autorów omawianej pracy, jest fakt, że kod źródłowy niekompilujący się bezpośrednio do kodu bajtowego maszyny wirtualnej Java, musi zostać najpierw przetworzony w taki sposób, aby mógł być przez tę maszynę zrozumiany i wykonany. Z tej przyczyny wydajność aplikacji wykonanej w oparciu o szkielet PhoneGap była zauważalnie gorsza. Jako iż Kotlin jest kompilowany bezpośrednio do kodu bajtowego Javy, a C++ - nie, można domniemywać, że aspekt ten będzie miał również znaczenie przy porównaniu tychże języków.

Kolejnym artykułem który stanowił istotne dla autorów źródło wiedzy jest [3]. Użyte i opisane w nim algorytmy (jak chociażby N-bodies) znalazły zastosowanie w niniejszej pracy, przez wzgląd na łatwość implementacji i miarodajność wyniku, będącego de facto sumą operacji matematycznych, niewymagających zastosowania (jak choćby przy odczycie/zapisie do pliku) użycia specjalizwoanych bibliotek. Sam artykuł porównuje wydajność, zużycie zasobów oraz, co interesujące, zużycie prądu podczas realizacji algorytmu w kilkudziesięciu językach programowania. Wykonane badania ustanowiły sprawność czasową C++ i Javy względem języka C na odpowiednio: 1:1,56 oraz 1:1,89. Nie jest jednakoż jasnym jak uplasowałby się Kotlin, zwłaszcza uruchomiony w środowisku mobilnym, z myślą o którym został stworzony.

W doborze narzędzi istotna okazała się praca [4], pokazująca różnicę pomiędzy zastosowaniem NDK oraz bezpośredniej cross-kompilacji na ARM. Aby sprawdzić hipotezę, badacze przeprowadzili testy oparte o sześć algorytmów, między innymi Quicksort, Szybką transformację Fouriera czy Algorytm Djikstry. Artykuł wykazał wyższość rozwiązania NDK, wykazującego się zdecydowanie krótszymi czasami wykonania – w zależności od algorytmu od 27,6% aż do 114,4% szybciej od rozwiązania bezpośredniego.

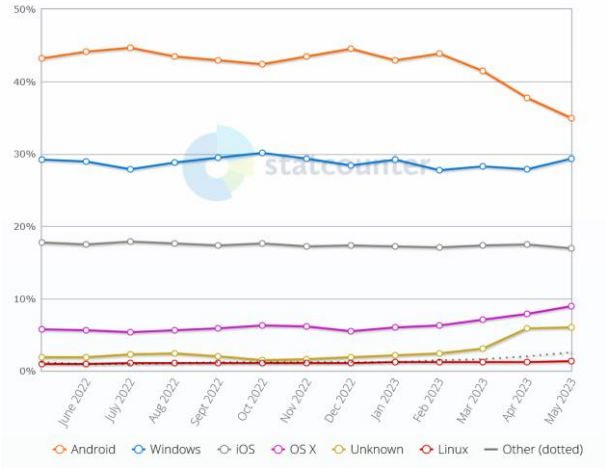
Kolejną godną uwagi publikacją, omawiającą problematykę wykorzystywania kodu C++ w aplikacjach dla systemu Android, jest prezentacja z wykładu [5]. Autor omawia w niej zalety oraz wady takiego podejścia, a także opisuje potencjalne problemy, na które natknąć może się programista w trakcie procesu budowania i uruchamiania swojej aplikacji wykorzystującej C++.

4. Omówienie technologii

Jako, iż tekst dotyczy wydajności kodu na platformie Android, większość zagadnień technologicznych, z którymi autorzy się zapoznali, jest związana z tym systemem.

4.1. System operacyjny Android

Android to otwartoźródłowy system operacyjny, oparty o jądro Linux, i rozwijany przez firmę Google. System jest wykorzystywany głównie na urządzeniach opartych o procesory ARM, takie jak smartfony, smartwatche, telewizory, czy komputery pokładowe w samochodach. Android to system operacyjny cieszący się dużą popularnością (Rysunek 1) - w poczet jego zalet zaliczyć można otwartoźródłowość, szeroką dostępność darmowych narzędzi wspomagających tworzenie oprogramowania, oraz możliwość wykorzystania wielu różnych języków, takich jak C++, Java, Kotlin, Dart, czy C# przy tworzeniu aplikacji. Testy opisane w niniejszej publikacji były przeprowadzane w środowisku Android 13, czyli najnowszej stabilnej wersji, dostępnej w momencie przygotowywania niniejszego artykułu.



Rysunek 1: Globalny udział rynkowy poszczególnych systemów operacyjnych w okresie kwiecień 2022 - kwiecień 2023 [1].

4.2. Język programowania C++

C++ to wieloparadygmatowy język programowania, bazujący na popularnym języku C. Jest on często stosowany tam, gdzie kluczowym jest osiągnięcie wysokiej wydajności kodu, na przykład przy obliczeniach związanych z grafiką trójwymiarową, bądź też w sytuacjach, w których programista potrzebuje zachować wysoki stopień kontroli nad sprzętem - przykładowo przy tworzeniu systemów operacyjnych [6]. C++ nie jest domyślnym językiem, zalecanym przez twórców systemu Android do tworzenia aplikacji na tej platformie. Mimo to, możliwe jest wykorzystanie kodu napisanego w tym języku poprzez zastosowanie narzędzi Android NDK, a w szczególności Java Native Interface - biblioteki umożliwiającej wywoływanie kodu C/C++ z poziomu kodu języka Java, bądź języków pochodzących od Javy, takich jak np. Kotlin [4].

4.3. Język programowania Kotlin

Kotlin to statycznie typowany język programowania działający na maszynie wirtualnej Javy, który jest głównie rozwijany przez JetBrains na potrzeby urządzeń z systemem Android, został on zresztą ogłoszony oficjalnym językiem programowania dla tychże na konferencji Google I/O 2019 [7]. Jedną z cech tego opartego o wirtualną maszynę Javy rozwiązania jest wykorzystanie kompilacji Just-in-time. Metoda ta kompiluje najczęściej wykonywane partie tworzonego kodu do postaci języka maszynowego. Skutkuje to skróceniem czasu wykonania. W języku wprowadzono ponadto elementy

nullpointer-safety, oraz uproszczono semantykę. Z perspektywy biznesowej największą zaletą języka jest jego interoperacyjność z zastępowaną przezeń Javą.

5. Metody badawcze

Aby porównać wydajność języków Kotlin i C++ na platformie Android, autorzy przygotowali aplikację testową, mającą służyć za środowisko do uruchamiania poszczególnych testów, rejestrowania czasów wykonania algorytmów testowych, oraz zapisywania zgromadzonych wyników. Szkielet aplikacji, logika odpowiadająca za obsługę interfejsu użytkownika, oraz za zbieranie i zapisywanie wyników zostały zaimplementowane w języku Kotlin. Testy przeprowadzono w środowisku systemu Android 13 na dwóch urządzeniach: maszynie wirtualnej dostarczonej przez środowisko programistyczne Android Studio, oraz fizycznym urządzeniu - smartfonie Samsung Galaxy A51.

Pomiary czasu wykonania testów zostały dokonane z użyciem funkcji „measureTimeMillis", pochodzącej ze standardowej biblioteki języka Kotlin. Funkcja ta przyjmuje jako parametr wskaźnik do funkcji, której czas wykonania ma zostać zmierzony. Uzyskany wynik zapisywany jest do zmiennej, którą następnie można na przykład wyświetlić bądź zapisać do pliku. Jest to prosty i skuteczny, a przy tym dostatecznie precyzyjny sposób na wykonanie pomiarów.

5.1. Zastosowane algorytmy

Dla każdego języka zaimplementowane zostało sześć testów, opartych na pięciu algorytmach:

- Sortowanie bąbelkowe
- Zapis pliku do pamięci telefonu
- Odczyt pliku z pamięci telefonu
- Symulacja n-ciał
- Obliczanie n-tego wyrazu ciągu Fibonacciego

Test oparty o sortowanie bąbelkowe został przeprowadzony w wersji ze zbiorem dwudziestu tysięcy elementów, oraz stu tysięcy elementów. Dla każdego z testów przygotowane zostały zestawy danych wejściowych, niezmienne pomiędzy kolejnymi uruchomieniami, tak aby zagwarantować powtarzalność testów.

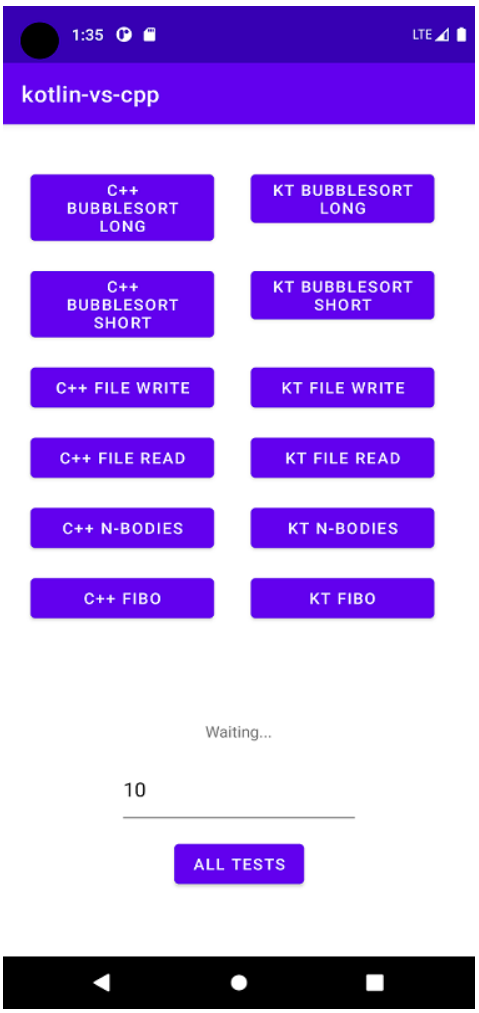
Test sortowania bąbelkowego polega na posortowaniu określonego zbioru danych. Sortowanie jest malejące, zaś przekazywany zbiór danych jest zawsze posortowany rosnąco. Sprawia to, że złożoność obliczeniowa algorytmu jest niezmienna dla każdego uruchomienia, co gwarantuje rzetelność wyników.

Test zapisu pliku polega na utworzeniu w określonej ścieżce pamięci telefonu pliku tekstowego, oraz zapisaniu do niego zadanego ciągu dwudziestu milionów znaków. Jeśli w momencie uruchomienia testu plik już istnieje, jest on usuwany przed rozpoczęciem pomiaru.

W ramach testu odczytu pliku, program ma za zadanie otworzyć uprzednio stworzony plik tekstowy i załadować jego zawartość, czyli ciąg dwudziestu milionów znaków, do zmiennej. Podobnie jak przy teście odczytu pliku, jeżeli w momencie uruchomienia testu plik nie istnieje, to jest on tworzony przed rozpoczęciem pomiaru.

Kolejny test, czyli symulacja n-ciał, to algorytm oparty na zasadach fizyki newtonowskiej, obliczający wielkości fizyczne opisujące ciała o zadanych parametrach, oraz ich zmiany wraz z upływem czasu. Algorytm zaimplementowany w ramach opisywanego testu symuluje pięć ciał niebieskich (Słońce, Neptun, Uran, Saturn, Jowisz) na przestrzeni dziesięciu tysięcy sekund, z krokiem czasowym wynoszącym jedną setną sekundy [3]. W ramach symulacji, przy każdym kroku czasowym obliczane jest położenie i prędkość każdego z ciał.

Ostatnim testem jest algorytm, obliczający w sposób iteracyjny wartość zadanego wyrazu ciągu Fibonacciego. Na potrzeby testu autorzy zdecydowali się każdorazowo znajdować wartość stumilionowego wyrazu ciągu.



Rysunek 2: Aplikacja testowa.

5.2. Aplikacja testowa

Interfejs graficzny aplikacji testowej przedstawiony został na rysunku 2. Aplikacja posiada jedną aktywność, z poziomu której można wywołać dowolny test, poprzez naciśnięcie odpowiednio opisanego przycisku. Ponadto możliwe jest wywołanie wszystkich testów za pomocą przycisku opisanego jako "All tests". Przycisk ten wywołuje wykonanie w losowej kolejności wszystkich zdefiniowanych w aplikacji testów określoną ilość razy - liczbę powtórzeń można podać w polu do wprowadzania liczb, znajdującym się powyżej.

5.3. Scenariusze testowe

W trakcie prowadzenia badań autorzy zauważyli, że mechanizm kompilacji just-in-time (JIT), wykorzystywany przez maszynę wirtualną Javy do optymalizowania kodu języka Kotlin, ma ogromny wpływ na wyniki eksperymentów. Aby uczynić prowadzone badania bardziej miarodajnymi, zdecydowano o przeprowadzeniu testów zgodnie z trzema różnymi scenariuszami. Należy dodać, że przez "pełny cykl testów", autorzy mają na myśli fakt pojedynczego wykonania każdego z sześciu przygotowanych testów dla obydwu języków - czyli przeprowadzenie łącznie dwunastu testów wraz z zapisem wyników.

Opracowano następujące scenariusze:

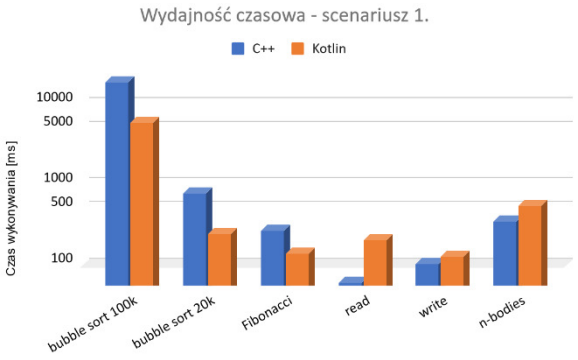
- Scenariusz nr 1 przewiduje stukrotne wykonanie pełnego cyklu testów, bez zamykania programu pomiędzy cyklami i przy standardowych ustawieniach środowiska Android, to jest z aktywnym mechanizmem kompilacji JIT.
- Scenariusz nr 2 również zakłada stukrotne wykonanie pełnego cyklu testów bez zamykania programu pomiędzy cyklami, ale w ramach tego scenariusza mechanizm kompilacji JIT jest nieaktywny.
- Scenariusz nr 3 zakłada trzynastokrotne wykonanie pełnego cyklu testów, przy czym po każdym cyklu aplikacja jest zamykana i uruchamiana ponownie. Kompilacja JIT jest aktywna.

6. Wyniki badań

Wszystkie badania zostały przeprowadzone na emulatorze systemu Android w wersji 13 (poziom API 33), dostarczonym przez środowisko Android Studio. Emulator uruchomiony był na systemie operacyjnym Windows 10, na komputerze opartym o architekturę x86 z następującymi podzespołami:

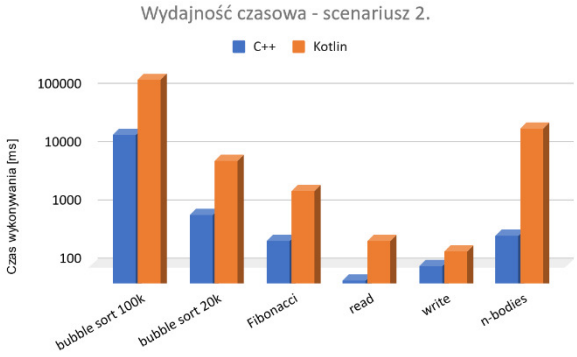
- procesor Inter Core i3 10100,
- 16 GB pamięci RAM DDR4,
- karta graficzna Nvidia GeForce 1060.

Należy dodać, że dla wszystkich trzech scenariuszy testowych, wydajność kodu C++ pozostała praktycznie niezmieniona. Główną różnicą pomiędzy scenariuszami był sposób wykorzystania mechanizmu kompilacji JIT, który ma wpływ tylko na wydajność kodu w języku Kotlin.



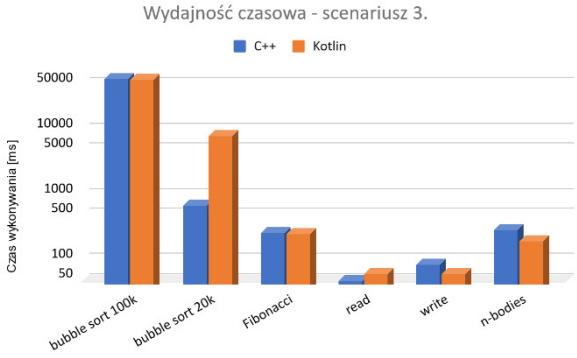
Rysunek 3: Wydajność czasowa w pierwszym scenariuszu.

W scenariuszu nr 1 Kotlin uzyskał najlepsze wyniki spośród wszystkich trzech scenariuszy (Rysunek 3). Okazał się być ponad dwukrotnie szybszy niż C++ zarówno w operacji sortowania bąbelkowego krótkiego i długiego zbioru, jak również w obliczaniu n-tego wyrazu ciągu Fibonacciego. Testy odczytu oraz zapisu do pliku, jak również symulacja n-ciał, wykonały się jednak blisko dwa razy szybciej w języku C++ niż w Kotlinie.



Rysunek 4: Wydajność czasowa w drugim scenariuszu.

W scenariuszu nr 2, którego wyniki widoczne są na powyższym rysunku (Rysunek 4) C++ okazał się być zdecydowanym zwycięzcą, pokonując Kotlina we wszystkich testach - wyłączenie mechanizmu kompilacji JIT drastycznie zmniejszyło wydajność Kotlina. Dla testów dotyczących sortowania bąbelkowego, Kotlin uzyskał ponad dwadzieścia razy gorszy wynik niż w scenariuszu nr 1, zaś dla symulacji n-ciał, wynik było około czterdzieści razy gorszy. Trzeba mieć na uwadze, że scenariusz ten raczej nie zaistniałby w warunkach normalnego użytkowania telefonu - wyłączenie mechanizmu kompilacji JIT nie ma praktycznego zastosowania poza prowadzeniem eksperymentów wydajnościowych.



Rysunek 5: Wydajność czasowa ostatniego scenariuszu.

W scenariuszu nr 3 Kotlin uzyskał wyniki zauważalnie gorsze niż w scenariuszu pierwszym, ale znacznie lepsze niż w drugim (Rysunek 5). Jak nietrudno zauważyć, różnica jest najbardziej widoczna w przypadku sortowania bąbelkowego, podczas gdy wyniki odczytu, zapisu, czy symulacji n-ciał pozostają zbliżone do wyników z pierwszego scenariusza.

7. Wnioski

Na podstawie wykonanych testów oraz analizy ich rezultatów można niemalże jednoznacznie wskazać wyższość języka C++ nad Kotlinem w dziedzinie szybkości działania, zwłaszcza w algorytmach bardziej złożonych. W zastosowaniach stosujących proste i repetetywne funkcje, a więc tam, gdzie ma zastosowanie mechanizm JIT, Kotlin wykazuje się nadspodziewaną szybkością, wykonując polecenia ponad dwukrotnie szybciej. Jakkolwiek całkowite zastąpienie Kotlina tudzież Javy przy tworzeniu aplikacji mobilnych nie jest zasadne, a wytwarzanie takowych aplikacji wiązałoby się z wykładniczo większym nakładem pracy, wskazanym i zasadnym jest zastępować język Kotlin wstawkami z C++, celem usprawnienia działania krytycznych elementów kodu, zwłaszcza tam, gdzie nie jest to związane ze współpracą z interfejsem użytkownika.

Literatura

[1] Globalne statystyki popularności wybranych systemów operacyjnych w latach 2022-2023, https://gs.statcounter.com/os-market-share, [28.06.2023]

[2] L. Corral, A. Sillitti, G. Succi. Mobile multiplatform development: An experiment for performance analysis, Procedia Computer Science 10 (2012) 736-743, https://doi.org/10.1016/j.procs.2012.06.094.

[3] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes, J. Saraiva, Energy Efficiency across Programming Languages: How Do Energy, Time, and Memory Relate, in: 10th ACM SIGPLAN International Conference (SLE'17), Vancouver, Canada, October 23–24, 2017.

[4] S. Lee, J. W. Jeon, Benchmarking Java application using JNI and native C application on Android, in: International Conference on Control, Automation and Systems (ICCAS), Gyeonggi-do, Korea (South), October 3-4, 2012, 1160-1163.

[5] M. Siggel, How to bring compute intensive C++ base apps to Android, in: Free and Open Source Conference 9 (FrOSCon) Sankt Augustin, Germany, August 23-24, 2014.

[6] What Is C++ Used For? https://www.codecademy.com/resources/blog/what-is-c-plus-plus-used-for/, [30.05.2023].

[7] Kotlin for Android, https://kotlinlang.org/docs/android-overview.html, [22.06.2023].

[8] N-body simulation – Wikipedia, https://en.wikipedia.org/wiki/N-body_simulation, [22.06.2023].