JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 135–139
Received: 30 March 2022
Accepted: 25 April 2022

C++ and Java performance on the Android platform

Wydajność języków C++ oraz Java na platformie Android

Paweł Wlazło*, Jakub Smołka

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

Abstract

The article presents a comparative analysis of Java and C ++ technologies in terms of performance on the Android platform. The purpose of this work was to point to a more efficient language for developing mobile applications. The study was carried out on custom applications. The tests concerned data sorting, prime numbers determination, bitmap modification, saving to the database and reading from a text file. The series of repetitions of each test were performed on Samsung and Xiaomi devices. The following criteria were used: test execution time, CPU load, and RAM usage. The performance, in most of the carried out tests, was in favor of the C++ language, and the main difference and with the greatest discrepancy between the technologies tested was the execution time, where C++ scored 18 points, and Java 3 points. For the CPU usage, the result was the same, but value differences were smaller. A nondiscerning parameter that was the use of RAM. C++ received 11 points and Java 10.

Keywords: Java; C++; performance; Android

Streszczenie

W artykule przedstawiono analizę porównawczą technologii Java i C++ w kontekście wydajności na platformie Android. Celem tej pracy było wskazanie wydajniejszego języka do tworzenia aplikacji mobilnych. Badania przeprowadzono na autorskich aplikacjach. Testy dotyczyły sortowania danych, wyznaczania liczb pierwszych, modyfikacji bitmapy, zapisu do bazy danych i odczytu z pliku tekstowego. Serie powtórzeń każdego testu wykonane zostały na urządzeniach marki Samsung oraz Xiaomi. Kryteria, którymi się posłużono to: czas wykonania testu, obciążenie procesora, wykorzystanie pamięci RAM. Wydajność w większości przeprowadzonych testów była na korzyść języka C++. Cechą wykazującą największe różnice między badanymi technologiami był czas wykonania, gdzie C++ uzyskał 18 punktów, a Java 3 punkty. Dla wykorzystania procesora wynik był taki sam, jednak różnice wartości mniejsze. Parametrem niewskazującym faworyta było wykorzystanie pamięci RAM. Uzyskano 11 punktów dla języka C++ i 10 punktów dla Javy.

Słowa kluczowe: Java; C++; wydajność; Android

*Corresponding author

Email address: wlazlopwl@gmail.com (P. Wlazło)

1. Wstęp

Ostatnie lata to ciągły bój o lidera w branży producentów telefonów komórkowych. Regularnie słyszy się o premierach smartfonów, które w najbliższej przyszłości trafić mają do sprzedaży. Główną zaletą, rzeczą, która wpływa na atrakcyjność danego urządzenia jest jego wydajność. I tak, kiedy producenci chwalą się lepszymi parametrami telefonu od konkurencji, tak i klienci szukając nowego, starają się wybrać ten posiadający więcej pamięci RAM (*ang. Random Access Memory*) lub mający lepszy procesor. Dla potencjalnego klienta smartfon ma po prostu działać. Inaczej jednak ma się sprawa z punktu widzenia programistów na platformę mobilną. Muszą oni tworzyć aplikacje w sposób umożliwiający jej płynne działanie. Nie może, a przynajmniej nie powinno dochodzić do sytuacji, w których aplikacja działa, ale zacina się, czy jest powolna – wtedy staje się ona wręcz bezużyteczna. Wybór rozwiązania, technologii, języka programowania nie jest sprawą łatwą i oczywistą. Przeskok technologiczny w ostatnich latach wzrósł do olbrzymich rozmiarów. Smartfony posiadają podzespoły niekiedy nawet lepsze niż niejeden laptop. Jednakże to nie rozwiązuje problemu wydajności. Wraz ze wzrostem technologii wzrasta zapotrzebowanie klientów, aplikacje mają za zadanie wykonywać co raz to bardziej skomplikowane operacje, wymagające sporych nakładów pracy podzespołów.

Poniższa praca porównuje języki C++ i Java na platformie Android. Kryteria, którymi kierowano się podczas oceny wydajności obu technologii to czas wykonania danego testu, obciążenie procesora i wykorzystanie pamięci RAM. Artykuł próbuje odpowiedzieć na pytanie, który spośród dwóch języków C++ i Java zapewnia większą wydajność aplikacji działających na platformie Android.

2. Cel i zakres badań

Celem badań jest analiza wydajności języków programowania Java i C++ na platformie Android poprzez wykonanie określonych zadań, tj. sortowanie danych, obliczanie liczb pierwszych, modyfikacja bitmapy na ekranie urządzenia, zapis do bazy danych i odczyt danych zawartych w pliku txt.

Zakres badań:

- opracowanie dwóch aplikacji w języku Java i C++ na platformę Android,
- przedstawienie metodyki badań,

- przeprowadzenie testów wydajnościowych na opracowanych aplikacjach,
- analiza wyników pomiarów: czasy wykonania, wykorzystania pamięci RAM i obciążenia procesora.

3. Przegląd literatury

W publikacji *FFT benchmark on Android devices: Java versus JNI* autorzy swoją uwagę skupili na implementacji algorytmu Szybkiej Transformacji Fouriera (*ang. Fast Fourier Transform*). Badania wykonano na 35 różnych jednostkach badawczych. Autorzy w swoich badaniach pod uwagę wzięli również obszary wielowątkowości. Zauważyli, że podczas korzystania z wielu wątków język natywny radzi sobie gorzej od języka Java, a spowodowane jest to jądrem systemu Android. Jak stwierdzają twórcy eksperymentu, korzystanie z JNI (*ang. Java Native Interface*) jest w stanie zwiększyć wydajność, ale wiąże się to ze wzrostem kosztów wywołania kodu natywnego względem języka Java, a to może negatywnie wpłynąć na działanie aplikacji [1].

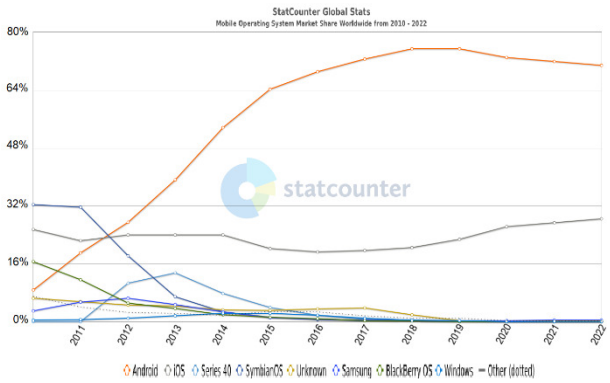
Kolejnym artykułem, w którym naukowcy starali się znaleźć odpowiedź na temat wydajności, a dokładnie odnośnie czynników wpływających na tą wydajność jest *Towards Performance-Enhancing Programming for Android Application*. Okazuje się, że to nie tylko wybór technologii w danej implementacji ma znaczenie, a czynników powodujących mniejszą bądź większą wydajność na platformie Android jest więcej. Autor Dong Kwan Kim zbadał wpływ różnych technik wykorzystywanych w programowaniu m.in. na systemy mobilne. Porównał fragmenty kodu takie jak pętla *for* w zapisie skróconym oraz pełnym, typy list, stosowanie metod rekurencyjnych, tworzenie zmiennych lokalnych, globalnych i ich wpływ na wykorzystanie procesora. Dwie, stworzone przez autora proste gry wykazały, że wybór sposobu implementacji ma znaczący wpływ na efektywność działania aplikacji na platformie Android. Nieodpowiednie, losowe stosowanie schematów zapisu funkcji przyczynić się może nawet do pogorszenia wydajności w teoretycznie lepszej technologii, a w konsekwencji zamiast przynieść korzyść, może spowodować jedynie szkody [2].

W publikacji *A performance comparision of Java and native C on Android* autorzy implementując, a następnie badając wyniki z testów, wysuwają wnioski mówiące o tym, że wydajność języków Java i C++ jest zbliżona. Według nich zapotrzebowanie na natywne technologie z każdą kolejną wydaną wersją platformy Android maleje, a jedyną potrzebą wykorzystywania narzędzia Android NDK może być użycie wcześniej stworzonego kodu w C/C++, tak by nie przepisywać danej funkcjonalności do języka Java [3].

4. Omówienie technologii

Początki systemu Android sięgają 2003 roku. Założyciele swe pierwsze kroki stawiali w Kalifornii gdzie założyli Android Inc. W 2005 roku firma Google kupiła Android Inc., jednak założyciele postanowili pozostać i wciąż rozwijać system. Platformę rozwijano z zastosowaniem jądra systemu Linux, co przyczyniło się do udostępnienia producentom telefonów komórkowych platformy Android bezpłatnie [4]. Firma Google stworzyła zestaw narzędzi dla programistów – SDK (*ang. Software Development Kit*). W 2008 roku opublikowano pierwszą wersję systemu Android [5].

Popularność systemu Android z roku na rok wzrasta. W roku 2021 udział platformy Android stanowił około 72% względem wszystkich dostępnych systemów na urządzenia mobilne [6]. Pomiędzy rokiem 2010, a 2022 system Android charakteryzował się największą popularnością (rysunek 1).



Rysunek 1: Udział w rynku mobilnych systemów operacyjnych na całym świecie pomiędzy 2010 a 2022 rokiem [6].

4.1. Android NDK i JNI

NDK to zestaw narzędzi, które stworzyła firma Google umożliwiających implementację wybranej części aplikacji Android w języku natywnym – C i C++. Według oficjalnej dokumentacji NDK, nie zaleca się korzystania ze wspomnianych języków na platformie Android, jeśli nie jest to uzasadnione [7]. Tworzenie aplikacji wyłącznie posługując się językami C i C++ przynieść może więcej szkód niż pożytku, a powstały kod stanie się nieczytelny.

JNI to tzw. natywny interfejs Javy. Umożliwia osobom tworzącym aplikację na system Android interakcję z językami natywnymi. JNI pozwala łączyć implementację w technologii Java z C czy C++. Jedną z głównych zalet omawianego rozwiązania jest możliwość wykorzystania kodu, który już wcześniej został stworzony w językach C/C++. Kolejną z zalet jest szybkość działania, a opóźnienie komunikacji JNI przy przekazywaniu ciągu znaków do aplikacji wynosi zaledwie 0,15 mikrosekundy [8]. Na listingu 1 przedstawiono metody JNI wykorzystane w badaniach.

Listing 1 Deklaracje metod JNI w języku Java (opracowanie własne)

```
public native String sorting(int checkedPosition);

public native void calculatePrimeNumbers(int range);

public native Bitmap rotateBitmap(Bitmap bitmap);

public native void initDb(String path);

public native String saveToFile(AssetManager mgr);
```

5. Metoda badań

Testy umożliwiające dokonania analizy przeprowadzono na trzech jednostkach badawczych – urządzeniach z systemem operacyjnym Android. Wykorzystane urządzenia to:

- Samsung A5 2016,
- Samsung A5 2017,
- Xiaomi Mi 11 Lite 4g.

Ponadto, w celu odczytania poszczególnych parametrów niezbędne okazało się użycie kolejnej jednostki badawczej – laptopa (Asus).

5.1. Opis testów

Każdy test wydajnościowy zawiera dwie implementacje, dla każdej technologii z osobna. Badania przeprowadzono przy użyciu pięciu rodzajów testów.

Pierwszy test zawiera implementacje pozwalające utworzyć zbiór liczb w kolejności malejącej, a następnie na posortowaniu uzyskanego zbioru w kolejności odwrotnej – rosnącej. Implementacja pozwala użytkownikowi wybrać 3 dostępne wielkości danych do sortowania, kolejno 25 000, 35 000 i 50 000. Do sortowania danych wykorzystano algorytm bąbelkowy, który cechuje się złożonością czasową równą $O(n^2)$, a to pozwala urządzeniom dokonywać dłuższych obliczeń wymagających większego wykorzystania podzespołów w porównaniu do innych algorytmów.

Kolejny test umożliwia wyznaczanie liczb pierwszych z podanego przedziału. W implementacji wykorzystano algorytm sita Eratostenesa. Eksperyment umożliwia wybranie losowej liczby z zakresu od 1 do 100 000 000, po czym rozpoczyna się proces detekcji liczb pierwszych, gdzie wykorzystanie parametrów jednostki badawczej zależne jest od wybranej wielkości przedziału.

Modyfikacja obrazu to kolejny test wykorzystany w badaniach. Stworzona metoda pozwala na kilkukrotny obrót bitmapy o 90 stopni w prawą stronę.

Kolejny, czwarty test pozwala na zmierzenie wydajności zapisu do bazy danych. Baza ta zawiera jedną tabelę z pięcioma kolumnami typu Integer.

Piąty, ostatni z założonych testów dotyczy odczytu danych zapisanych w pliku txt**.** W pliku znajduje się 1 180 452 linii tekstu (wartość losowa), a każda z nich zawiera 200 znaków – małych i dużych liter alfabetu. Przyjęto dany rząd wielkości ze względu na konieczność stosownego obciążenia urządzeń.

5.2. Warunki dotyczące testów

Przeprowadzenie badań wiąże się z koniecznością zachowania spójnych warunków pozwalających w końcowym etapie uzyskać miarodajne wyniki. W jednostkach badawczych przed przystąpieniem do testów wyłączono funkcjonalności takie jak: Wi-Fi, GPS, dane komórkowe, Bluetooth. W każdym z urządzeń pozostawiono jedynie aplikacje systemowe (i te odpowiedzialne za przeprowadzenie testów). Po każdej z prób wymuszano zakończenie procesów działających w tle. Każdy z zaimplementowanych testów powtórzono 10-krotnie. Działania te podyktowane były zamiarem wyeliminowania niepożądanych i zaburzonych wyników uniemożliwiających rzetelną analizę.

5.3. Opis pomiaru czasu

W badaniach, dla pomiaru czasu zdecydowano się wykorzystać metodę *System.currentTimeMillis.* Wywoływała się ona tuż przed rozpoczęciem testu i ponownie zaraz po jego zakończeniu. Posiadając pobrane dokładne czasy odejmowano pierwszy wynik od drugiego uzyskując wynik.

5.4. Wykorzystanie CPU i RAM

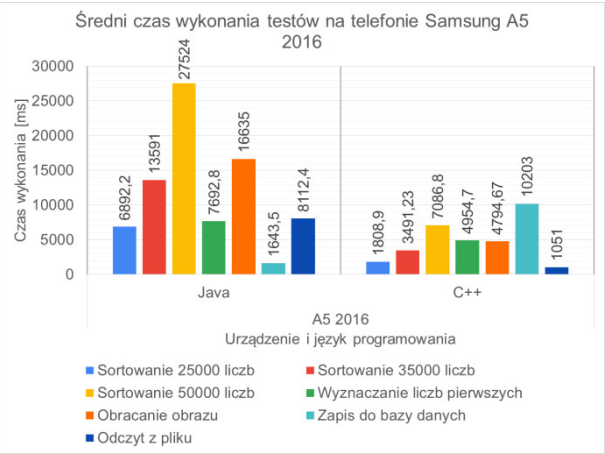
Obciążenie procesora i wykorzystanie pamięci operacyjnej RAM zmierzono za pomocą wbudowanego w kompilator Android Studio narzędzia o nazwie Android Profiler, dzięki któremu możliwe jest śledzenie wyżej wymienionych parametrów w czasie rzeczywistym zarówno w formie wykresów jak i wartości liczbowo-procentowych.

6. Analiza wyników

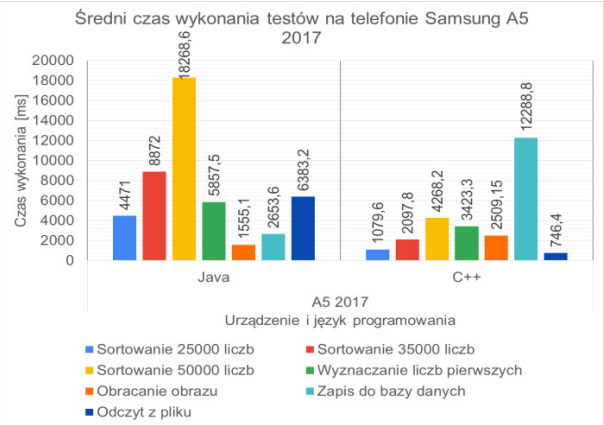
Po uzyskaniu wyników przydzielano punkty odnoszące się do każdego testu, języka oraz każdej jednostki badawczej z osobna. Badane parametry w danej technologii otrzymywały 0, 1 bądź 0,5 punktu. W każdym przypadku pod uwagę brano wyniki z jednego testu oraz odpowiedniego parametru w obu językach, a następnie porównywano wyniki, gdzie lepszy otrzymywał 1 punkt, natomiast gorszy 0. W analizie przyjęto również możliwość uzyskania bardzo zbliżonych wyników w obu badanych językach. Różnica wyników mniejsza niż 1% decydowała o przyznaniu dla danego testu i urządzenia po 0,5 punktu dla obu języków.

6.1. Czas wykonania

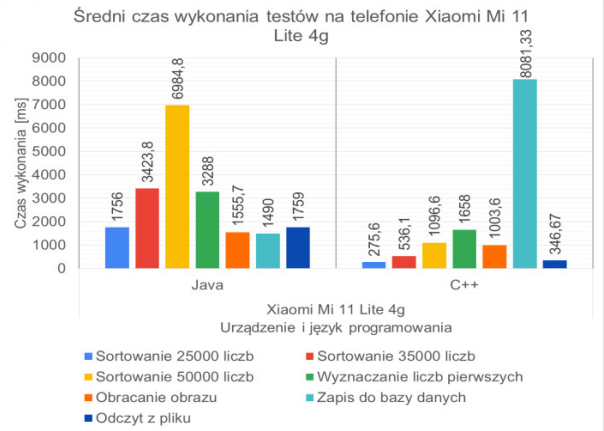
Czas wykonania danego zadania zależny jest od jednostki badawczej. Nowsze urządzenie, posiadające lepsze parametry, więcej dostępnej pamięci, wydajniejszy procesor osiąga lepsze – krótsze czasy niezbędne do ukończenia zaimplementowanego testu.



Rysunek 2: Średni czas wykonania testów na telefonie Samsung A5 2016 (opracowanie własne).



Rysunek 3: Średni czas wykonania testów na telefonie Samsung A5 2017 (opracowanie własne).

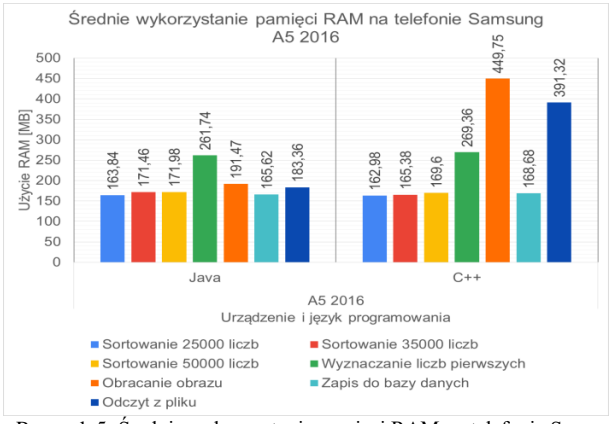


Rysunek 4: Średni czas wykonania testów na telefonie Xiaomi Mi 11 Lite 4g (opracowanie własne).

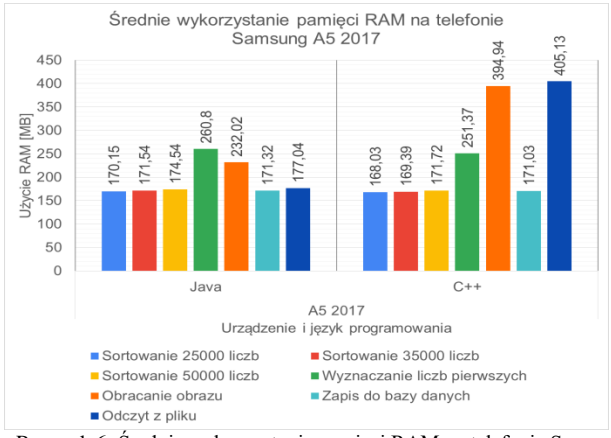
Na rysunkach 2, 3 i 4 widać, że wykorzystanie urządzenia z lepszymi parametrami powoduje krótszy czas wykonania. W większości testów język C++ wykonał zadania szybciej. Zarówno w urządzeniach marki Samsung jak i Xiaomi zapis do bazy danych okazał się być bardziej czasochłonny w języku C++ niż w Java. Ponadto telefon Samsung A5 2017 wymagał nieznacznie, ale jednak więcej czasu na ukończenie testu związanego z obracaniem bitmapy. Wszystkie pozostałe zaimplementowane testy w krótszym czasie wykonały się w języku C++.

6.2. Wykorzystanie RAM

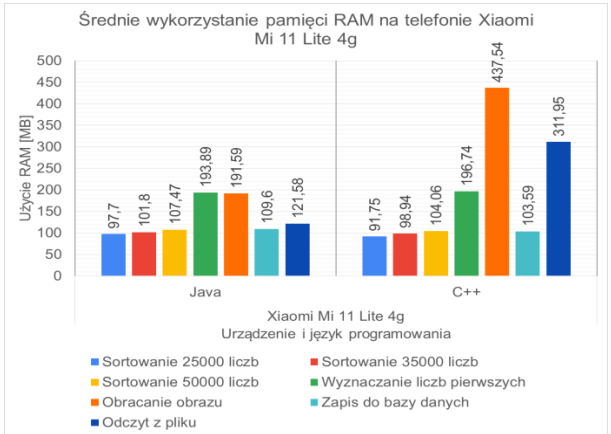
Wykorzystanie pamięci operacyjnej RAM było zróżnicowane, zależne od wykonywanego testu. Porównując słupki na rysunkach 5, 6 i 7 zauważa się, że niektóre z testów (np. sortowanie lub zapis do bazy danych) w obu technologiach uzyskały zbliżone wartości, a inne (np. obracanie obrazu) różniły się znacznie – niekiedy dwukrotnie. W przypadku pomiarów testu dotyczącego zapisu do bazy danych na urządzeniu Samsung A5 2017 oraz sortowania zbioru 25 000 liczb różnice nie przekraczały 1%. W tych konkretnych sytuacjach uznano remis obu technologii przyznając po 0,5 punktu dla każdej z nich.



Rysunek 5: Średnie wykorzystanie pamięci RAM na telefonie Samsung A5 2016 (opracowanie własne).



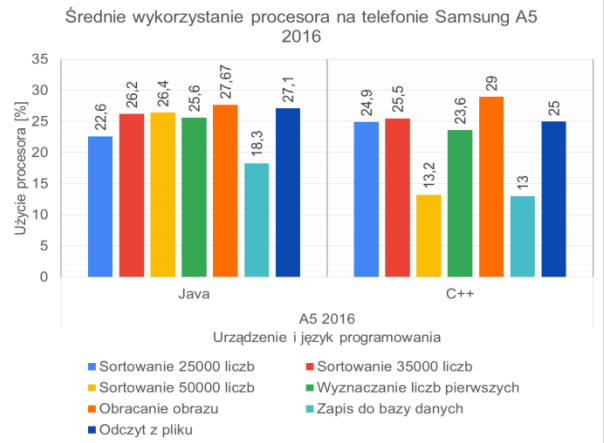
Rysunek 6: Średnie wykorzystanie pamięci RAM na telefonie Samsung A5 2017 (opracowanie własne).



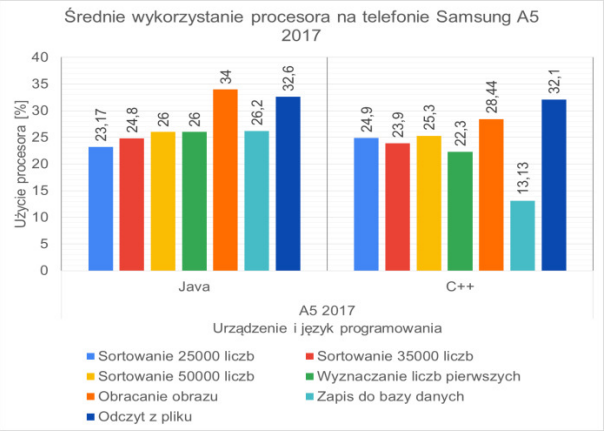
Rysunek 7: Średnie wykorzystanie pamięci RAM na telefonie Xiaomi Mi 11 Lite 4g (opracowanie własne).

6.3. Obciążenie procesora

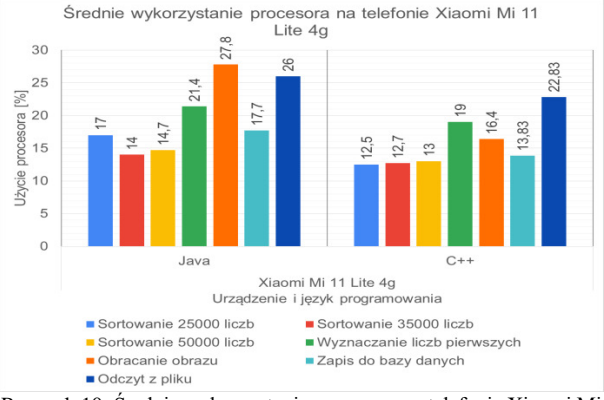
Podczas określania wydajności urządzenia, obok czasu wykonania i wykorzystania pamięci RAM, istotnym parametrem jest obciążenie procesora. Poniższe rysunki (8, 9, 10) są graficznym odzwierciedleniem średnich wyników uzyskanych podczas pomiarów obciążenia procesora na jednostkach badawczych wykorzystanych podczas badań. Obciążenie procesora w każdym z badanych urządzeń w obu technologiach kształtowało się na podobnym poziomie. W niektórych przypadkach na korzyść języka C++, w innych – Java. Sumarycznie, mimo małych różnic, pierwszy z nich był lepszy w 18 przypadkach, a drugi jedynie 3 razy.



Rysunek 8: Średnie wykorzystanie procesora na telefonie Samsung A5 2016 (opracowanie własne).



Rysunek 9: Średnie wykorzystanie procesora na telefonie Samsung A5 2017 (opracowanie własne).



Rysunek 10: Średnie wykorzystanie procesora na telefonie Xiaomi Mi 11 Lite 4g (opracowanie własne).

7. Wnioski

Badania przeprowadzone w niniejszej publikacji pozwalają stwierdzić, że w programowaniu aplikacji na platformę Android przewagę pod względem wydajnościowym ma język C++. Biorąc pod uwagę wszystkie wykonane testy oraz wszystkie jednostki badawcze, które brały udział w testach, język C++ uzyskał 46 punktów, natomiast język Java zaledwie 17. Zatem przeprowadzone testy wskazały, że wydajniejsza jest technologia C++ w aspekcie systemu Android.

Badania wykazały, że wpływ na wydajność ma nie tylko urządzenie i podzespoły, które kryją się w jego wnętrzu. Duży wpływ ma również rodzaj testu. To, że język C++ sortował dane w sposób wydajniejszy, w żaden sposób nie oznacza, że z porównywalną wydajnością będzie w stanie umieszczać i zapisywać w bazie danych. Programista powinien mieć świadomość, że technologia C++ jest wydajniejsza, ale nie nadaje się do wszystkiego. Zgubne może okazać się wykorzystywanie języka C++ do każdego złożonego obliczeniowo zadania. Mówi to zarówno dokumentacja Android [9] jak i przeprowadzone badania.

Obok wydajności, uwagę należy również zwrócić na próg wejścia i tworzenia aplikacji za pomocą języka C++ bądź narzędzia NDK. Próba tworzenia całych aplikacji w języku natywnym nie jest zalecana [7]. Osoba, szczególnie ta niedoświadczona powinna z rozwagą podejść do tematu wydajności i wiedzieć, że zgubnym jest usilna chęć wytworzenia oprogramowania bardziej wydajnego, kosztem dużo większych nakładów pracy, gdzie finalnie okazać może się, że niewystarczająca wiedza i nietrafny wybór mogą jeszcze obniżyć wydajność bądź w najgorszym wypadku doprowadzić do niepowodzenia i nieukończenia wytwarzania danej aplikacji na platformę Android.

Literatura

[1] A. Carvalho, M. Rosan, A. Bianchi, M. Queiroz, FFT benchmark on Android devices: Java versus JNI, Proceedings of the 14th Brazilian Symposium on Computer Music, Brasilia, Brazil (2013) 4-7.

[2] D. K. Kim, Towards performance-enhancing programming for Android application development, International Journal of Contents 13 (2017) 39-46.

[3] A. Ulvesand, D. Eriksson, Native code on Android: A performance comparison of Java and native C on Android. Bachelor's thesis at NADA, KTH Royal Institute of Technology (2011).

[4] J. Annuzzi, L. Darcey, S. Conder, Android. Wprowadzenie do programowania aplikacji, Helion, Gliwice, 2016.

[5] Historia i ewolucja systemu Android, https://www.androidauthority.com/history-android-os-name-789433/, [08.01.2022].

[6] Udział mobilnych systemów operacyjnych na świecie, https://gs.statcounter.com/os-market-share/mobile/worldwide/#yearly-2010-2022, [08.01.2022].

[7] Oficjalna dokumentacja Android NDK, https://developer.android.com/ndk, [08.01.2022].

[8] S. Lee, J.W Jeon, Evaluating performance of Android platform using native C for embedded systems, Proceedings of the International Conference on Control, Automation and Systems, ICCAS 2010, Gyeonggi-do, South Korea (2010) 1160-1163, https://doi.org/10.1109/ICCAS.2010.5669738.

[9] Oficjalna dokumentacja Android, https://developer.android.com/docs, [10.01.2022].