JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 25 (2022) 409–416
Received: 13 September 2022
Accepted: 29 September 2022

Swift performance in statistical aplications

Wydajność języka Swift w zastosowaniach statystycznych

Sylwester Tylec*, Karol Woś

Abstract
This paper aims to test Swift's performance against C++ in performing statistical calculations. The analyzed issues, apart from performance, are code transparency and syntax, libraries available for these languages and the use of device hardware resources during testing. For this purpose, a comparative analysis of the two above-mentioned languages was carried out, based on the results obtained from a series of experiments carried out with the use of specially developed test applications. The tests consisted in calculating the standard deviation, median and arithmetic, harmonic and geometric mean, and during the tests, execution times, operating memory usage and CPU load were recorded. Based on the results of the research, it was found that the Swift language is not optimized for statistical calculations.

Keywords: Swift; statistics; performance; analysis

Streszczenie
Niniejsza praca ma na celu przeprowadzenie testów wydajności języka Swift w porównaniu z językiem C++ przy wykonywaniu obliczeń statystycznych. Analizowanymi zagadnieniami poza wydajnością są przejrzystość i składnia kodu, biblioteki dostępne dla tych języków oraz wykorzystanie zasobów sprzętowych urządzenia podczas przeprowadzania testów. W tym celu przeprowadzono analizę porównawczą dwóch wyżej wymienionych języków, opierającą się na wynikach uzyskanych z serii eksperymentów przeprowadzonych przy użyciu specjalnie utworzonych aplikacji testowych. Testy polegały na liczeniu odchylenia standardowego, mediany i średnich arytmetycznej, harmonicznej i geometrycznej a w trakcie testów rejestrowano czasy wykonania, użycie pamięci operacyjnej i obciążenie procesora. Na podstawie wyników badań ustalono, że język Swift nie jest zoptymalizowany pod kątem obliczeń statystycznych.

Słowa kluczowe: Swift; statystyka; wydajność; analiza

*Corresponding author
Email address: **sylwester.tylec@pollub.edu.pl** (S. Tylec)

1. Wstęp

Programy komputerowe pomagają użytkownikom wykonywanie wielu czynności, takich jak obliczenia, symulacje, zarządzanie danymi, obróbka grafiki i wiele innych. Jednak, aby program jak najwydajniej pełnił swoją rolę, musi zostać napisany przy pomocy odpowiedniego i najbardziej adekwatnego do powierzonego zadania języka z wykorzystaniem jego bibliotek i funkcji. Mnogość języków wraz z dostępne dla nich narzędzia mogą przysparzać trudności podczas ich wyboru. Pod uwagę należy wziąć zadania, jakie zostaną podstawione przed programem.

Język Swift świetnie nadaje się do przeprowadzenia analizy ze względu na obiektowość, która jest obecnie najpopularniejszym podejściem podczas tworzenia aplikacji. Dodatkowym atutem przy uwzględnieniu tego języka jest fakt, że jest on stosowany zarówno na urządzeniach mobilnych, jak i desktopowych firmy Apple. Język C++ jest językiem znacznie starszym, gdyż powstał w roku 1985. Niemniej jednak w dalszym ciągu cieszy się dużą popularnością ze względu na dużą szybkość i wydajność, a wiele frameworków i narzędzi tworzonych oraz używanych w dzisiejszych czasach bazuje na tymże języku.

Znaczna część użytkowników sprzętu komputerowego jest w posiadaniu co najmniej dwóch urządzeń mogących służyć wykonywaniu obliczeń. Rosnąca popularność rozwiązań mobilnych sprawia, że coraz większą liczbę działań można przeprowadzić na sprzęcie przenośnym, który zazwyczaj nie dysponuje tak dużą mocą obliczeniową, jak urządzenia stacjonarne. Warto jest więc zastanowić się, jaki sprzęt i jakie narzędzia należy wybrać, aby jak najlepiej spełniały oczekiwania względem wydajności. Przeprowadzenie porównania i analizy wydajnościowej wymaga doboru języka, który z założenia może zostać wykorzystywany na takich samych płaszczyznach.

Niniejsza praca została poświęcona analizie porównawczej języków C++ i Swift. Porównanie będzie obejmować równoległe wykonywanie obliczeń statystycznych na tych samych danych źródłowych, co pozwoli na zbiorcze zestawienie wyników przeprowadzonych badań. Wśród nich zawierać się będą czasy przeprowadzenia operacji oraz obciążenie systemu podczas działania programu.

2. Cel badań

Niniejszy artykuł ma na celu przeprowadzenie analizy wydajności języka Swift pod kątem obliczeń statystycznych. W tym celu dokonano porównania analogicznych programów stworzonych w języku Swift i C++. Głównymi parametrami, które były brane pod uwagę przy analizie porównawczej były obciążenie procesora, zużycie pamięci RAM i czas wykonania programów. Pozostałymi badanymi aspektami podczas porównywania były budowa i składnia języka, dostępność bibliotek oraz popularność języków wśród społeczności.

3. Metoda badawcza

W celu przeprowadzenia analizy wydajności autorzy niniejszej pracy badawczej utworzyli trzy aplikacje różniące się stopniem złożoności. Zostały one zaimplementowane dla obu badanych języków. Aplikacje wykorzystują te same algorytmy i zostały napisane w możliwie identyczny sposób. Pierwsza z nich ma za zadanie obliczyć odchylenie standardowe, druga natomiast sortuje tablicę z danymi w celu znalezienia mediany, trzecia oblicza średnią arytmetyczną, geometryczną i harmoniczną. Podczas wykonywania badań zostały użyte te same zestawy danych, które były wygenerowane losowo w liczbie odpowiednio 100, 1000 i 10000. Łącznie przeprowadzono 82 pomiary, przy czym dla każdego zestawu danych w przypadku aplikacji na oba języki przeprowadzono po 7 pomiarów.

Badania odbywały się na urządzeniu testowym, którego specyfikacja przedstawiona jest w Tabeli 1. Oba programy zostały napisane i uruchomione w środowisku Xcode w wersji 13.2.1.

Tabela 1: Specyfikacja urządzenia testowego

Model urządzenia	MacBook Pro MYD92ZE/A/R1
Procesor	Apple M1
Pamięć RAM	16 GB
System operacyjny	macOS Monterey
Dysk	512 GB SSD PCIe
Rok premiery	2020

Na urządzeniu zostały wyłączone wszelkie niepotrzebne procesy oraz było ono podłączone do źródła zasilania, aby na czas przeprowadzenia pomiarów zasoby sprzętowe nie były niczym ograniczone, co zapewniło powtarzalne warunki testowe.

Czas wykonania programów mierzony był przy użyciu dostępnych w standardowych bibliotekach funkcji, które umożliwiają precyzyjne odmierzanie czasu. Dla języka Swift zastosowano funkcję „CFAbsoluteTimeCurrent()", natomiast w języku C++ była to funkcja „chrono::steady_clock::now()". Pozostałe parametry, czyli obciążenie wszystkich rdzeni procesora oraz zużycie pamięci operacyjnej mierzone były za pomocą wbudowanego w system operacyjny monitora aktywności. Podczas badania wydajności pod uwagę były brane najwyższe wartości dotyczące wykorzystania zasobów sprzętowych dla procesu odpowiadającego za dany program, z których następnie wyciągnięto średnią z pomiarów.

Strukturę kodu i składnię języka porównano posiłkując się oficjalną dokumentacją dostępną dla obu języków. Według autorów odpowiednim kryterium do zbadania popularności wśród społeczności była liczba wyników zapytań odnoszących się do danego języka w serwisach GitHub oraz Stack Overflow.

Algorytmy użyte w programach wykorzystanych w badaniach były napisane w najprostszy możliwy sposób w celu zmniejszenia liczby operacji. Fragmenty aplikacji zostały przedstawione na Listingach 1, 2, 3, 4, 5 oraz 6.

Listing 1: Fragment aplikacji liczącej średnie w języku C++

```cpp
double MeanA(double data[]) {
    double sum = 0.0, meanA;
    int i;
    for(i = 0; i < 10; ++i) {
    sum += data[i];
    }
    meanA = sum / 10;
    return meanA;
}
double MeanG(double data[]){
    double iloczyn = 1.0, meanG=0;
    int i;
    for(i = 0; i < 10; ++i) {
    iloczyn *= data[i];
    }
    meanG=pow(iloczyn, 1.0/10);
    return meanG;
}
double MeanH(double data[]) {
    double sum = 0.0, meanH;
    int i;
    for(i = 0; i < 10; ++i) {
    sum += 1/data[i];
    }
    meanH = 10/sum;
    return meanH;
}
```

Listing 2: Fragment aplikacji liczącej średnie w języku Swift

```swift
func meanA(arr: [Double])->Double{
    var suma, mean: Double
    mean=0
    suma=0
    for i in 0...n{
        suma+=arr[i]
    }
    mean=suma/g
    return mean
}
func meanG(arr: [Double])->Double{
    var iloczyn, meanG: Double
    meanG=0
    iloczyn=1
    for i in 0...n{
        iloczyn*=arr[i]
    }
    meanG=pow(iloczyn, (1/g))
    return meanG
}
func meanH(arr: [Double])->Double{
    var suma, meanH: Double
    meanH=0
    suma=0
    for i in 0...n{
        suma+=1/arr[i]
    }
    meanH=g/suma
    return meanH
}
```

Listing 3: Fragment aplikacji liczącej odchylenie standardowe w języku Swift

```
23      var suma, mean, standardDeviation: Double
24      standardDeviation=0
25      mean=0
26      suma=0
27      for i in 0...n{
28          suma+=arr[i]
29      }
30      mean=suma/g
31      for i in 0...n{
32          standardDeviation+=pow(arr[i]-mean, 2)
33      }
```

Listing 4: Fragment aplikacji liczącej odchylenie standardowe w języku C++

```
24 double calculateSD(double data[]) {
25   double sum = 0.0, mean, standardDeviation = 0.0
26   int i;
27
28   for(i = 0; i < 10; ++i) {
29     sum += data[i];
30   }
31
32   mean = sum / 10;
33
34   for(i = 0; i < 10; ++i) {
35     standardDeviation += pow(data[i] - mean, 2);
36   }
```

Listing 5: Fragment aplikacji liczącej medianę w języku Swift

```
13 func calculateMedian(arr: [Double]) -> Double {
14 let sorted = arr.sorted()
15 if sorted.count % 2 == 0 {
16 return Double((sorted[(sorted.count / 2)]
17     + sorted[(sorted.count / 2) - 1])) / 2
18 } else {
19 return Double(sorted[(sorted.count - 1) / 2])
20 }
21 }
```

Listing 6: Fragment aplikacji liczącej medianę w języku Swift

```
26  double mediana (double data[])
27  {
28      double mediana;
29      int n=10;
30      if (n%2 == 0)
31      {
32          mediana = data[(n-1)/2]+data[n/2];
33          mediana = mediana/2;
34      }
35      else
36      {
37          mediana = data [n/2];
38      }
39      return mediana;
40  }
```

4. Przegląd literatury

W celu dokonania analizy wydajności języka Swift przeprowadzono przegląd literatury dotyczący powiązanych zagadnień. Zważywszy na to, iż istnieje niewiele prac, w których autorzy porównują bezpośrednio język Swift z C++, użyta do napisania niniejszego artykułu literatura dotyczy zagadnień związanych z porównywaniem dwóch języków, dzięki czemu można było dobrać odpowiednie metody badawcze.

Istotnym źródłem wiedzy użytecznej podczas pisania tej pracy były oficjalne dokumentacje techniczne używanych języków [1, 2]. Znajdują się w nich wszelkie niezbędne informacje opisujące budowę języka, podstawowe biblioteki i instrukcje dla programistów, na podstawie których można zestawić ze sobą oba języki pod kątem składni, stopnia skomplikowania, występowania typów zmiennych i intuicyjności pisania kodu.

Posługując się wiedzą zawartą w książkach do nauki poszczególnych języków w naszym przypadku są to: „Swift. 4 Koduj jak mistrz” autorstwa J. Hoffmana [3] i „Język C++. Szkoła programowania” autorstwa S. Prata [4]. Mając na uwadze zróżnicowane poziomy trudności w nauce tych dwóch języków należy zastanowić się, który będzie lepszym wyborem w przypadku ukierunkowania na obliczenia statystyczne. Książki te przydatne były do analizowania składni kodu i praktyk stosowanych podczas tworzenia programów w tych językach.

Autorom niniejszej pracy nie udało się znaleźć publikacji poruszającej temat wykorzystania języka Swift w statystyce. Prace, które przykuły uwagę zajmowały się analizą i porównaniem innych języków, a ich konkluzją było przedstawienie ich zalet i wad. Pracami, w których autorzy porównywali ze sobą dwa zagadnienia istotnym kryterium był czas wykonywania i analiza kodu. „Analiza porównawcza wydajności frameworków Angular oraz Vue.js” [5], której autorami są R. Baida, M. Andriienko i M. Plechawska-Wójcik w dużym stopniu skupia się na pomiarze czasu działania dwóch aplikacji internetowych. Autorzy badali czas wysyłania żądań do serwera i czas renderowania stron www, a w wynikach przedstawili oni silne i słabe strony porównywanych rozwiązań.

Pracami o podobnej tematyce są „A Comparative Study: Java vs Kotlin Programming in Android Application Development” [6] oraz „Comparative Analysis of Python and Java for Beginners” [7]. Jedynym artykułem, który w jakiś sposób omawiał język Swift jest praca „Speed Performance Between Swift and Objective-C” autorstwa H. Singha [8]. Autor porównywał język Swift z jego prekursorem, czyli Objecive-C. Autor na podstawie porównania do innych języków wskazał zalety języka Swift i zaprezentował go jako nowy, intuicyjny i przyjazny programiście język, który ma z powodzeniem zastąpić objective-C.

Praca „Analiza porównawcza języków Kotlin i Java używanych do tworzenia aplikacji na system Android” autorstwa D. Sulowskiego i G. Kozieła opublikowana w grudniu 2019 traktuje o podobnym zagadnieniu [9]. Autorzy porównywali dwa języki programowania pod kątem ich wydajności i użyteczności. Pod uwagę brane były takie kryteria, jak obciążanie procesora, użycie pamięci RAM oraz czasy wykonania programów.

Są to typowe prace, w których został poruszony temat porównania dwóch bezpośrednio konkurujących ze sobą aspektów w dziedzinie informatyki. Autorzy w swoich pracach badali wykorzystanie zasobów przez

badane technologie, mierzyli czas potrzebny na wykonanie czynności będących celem badań oraz oceniali praktyczność i możliwości zastosowań tychże technologii. Zatem mogą stanowić ona dobry przykład na to, jak należy przeprowadzić podobne porównanie i w jaki sposób zinterpretować wyniki otrzymane podczas przeprowadzania badań oraz postawić odpowiednią hipotezę badawczą.

5. Porównanie cech badanych języków

5.1. Różnice w budowie języków

Pierwszą wyraźną różnicą jest inna składania języków. Jak każdy język programowania Swift i C++ są w jakimś stopniu do siebie podobne, jednak należy zauważyć różnice w sposobie deklaracji zmiennych, budowania prototypów funkcji, wywoływania funkcji w programie, operacjach na tablicach. W badanych programach różnice w budowie kodu są niewielkie i zaawansowane opanowanie tworzenia programów w języku C++ pozwala w miarę łatwo i intuicyjnie posługiwać się językiem Swift.

Najważniejsze różnice w składni porównywanych języków:

- język Swift charakteryzuje się bezwzględną przestrzenią nazw automatycznie obejmującą wszystkie typy zmiennych,
- W języku C++ trzeba zawsze zadeklarować przestrzeń nazw, z której pochodzi dana metoda. Uproszczona wersja została zaprezentowana na Listingu 7.

Listing 7: Deklaracja przestrzeni nazw w języku C++

```
5  using namespace std;
```

- W języku Swift trzeba zadeklarować, czy dana będzie stała, czy zmienna. Stałą wartość oznacza się deklaracją „let" a zmienną „var". Inicjalizowanie stałej w języku Swift należy przeprowadzić od razu, w przeciwnym razie kompilator będzie proponował zmianę użytkownikowi. Dodatkowo po znaku dwukropka należy przypisać typ danej. Przykład został pokazany na Listingu 8,
- W języku C++ domyślnie wszystkie dane są traktowane jako zmienne. Stałe dane trzeba wyraźnie zadeklarować. Na Listingu 9 pokazano deklarację typów danych w języku C++.

Listing 8: Deklaracja typów danych w języku Swift

```
8  var a: Int
9  let b: Double = 1
```

Listing 9: Deklaracja typów danych w języku C++

```
8  int a;
9  const int b=9;
```

- Deklaracja tablic nie różni się od siebie w znaczący sposób. W przypadku obu języków przy deklaracji typu danych należy umieścić znak kwadratowych nawiasów. Na Listingu 10 zaprezentowano przykład deklaracji tablicy w języku Swift,
- W przypadku języka Swift zawartość tablicy umieszcza się w kwadratowych nawiasach, natomiast w języku C++ w nawiasach klamrowych. Na Listingu 11 zaprezentowano przykład deklaracji tablicy w języku C++.

Listing 10: Deklaracja tablicy w języku Swift

```
12  let tablica: [Double]
13  tablica=[1,2,3,4,5]
```

Listing 11: Deklaracja tablicy w języku C++

```
17  double data[]={1,2,3,4,5};
```

- Badane języki używają bardzo zbliżonych deklaracji pętli For. W przypadku języka Swift operacji na iteratorze nie umieszcza się w nawiasach, operacje inkrementacji i dekrementacji wykonują się automatycznie w ustalonym przedziale. Deklaracja pętli For w języku Swift przedstawiona jest na Listingu 12,
- Zmienna iterowana w pętli for w języku C++ może być zadeklarowana wewnątrz pętli jak i przed nią, natomiast w języku Swift nie można umieścić deklaracji wewnątrz pętli, trzeba używać już istniejącej zmiennej. Deklaracja pętli For w języku C++ przedstawiona jest na Listingu 13.

Listing 12: Deklaracja pętli For w języku Swift

```
17      for i in 0...n{
18          suma+=arr[i]
19      }
```

Listing 13: Deklaracja pętli For w języku C++

```
27    for(i = 0; i < 10; ++i) {
28      sum += data[i];
29    }
```

- Deklarację funkcji w języku Swift należy zacząć od słowa „func". Tak jak w większości innych języków argumenty funkcji znajdują się po jej nazwie w nawiasach okrągłych. Typ zmiennej zwracanej przez funkcję umieszcza się po strzałce. Przykład deklaracji funkcji w języku Swift przedstawiony został na Listingu 14,
- Deklaracja funkcji w języku C++ rozpoczyna się od przypisaniu typu danej zwracanej przez tą funkcję, po której znajduje się nazwa funkcji. W okrągłych nawiasach podobnie jak w przypadku języka Swift umieszcza się argumenty funkcji. Przykład deklaracji funkcji w języku Swift przedstawiony został na Listingu 15.

Listing 14: Deklaracja funkcji w języku Swift

```
13  func calculateSD(arr: [Double])->Double{
```

Listing 15: Deklaracja funkcji w języku C++

```
22  double calculateSD(double data[]) {
```

- Deklaracje klas i struktur w obu porównywanych językach nie różnią się znacząco od siebie. W obu przypadkach deklarację poprzedza słowo kluczowe „struct” lub „class”. Zawartość stanowić mogą zmienne różnego typu z odpowiednimi specyfikatorami dostępu. W swifcie domyślnym specyfikatorem, czyli takim, który ustawia się automatycznie w przypadku braku przyznania wartości przez użytkownika jest Internal access. Zezwala on na dostęp do danego pola w obrębie całego projektu, czyli z poziomu pozostałych klas i funkcji. Domyślnym specyfikatorem w C++ jest private, co sprawia, że do tak oznaczonego pola nie ma dostępu spoza klasy. Definicje metod i konstruktorów nie różnią się od siebie, a do poszczególnych pól klas i struktur można odwołać się przez znak kropki następujący po utworzonym obiekcie.

5.2. Popularność wśród społeczności

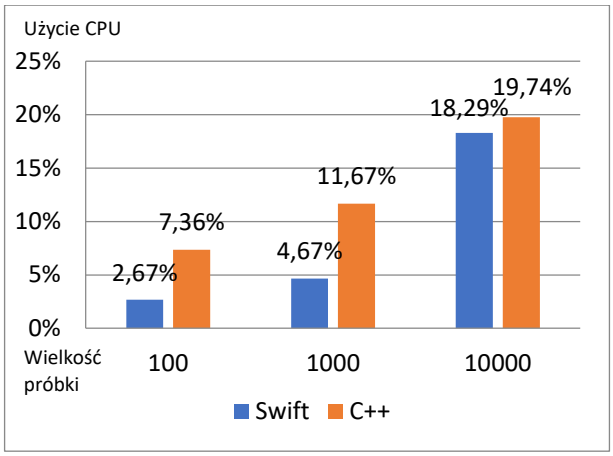
Na podstawie zapytań o język Swift na popularnych platformach dostępnych w Internecie w czerwcu 2022 roku uzyskano około 32 miliony wyników w wyszukiwarce Google, ponad 314 tysięcy pytań o język Swift na stronie Stack Overflow oraz 241 tysięcy wyszukań repozytoriów na portalu GitHub. Z kolei dla języka C++ jest to kolejno około 105 milionów, 768 tysięcy i 911 tysięcy. Świadczy to o dużo większym zainteresowaniu i wsparciu społeczności dla języka C++. Powodem może być wiek i ugruntowana pozycja na rynku. C++ to język o wielu zastosowaniach na różnych platformach, w odróżnieniu od języka Swift, będącego językiem stosunkowo młodym i używanym w znakomitej większości na urządzeniach firmy Apple.

6. Wyniki

Na podstawie wyników przeprowadzonych badań obliczono średnie wartości dla pomiarów poszczególnych parametrów. Odpowiednio dla każdego zestawu danych liczących 100, 1000 i 10000 wartości przedstawione zostały uśrednione wyniki z przeprowadzonych pomiarów. Jeden z programów miał za zadanie obliczyć odchylenie standardowe, natomiast drugi wyznaczał medianę danego zbioru liczb.

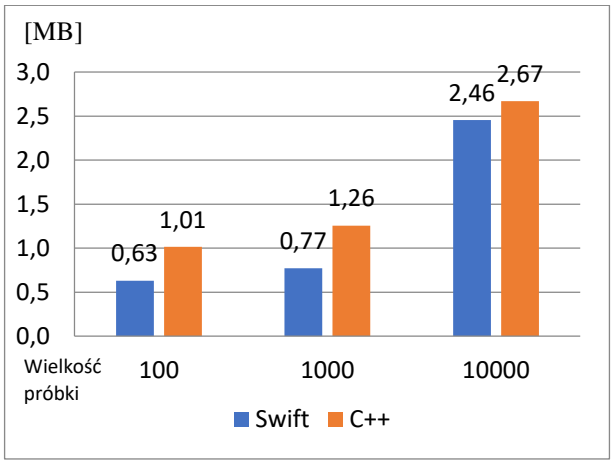
6.1. Seria pomiarowa 1

Pierwszy z omawianych programów miał za zadanie wyliczyć odchylenie standardowe dla każdego zestawu danych wprowadzonych statycznie do tablicy. Na Rysunku 1 przedstawione zostało średnie obciążenie procesora. Wyraźnie widać różnicę w wykorzystanej mocy obliczeniowej dla dwóch mniejszych zestawów danych, natomiast przy największym zestawie danych ta różnica jest znacząco mniejsza. Należy więc zauważyć, że wraz ze wzrostem ilości danych użycie procesora w przypadku obu badanych programów przedstawia się na podobnym poziomie, jednak w każdym z pomiarów język Swift wykazuje się mniejszym zapotrzebowaniem na moc obliczeniową procesora.



Rysunek 1: Obciążenie procesora wyrażone w %.

Na Rysunku 2 przedstawione zostało średnie zużycie pamięci RAM wyrażone w megabajtach. Swift również pod tym względem cechuje się mniejszym zapotrzebowaniem na zasoby sprzętowe. Różnica w wykorzystaniu pamięci między zastosowanymi językami jednak nie jest tak znacząca jak w przypadku zużycia procesora, a dla największego zestawu danych jest prawie niezauważalna. Wraz ze wzrostem ilości danych wpływ zastosowanego języka pod względem zużycia zasobów pamięci operacyjnej maleje.



Rysunek 2: Wykorzystanie pamięci RAM wyrażone w MB.

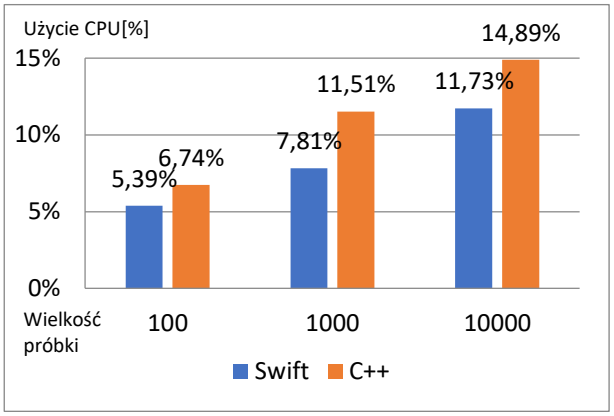
W Tabeli 2 przedstawione zostały czasy wykonania programu wyrażone w mikrosekundach. W przypadku czasów wykonania język Swift potrzebował znacząco więcej czasu w porównaniu do języka C++. Na podstawie tabeli można zauważyć, że wraz ze wzrostem liczby próbek różnica w czasie pomiędzy badanymi językami zmniejsza się.

Tabela 2: Czas wykonania programu wyrażony w mikrosekundach

Liczba próbek	Swift[µs]	C++[µs]
100	251,43	2,59
1000	672,86	22,74
10000	4885,71	234,12

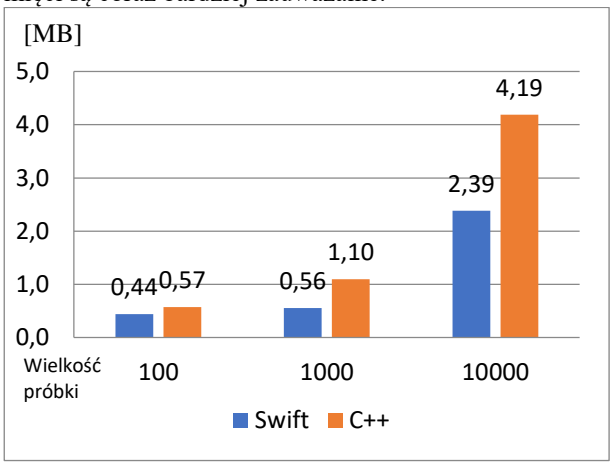
6.2. Seria pomiarowa 2

Drugi z badanych programów służył do obliczenia mediany i wykorzystywał zestawy danych, które użyte były przy pierwszym programie. Użycie mocy obliczeniowej procesora podczas działania drugiego programu przedstawione zostało na Rysunku 3. Różnice w wykorzystaniu zasobów procesora dla tej serii pomiarowej dla każdego z przypadków są mniejsze niż dla serii poprzednich pomiarów. Dla pierwszego zestawu danych średnia różnica w wykorzystaniu procesora nie przekracza jednego procenta, natomiast dla dwóch pozostałych badanych zestawów różnica ta jest nieznacznie większa, jednak nie przekracza ona czterech procent. Należy zauważyć, że w przypadku drugiej serii pomiarowej użycie procesora dla obu badanych programów przedstawia się na podobnym poziomie, jednak język Swift potrzebuje mniej zasobów sprzętowych.



Rysunek 3: Obciążenie procesora wyrażone w %.

Na Rysunku 4 przedstawione zostało średnie zużycie pamięci RAM wyrażone w megabajtach. Dla najmniejszego zestawu danych różnica w wykorzystaniu pamięci jest wręcz pomijalna, jednak dla pozostałych przypadków jest niemal dwukrotnie większa na korzyść języka Swift. Zauważyć należy tendencję odwrotną niż w przypadku pierwszej serii pomiarowej, gdyż wraz ze wzrostem ilości danych, różnice w wykorzystaniu pamięci są coraz bardziej zauważalne.



Rysunek 4: Wykorzystanie pamięci RAM wyrażone w MB.

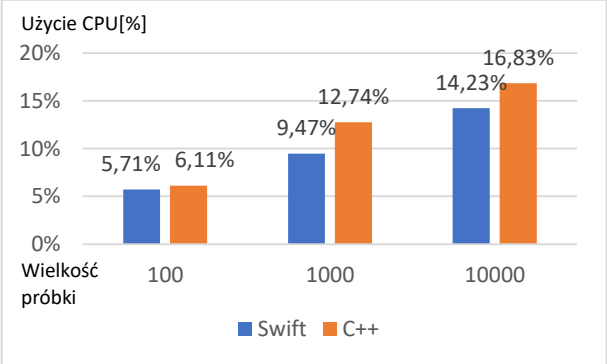
W Tabeli 3 przedstawione zostały czasy wykonania programu wyrażone w mikrosekundach. W przypadku języka Swift czas wykonania programu rośnie liniowo, natomiast dla języka C++ różnica w czasie dla zestawu 100 i 1000 wartości wynosi około dwa i pół, a dla próbek 1000 i 10000 wynosi ponad sześć.

Tabela 3: Czas wykonania programu w mikrosekundach

Liczba próbek	Swift[µs]	C++[µs]
100	592,86	20,25
1000	2872,86	48,98
10000	18300	303,46

6.3. Seria pomiarowa 3

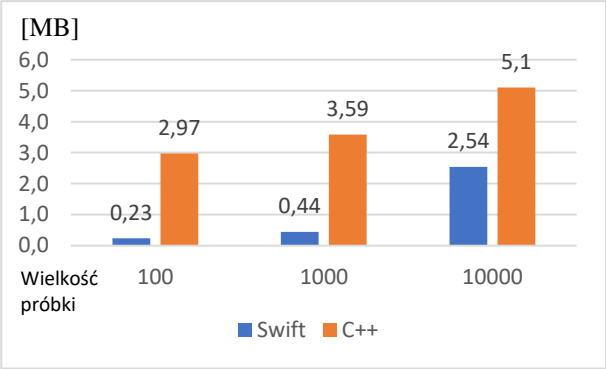
Program liczący średnie dla najmniejszej próbki liczącej 100 danych losowych potrzebuje prawie takiej samej mocy obliczeniowej procesora dla programów zarówno napisanych w języku Swift jak i C++, różnica wynosi zaledwie 0,4%. Dla dwóch większych zestawów danych różnice te wynoszą 3,27% oraz 2,6% odpowiednio dla próbek liczących 1000 i 10000 danych. Również w tym programie język Swift potrzebował mniejszej mocy obliczeniowej jednak różnica ta była o wiele mniejsza niż w przypadku dwóch poprzednich programów, mogła na to wpłynąć liczba obliczeń jakie były do wykonania, Program ten miał w sobie trzy funkcje, po jednej dla każdej z liczonych średnich i każda z nich zawierała pętlę, która musiała wykonać pewne działania arytmetyczne na wszystkich elementach tablicy. Na Rysunku 5 przedstawiono procentowe użycie procesora.



Rysunek 5: Obciążenie procesora wyrażone w %.

Podczas wykonywania programu liczącego średnie język C++ potrzebował znacznie większej ilości pamięci operacyjnej. Dal najmniejszej próbki liczącej zaledwie 100 danych język Swift potrzebował 0,23MB pamięci natomiast język C++ aż 2,97MB, różnica ta jest prawie trzynastokrotna. Druga próbka licząca 1000 obciążyła program na tyle, że potrzebował on średnio 3,59MB pamięci RAM w przypadku języka C++ 0,44MB w przypadku języka Swift, różnica nie jest aż tak duża jak dla pierwszej próbki, jednakże wciąż jest to ponad ośmiokrotność zapotrzebowania przez język Swift. Dla największej próbki zapotrzebowanie na pamięć operacyjną wynosiło 2,54MB w przypadku języka Swift oraz 5,1MB w przypadku języka C++, co oznacza, że C++ potrzebuje ponad dwa razy więcej pamięci do wykonania obliczeń. Tak rozbieżne wyniki dal obu języków mogą być spowodowane ilością obliczeń jakie

wykonuje program do policzenia trzech średnich. Na Rysunku 6 przedstawione zostały średnie użycie pamięci operacyjnej.



Rysunek 6: Wykorzystanie pamięci RAM wyrażone w MB.

W Tabeli 4 przedstawiono czasy wykonania programu w mikrosekundach. W przypadku pierwszej próbki liczącej zaledwie 100 danych różnica jest ponad stuosiemdziesięciokrotna. Program napisany w języku Swift potrzebuje średnio 281,19µs na wykonanie, natomiast program napisany w języku C++ potrzebuje zaledwie 1,51µs. Dla drugiej próbki liczącej 1000 danych różnica ta zmniejsza się i wynosi niewiele ponad 80 razy więcej, co i tak jest bardzo dużą różnicą, czasy jakich potrzebują programy do wykonania się przy tej próbce to 900,80µs w przypadku języka Swift oraz 11,18µs w przypadku języka C++. Największa próbka licząca 10000 danych obciążyła program w dużym stopniu, język Swift potrzebuje średnio 7368,57µs na wykonanie programu, a języka C++ 115,96µs co stanowi wartość ponad 60 razy mniejszą.

Tabela 4: Czas wykonania programu w mikrosekundach

Liczba próbek	Swift[µs]	C++[µs]
100	281,19	1,51
1000	900,80	11,18
10000	7368,57	115,96

7. Wnioski

W artykule dokonano analizy wydajności języka Swift w porównaniu do języka C++ podczas wykonywania obliczeń statystycznych. Na podstawie przeprowadzonych badań i zestawieniu wyników można zauważyć, iż programy napisane w języku Swift charakteryzują się niższym zapotrzebowaniem na moc obliczeniową, gdyż w przypadku obu badanych programów zużycie procesora i wykorzystanie pamięci operacyjnej było niższe.

Przy pierwszej serii pomiarowej badany program liczył odchylenie standardowe. Dla mniejszych zestawów danych jest różnica na korzyść języka Swift, a dla większych wykorzystanie procesora w przypadku dwóch języków było prawie takie samo. Wykorzystanie pamięci dla obu języków przedstawia się na zbliżonym poziomie, a dla największego z zestawów danych różnica w zapotrzebowaniu jest niemal pomijalna. Aplikacja napisana w języku Swift potrzebuje znacznie więcej czasu na wykonanie obliczeń, a zwiększenie rozmiaru danych, na których wykonywane są obliczenia zmniejsza różnicę w czasie wykonania.

Dla drugiej serii pomiarowej program obliczał medianę zestawu danych. Jest to zadanie bardziej skomplikowane niż w przypadku pierwszej serii, gdyż wszystkie liczby muszą zostać posortowane, co wprowadza konieczność wykonania większej liczby obliczeń. Wybór języka nie wpływa w znacznym stopniu na obciążenie procesora, jednak dla każdego zbioru liczb zauważyć należy przewagę języka Swift. Różnica w zużyciu pamięci operacyjnej nie wydaje się zależeć od rozmiaru zestawu danych, natomiast ponownie język Swift charakteryzuje się mniejszym wykorzystaniem zasobów urządzenia.

Trzecia seria pomiarowa przeprowadzona została dla programu liczącego zestaw trzech średnich. Jest to program o największej złożoności, składający się z trzech, niezależnie działających funkcji i zwracający wyniki trzech obliczeń. Procesor bez względu na zastosowany język był obciążony w bardzo podobnym stopniu, przy czym program napisany w języku Swift charakteryzował się nieznacznie mniejszym zapotrzebowaniem. Dużo większe różnice zaobserwować można pod kątem wykorzystanej pamięci operacyjnej. Dla dwóch mniejszych zestawów użycie pamięci było około 10 razy niższe, a dla największego z zestawów program napisany w języku Swift potrzebował około dwukrotnie mniej pamięci. Tak jak w przypadku dwóch poprzednich programów czas wykonania dla języka Swift był znacząco większy, jednak różnica w ilości danych wejściowych nie jest wprost proporcjonalna do czasu wykonania programu.

Programy napisane w języku Swift charakteryzują się mniejszymi wymaganiami sprzętowymi ze względu na ukierunkowanie na działanie pod kontrolą jednego systemu operacyjnego, który oficjalnie wspierany jest w znakomitej większości na urządzeniach firmy Apple. Jednak wraz ze wzrostem poziomu skomplikowania projektu przewaga polegającą na mniejszym zużyciu zasobów sprzętowych języka Swift zaczyna się zacierać, ponieważ wykorzystana moc obliczeniowa procesora, bądź zużycie pamięci zaczyna wzrastać do poziomów zbliżonych podczas działania programów napisanych w języku C++. Czas wykonania aplikacji jednoznacznie pokazuje, że język C++ jest wielokrotnie szybszy, a autorom nie udało się wskazać przewagi języka Swift na tym polu.

Ze względu na fakt, iż Swift jest dość młodym językiem istnieje możliwość, iż będzie on stale udoskonalony w kolejnych wersjach. C++ jest łatwiejszy, jeśli chodzi o poziom trudności podczas nauki oraz bardziej dostępny ze względu na to, że jest szeroko stosowany na najpopularniejszych systemach operacyjnych obsługiwanych przez urządzenia desktopowe. Języka Swift również można używać na większości urządzeń natomiast nie wszystkie są tak dobrze przystosowane do tego jak urządzenia firmy Apple. Ponadto duża dostępność bibliotek, gotowych programów, wpisów na forach sprawia, że jest on dobrym wyborem dla początkujących programistów. Jednak niewykluczone, że

w przyszłości rozwój języka Swift i zapotrzebowanie na aplikacje dedykowane konkretnemu środowisku będą na tyle duże, że popularność tego języka wzrośnie. Podczas wyboru języka do obliczeń statystycznych należy wziąć pod uwagę dostępny sprzęt oraz potrzeby użytkownika, dla słabszych urządzeń lepszym wyborem okazuje się język Swift, natomiast dysponując bardziej wydajną konfiguracją sprzętową wybór ten może być zależny wyłącznie od osobistych preferencji programisty.

Literatura

[1] Oficjalna dokumentacja języka Swift, https://www.swift.org/documentation, [15.06.2022].

[2] Oficjalna dokumentacja języka C++, https://docs.microsoft.com/pl-pl/cpp/cpp/?view=msvc-170, [15.06.2022].

[3] J. Hoffman, Swift 4. Koduj jak mistrz, Helion, Gliwice, 2018.

[4] S. Prata, Język programowania C++. Szkoła programowania, Helion, Gliwice, 2012.

[5] R. Baida, M. Andriienko, M. Plechawska-Wójcik, Analiza porównawcza wydajności frameworków Angular oraz Vue.js, Journal of Computer Sciences Institute 14 (2020) 59-64, https://doi.org/10.35784/jcsi.1577.

[6] S. Bose, A Comparative Study: Java vs Kotlin Programming in Android Application Development, International Journal of Advanced Research in Computer Science 9(3) (2018) 41-45.

[7] S. Khoirom, S. Moirangthem, B. Laikhuram, J. Laishram, T. D. Singh, Comparative Analysis of Python and Java for Beginners, Int. Res. J. Eng. Technol 7(8) (2020) 4384-4407.

[8] H. Singh, Speed Performance Between Swift and Objective-C, Int. J. Eng. Appl. Sci. Technol 1(10) (2016) 185-189.

[9] D. Sulowski, G. Kozieł, Comparative Analysis of Kotlin and Java languages Used to Create Applications for the Android System, Journal of Computer Sciences Institute 13 (2019) 354-358, https://doi.org/10.35784/jcsi.1332.