Analiza szybkości wielowątkowego wykonywania programów

Gabriel Łapieś i Marcel Kańduła

1. Opis zadania

W ćwiczeniach przetwarzano zbiór liczb całkowitych losowych z przedziału [0,1000] o rozmiarze n = 1000 (będącym parametrem) za pomocą dwóch podejść:

- Przetwarzanie sekwencyjne - Jednowątkowe wykonanie obliczeń na całym zbiorze danych.

- Przetwarzanie równoległe (wielowątkowe) - Podział zadań między kilka wątków (2-16), które wspólnie pobierają dane ze wspólnej kolejki i wykonują obliczenia.

Oba podejścia mierzyły czas przetwarzania. Wyniki (w postaci obiektów klasy Result) zbierane były przez dedykowany obiekt ResultsCollector.

2. Implementacja

Data:
Odpowiada za generowanie i przechowywanie zbioru liczb wejściowych w kolejce. Metody getNext() oraz isEmpty() umożliwiają bezpieczny pobór elementów w środowisku wielowątkowym.
Calculations:
Implementuje logikę obliczeniową. W metodzie run() każdy wątek pobiera dane z kolejki, wykonuje obliczenia (metoda computation) i przekazuje wynik do ResultsCollector.
Result:
Pakuje wynik obliczenia, zawierając dane wejściowe, wynik oraz czas przetwarzania.
ResultsCollector:
Przechowuje wyniki obliczeń jako lista obiektów klasy Result
Klasa Main:
Odpowiada za uruchomienie przetwarzania. Na początku wykonywane są obliczenia równoległe przy użyciu określonej liczby wątków (przekazywanej jako parametr). Następnie wykonywane są obliczenia sekwencyjne. W obu przypadkach mierzony jest czas wykonywania, co pozwala na porównanie obu podejść.

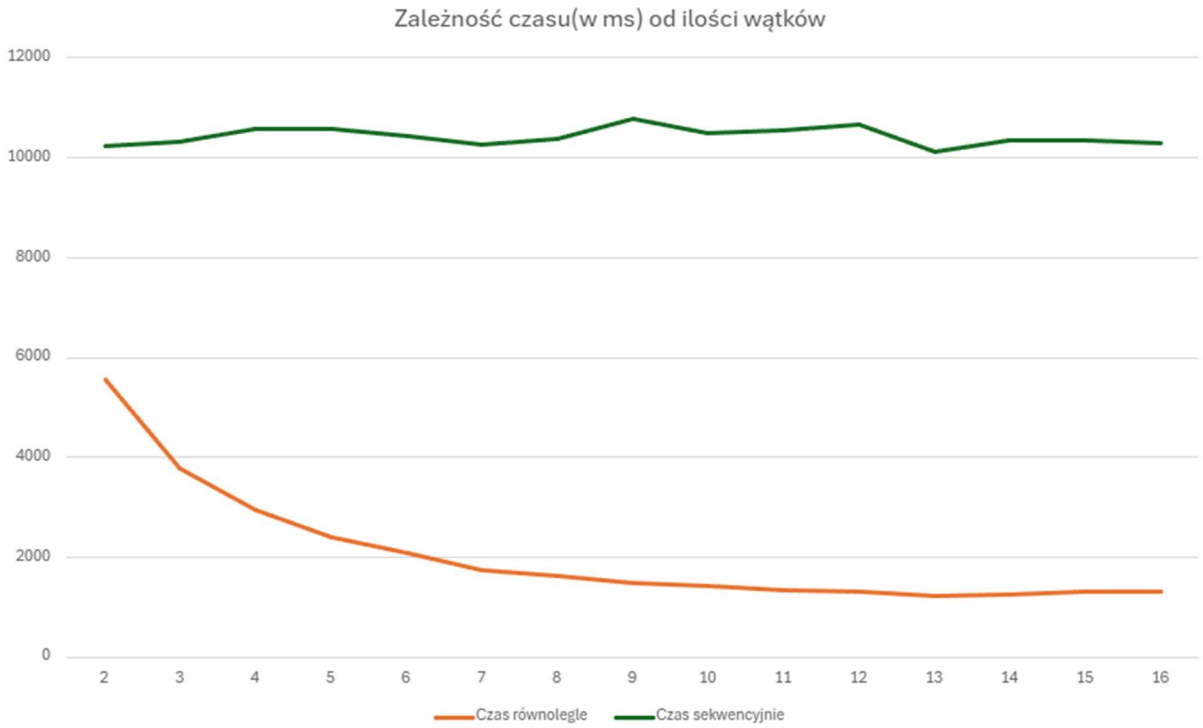
Nasze obliczenia mają charatker *stricte* teoretyczny i służą wydłużeniu czasu przetwarzania pojedynczego wyniku.

Dla każdej liczby liczymy jej wynik za pomocą następującej metody computation:

```
private int computation(int input){
    int result = 1;

    int iterations = 1400 * input + 13;
    for (int i = 1; i<= iterations; i++){
        result = (result * i + input) % 2147483647;
        result = result - (int)(result/i - Math.sqrt(result));
    }
    return result;
}
```

3. Pomiary



Wykres przedstawia czas wykonywania programu (w mili sekundach) równolegle (pomarańczowa linia) oraz sekwencyjnie (zielona linia) na procesorze o 6 rdzeniach i 12 wątkach.

Pomiary czasu wykonywania sekwencyjnego są zależne od danych wejściowych, ich czasy nie odbiegają znacznie od siebie. Czas wykonywania równoległego jest zależny od danych wejściowych jak i liczby wątków. Dodanie wątków, gdy jest ich mniej niż 6 znacznie skraca czas wykonania programu, dodanie kolejnych wątków przynosi coraz

mniejsze korzyści. Gdy liczba wątków wynosi 14 dodanie następnych nieznacznie spowalnia program. Dzieje się tak, ponieważ procesor przekracza optymalną liczbę wątków dla danego programu, przez co przełączanie kontekstu między wątkami staje się bardziej kosztowne niż posiadanie mniejszej ilości wątków.

Na podstawie pomiarów czasu przetwarzania stwierdziliśmy, że

- Czas **przetwarzania sekwencyjnego** zależy głównie od danych wejściowych - poszczególne pomiary nie wykazują dużych różnic.
- Czas **przetwarzania równoległego** zależy zarówno od danych wejściowych, jak i od liczby wątków:
    * Przy mniejszej liczbie wątków dodanie kolejnych wątków znacząco skraca czas wykonania.
    * Po osiągnięciu optymalnej liczby kolejne wątki przynoszą mniejsze korzyści ze względu na narzut związany z przełączaniem kontekstu.
    * Gdy liczba wątków przekroczy liczbę wątków logicznych, wzrost liczby wątków może nawet nieznacznie spowalniać program.

4. Wnioski

1. **Wielowątkowość** - przy odpowiednim doborze liczby wątków (dopasowanej do możliwości sprzętowych) znacznie poprawia wydajność programu.

2. **Sekwencyjne przetwarzanie** jest bardziej przewidywalne (liniowy czas przetwarzania, zależny głównie od rozmiaru danych), ale przy większych zbiorach danych okazuje się znacznie wolniejsze.

3. W praktycznych zastosowaniach należy uwzględniać charakterystykę sprzętu, gdyż nadmiar wątków może przynieść efekt przeciwny do zamierzonego.

Analiza obu podejść pozwala nie tylko dostrzec zalety równoległego przetwarzania (np. skrócenie czasu wykonania obliczeń), ale także ujawnia ograniczenia skalowalności - zwiększenie liczby wątków ponad optymalny poziom niekoniecznie przekłada się na lepszą wydajność.