Implementacja oraz porównanie architektur SISD oraz SIMD

Piotr Szczypior

1 Wstęp

Professor Michael Flynn w 1966 roku zaproponował podział architekrury komputera ze względu na przetwarzanie danych na 4 rodzaje: `SISD`, `SIMD`, `MISD` oraz `MIMD`.

Pomimo istnienia modeli teoretycznych przez wiele lat komputery, które przetwarzały dane równolegle, były obecne tylko w 'superkomputerach'. Dopiero w 1996 roku powstał pierwszy produkowany na masową skale procesor `Pentium MMX` zaprojektowany przez firmę *Intel*, który używał przetwarzania równoległego. Rozwuj tego procesora zapoczątkował nową erę, w tym jak ludzie używali komputera. W późniejszych latach powstawały coraz lepsze rozwiązania o architekturze SIMD. Obecnie powrzechnie używanym jest `AVX2` wspierany zarówno procesory *AMD*, jak i *Intel*. Architektura SIMD jest powszechnie używana w kompresji, grafice komputerowej, przetwarzaniu sygnałów i wielu innych dziedzinach, gdzie dane przetwarzane są równolegle.

2 Implementacja

2.1 SIMD

W celu zaimplementowania kodu w assemblerze, który wykonuje operację opartę o architekturę SIMD, należy użyć rejestrów 128-bitowych XMM (dla wektorów 128-bit). Poniższy fragment kodu przedtawia wstawkę assemblerową, która pozwala na wykonanie dodawania na wspomnianych wyżej wektorach 128 bitowych wykorzystując XMM.

```
__asm__ (
       "movups %1, %%xmm0\n\t"
       "movups %2, %%xmm1\n\t"
       "addps %%xmm0, %%xmm1\n\t"
       "movups %%xmm1, %0\n\t"
       : "=m" (result)
       : "m" (first_vector), "m" (second_vector)
   );
```

Podobne wstawki w programie powstały w implementacji klasy `CalculatorSIMD` dla odejmowania, mnożenia oraz dzielenia.

2.2 SISD

W przypadku SISD, został zaimplementowana podobna klasa jak w przypadku SIMD - `CalculatorSISD`. Posiada ona również implementację podstawowych działań arytmetycznych.

```
__asm__ (
       "movups %1, %%xmm0\n\t"
       "movups %2, %%xmm1\n\t"
       "addps %%xmm0, %%xmm1\n\t"
       "movups %%xmm1, %0\n\t"
       : "=m" (result)
       : "m" (first_vector), "m" (second_vector)
   );
```

2.3 Pomiar czasu

Czas działania operacji na liczbach w programie został zmierzony za pomocą biblioteki `chrono`. W tym celu powstała klasa, która udostępnia metody potrzebnę do obliczania czasu wykonywania się operacji.

```
class Timer {
private:
    high_resolution_clock::time_point endTime;
    high_resolution_clock::time_point initialTime;
public:
    void timeStart();
    void timeStop();
    long elapsedTime();
};
```

W celu jak najdokładniejszych pomiarów czasu timer liczy tylko wykowanie się wstawki assemblerowej.

3 Wynik pomiarów

W celu stworzenia wykresów został stworzony skryp w pythonie `plot.py`. Wykresy zostały swtorzone wykorzystując paczki *pyplotlib* oraz *pandas*.

W poniższej tabeli znajdują się pomiary dla SISD.

Tabela 1. Pomiary czasu wykonanych operacji dla SISD

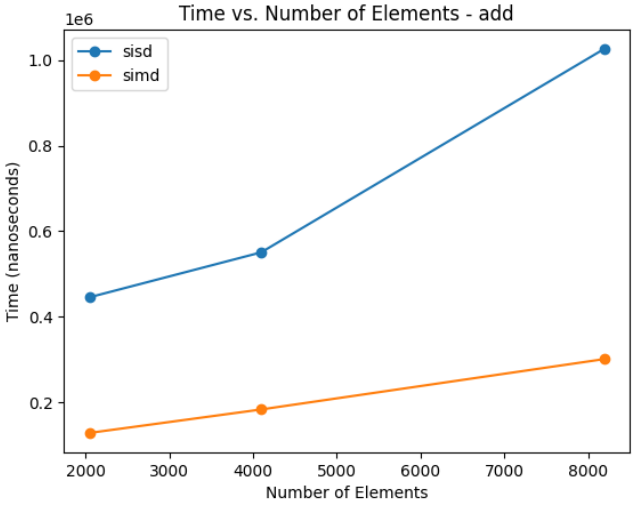
liczba elementów $[n]$	dodawanie $[\mu s]$	odejmowanie $[\mu s]$	mnożenie $[\mu s]$	dzielenie $[\mu s]$
2048	445	443	374	425
4096	550	536	555	611
8192	1025	9891	1002	1015

Natomiast poniżej została przedstawiona tabela pomiarów dla SIMD.

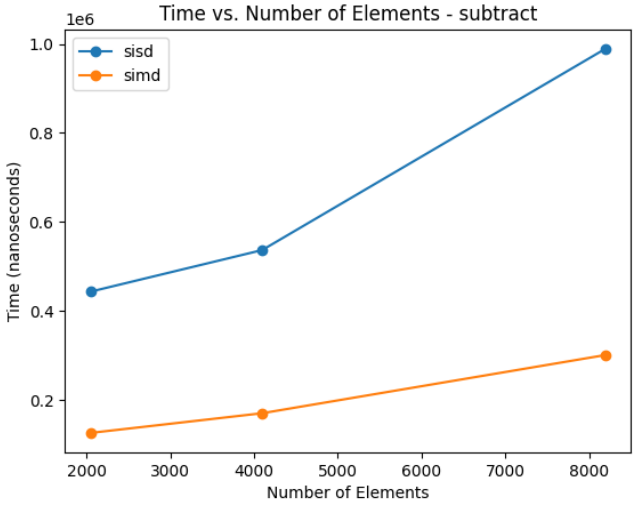
Tabela 2. Pomiary czasu wykonanych operacji dla SIMD

liczba elementów $[n]$	dodawanie $[\mu s]$	odejmowanie $[\mu s]$	mnożenie $[\mu s]$	dzielenie $[\mu s]$
2048	129	125	132	137
4096	184	170	187	192
8192	301	300	297	328

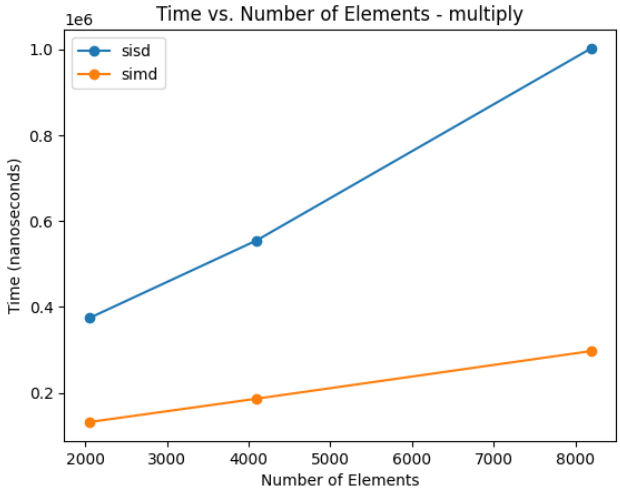
Na podstawie powyższych tabel zostały stworzone wykresy.



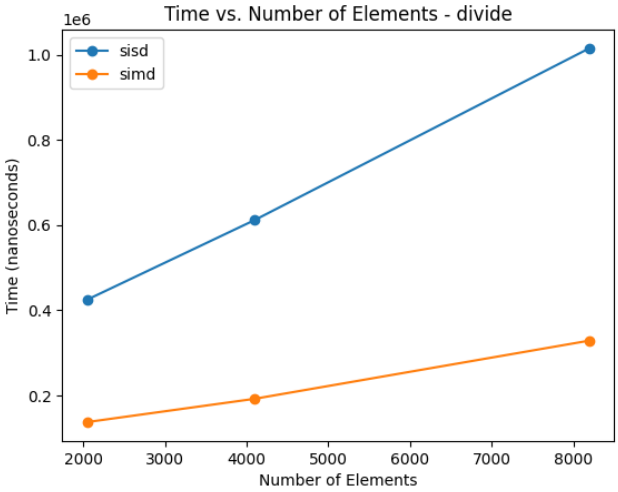
Rysunek 1. Wykres czasu operacji dodawania dla obu rodzajów architektury



Rysunek 2. Wykres czasu operacji odejmowania dla obu rodzajów architektury



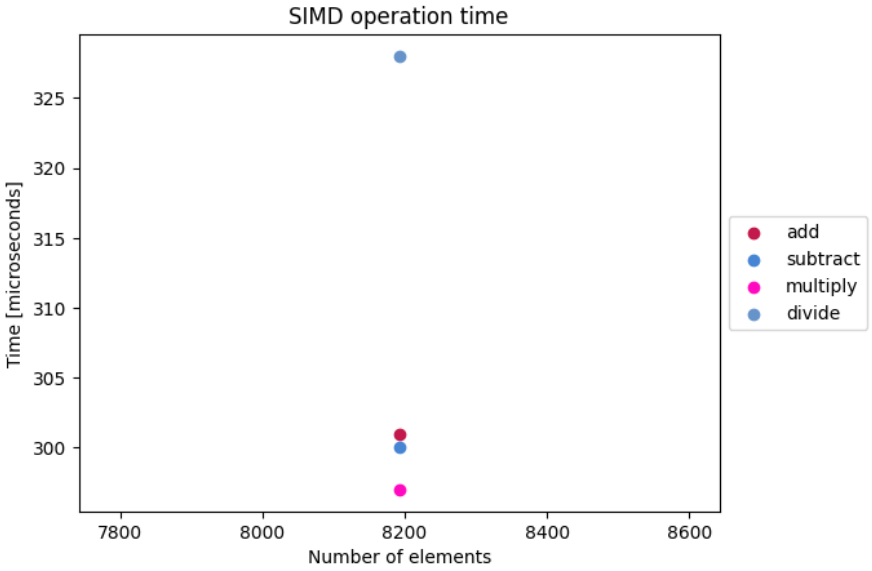
Rysunek 3. Wykres czasu operacji mnożenia dla obu rodzajów architektury



Rysunek 4. Wykres czasu operacji dzielenia dla obu rodzajów architektury

Na podstawie danych z tabeli oraz wykresów można zauważyć, że SIMD jest zdecydowanie skuteczniejsze oraz że zysk z używania operacji SIMD jest około 4 krotny zatem czas operacji na architekturze SIMD będzię trwał około 25% czasu operacji SISD.

Dodatkowo na poniższym wykresie został umieszczony czas operacji dla architekrury SIMD ze względu na rodzaj operacji.



Rysunek 5. Wykres przedstawiający czas wykonania operacji dla SIMD

Na podstawie powyższego wykresu można zauważyć, że najwięcej czasu zajęła operacja dzielenia.

4 Wnioski

Na podstawie przeprowadzonych badań można wywnioskować, że architektura SIMD jest szczególnie skuteczna w przypadku zadań, które wymagają równoległego przetwarzania dużej ilości danych. Dzięki możliwości przetwarzania wielu danych jednocześnie, SIMD może znacznie skrócić czas wykonania operacji w porównaniu do SISD. Zysk z używania tej architektury wynosi około 25% względem czasu potrzebnego dla tych samych procesów w architekturze SISD, jednakże do dokładniejszego oszacowania zysku powinno się przeprowadzić więcej badań benchmarkowych.

Wyniki pomiarów czasu operacji na SIMD i SISD pozwalają również na identyfikację obszarów, które mogą być zoptymalizowane. Analiza wyników może ujawnić, że pewne operacje są bardziej czasochłonne na SIMD lub SISD.

Wyniki badań mogą różnić się w zależności od konkretnych implementacji SIMD i SISD oraz używanych procesorów. Dlatego ważne jest, aby przeprowadzić badania benchmarkowe na konkretnych systemach i architekturach, aby uzyskać dokładniejsze wnioski.

Podsumowując, badania pomiaru czasu operacji na SIMD i SISD pozwalają na ocenę wydajności tych architektur, identyfikację optymalizacji oraz wykazują, że SIMD jest wydajniejszy od SISD, zwłaszcza w przypadku równoległego przetwarzania dużej ilości danych. Jednak dokładny zysk wynikający z użycia SIMD może być różny w zależności od konkretnych warunków testowych.