Uniwersytet Bielsko-Bialski

Wydział budowy maszyn i informatyki

Architektura komputerów

(Laboratorium №4)

Temat ćwiczenia: Praca potokowa procesorów

Data wykonania ćwiczenia: 20.12.2023

Igor Gawłowicz

1. Wprowadzenie Praca potokowa procesorów to technika zwiększania wydajności procesorów poprzez równoczesne wykonanie wielu instrukcji w różnych fazach procesu wykonawczego. Procesory wykonują wiele operacji na danych, takie jak pobieranie danych z pamięci, dekodowanie instrukcji, wykonanie operacji arytmetycznych itp. Praca potokowa polega na podzieleniu tych operacji na kilka etapów i równoczesnym przetwarzaniu różnych instrukcji w różnych etapach.

Główne etapy potoku w procesorze to:

- Pobieranie instrukcji: W tym etapie procesor pobiera instrukcje z pamięci programu.
- Dekodowanie instrukcji: Procesor dekoduje pobrane instrukcje, rozumiejąc jakie operacje należy wykonać.
- Wykonywanie operacji: Procesor wykonuje właściwe operacje, takie jak operacje arytmetyczne, logiczne itp.
- Dostęp do pamięci: Jeśli instrukcje wymagają dostępu do pamięci, procesor odczytuje lub zapisuje dane w pamięci.
- Zapis wyników: Wyniki operacji są zapisywane w odpowiednich rejestrach lub pamięci.

Dzięki pracy potokowej, różne instrukcje mogą być równocześnie w różnych fazach procesu wykonawczego, co pozwala na zwiększenie wydajności procesora. Praca potokowa umożliwia również zmniejszenie czasu wykonywania pojedynczej instrukcji, co przyspiesza ogólny czas wykonywania programu. Niemniej jednak praca potokowa może napotykać na pewne wyzwania, takie jak zależności między instrukcjami, które mogą powodować opóźnienia, oraz zagadnienia związane z równoważeniem obciążenia potoku, aby zapewnić równomierne wykorzystanie wszystkich etapów.

2. Zadanie do zrealizowania: 2.1. Przeanalizuj poniższy kod pod kątem wykorzystania potoków. Zaproponuj modyfikacje kodu w celu lepszego wykorzystania potoków poprzez rozpisanie najbardziej wewnętrznej części pętli for:

```
// Wersja v1 programu z jednym buforem „s1”
#include <iostream>
#include <iomanip>
#include <chrono>

int main()
{
    const int n = 1024; // rozmiar tablicy
    double* tab = new double[n]; // tworzenie obietu w pamięci dla tablicy
dynamicznej
    for (int i = 0; i < n; ++i) tab[i] = 0.0625; //wypełnianie tablicy
    std::chrono::steady_clock::time_point begin =
std::chrono::steady_clock::now();
    double s1 = 0.0; //bufor dla sumy wszystkich elementów tablicy
    //poniższy fragment należy zmienić
    for (int t = 0; t < n * n; ++t)
    {
        for (int i = 0; i < n; ++i)
        {
            s1 += tab[i];
        }
```

```
    }
    std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
    std::cout << std::fixed;
    std::cout << "1. suma = " << std::setprecision(30) << s1 << " ";
    std::cout << "Czas wykonania = " <<
std::chrono::duration_cast<std::chrono::microseconds>(end - begin).count() << "
[µs]" << std::endl;
    delete[] tab; //kasowanie tablicy
    return 0;
}
```

Początkowo program zwraca nam taki wynik

```
suma = 67108864.000000000000000000000000000000 Czas wykonania = 4727504[µs]
```

z użyciem 2 buforów dla tymczasowych wyników pośrednich s1 i s2;

```
#include <iostream>
#include <iomanip>
#include <chrono>

int main()
{
    const int n = 1024; // rozmiar tablicy
    double* tab = new double[n]; // tworzenie obietu w pamięci dla tablicy
dynamicznej
    for (int i = 0; i < n; ++i) tab[i] = 0.0625; //wypełnianie tablicy
    std::chrono::steady_clock::time_point begin =
std::chrono::steady_clock::now();
    double s1 = 0.0; // bufor pierwszy
    double s2 = 0.0; // bufor drugi

    for (int t = 0; t < n * n; ++t)
    {
        for (int i = 0; i < n; ++i)
        {
            if (t % 2 == 0) {
                s1 += tab[i];
            } else {
                s2 += tab[i];
            }
        }
    }

    // Sumowanie wyniku końcowego z obu buforów
    double final_sum = s1 + s2;

    std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
    std::cout << std::fixed;
```

```
    std::cout << "s1 = " << s1;
    std::cout << "s2 = " << s2 << std::endl;
    std::cout << "1. suma = " << std::setprecision(30) << final_sum << " ";
    std::cout << "Czas wykonania = " <<
std::chrono::duration_cast<std::chrono::microseconds>(end - begin).count() << "
[µs]" << std::endl;
    delete[] tab; //kasowanie tablicy
    return 0;
}
```

Wynik

```
s1 = 33554432.000000s2 = 33554432.000000
1. suma = 67108864.000000000000000000000000000000 Czas wykonania = 4993724[µs]
```

Zasada przy następnych będzie taka sama więc przejdę od razu do tabeli z wynikami

```
s1 = 16777216.000000
s2 = 16777216.000000
s3 = 16777216.000000
s4 = 16777216.000000
1. suma = 67108864.000000000000000000000000000000 Czas wykonania = 5235796[µs]
```

```
s1 = 8388608.000000
s2 = 8388608.000000
s3 = 8388608.000000
s4 = 8388608.000000
s5 = 8388608.000000
s6 = 8388608.000000
s7 = 8388608.000000
s8 = 8388608.000000
1. suma = 67108864.000000000000000000000000000000 Czas wykonania = 4508928[µs]
```

```
s1 = 4194304.000000
s2 = 4194304.000000
s3 = 4194304.000000
s4 = 4194304.000000
s5 = 4194304.000000
s6 = 4194304.000000
s7 = 4194304.000000
s8 = 4194304.000000
s9 = 4194304.000000
s10 = 4194304.000000
s11 = 4194304.000000
s12 = 4194304.000000
```

```
s13 = 4194304.000000
s14 = 4194304.000000
s15 = 4194304.000000
s16 = 4194304.000000
1. suma = 67108864.000000000000000000000000000000 Czas wykonania = 5315598[µs]
```

Wersja kodu	Liczba buforów	Czas programu
v1	1	4.727
v2	2	4.993
v3	4	5.235
v4	8	4.508
v5	16	5.315

Najkrótszy czas wykonania (4.508 [µs]) występuje dla wersji kodu z 8 buforami (v4).
Czas wykonania dla 2 i 16 buforów jest zbliżony, ale nieznacznie dłuższy niż dla 8 buforów.
Wersje z 1 i 4 buforami mają podobny czas wykonania, choć dla 4 buforów jest nieco dłuższy niż dla 1 bufora.

2.2

Wersja kodu	Liczba buforów	Suma wszystkich elementów
v1	1	107374180.70588229596614837646484375000
v2	2	107374183.41176469624042510986328125000
v3	4	107374182.82352940738201141357421875000
v4	8	107374182.14705882966518402099609375000
v5	16	107374182.29411767423152923583984375000

Wartości sumy są bliskie wartości 107374182.5, która jest sumą łączną dla 1024 elementów o wartości 0.1.
Różnice w sumach są wynikiem błędów arytmetyki zmiennoprzecinkowej, co jest naturalne w tego typu operacjach.

Wnioski

Czas wykonania programu zależy od liczby buforów, ale wyniki nie pokazują jednoznacznej zależności: największa liczba buforów (v5) nie zawsze oznacza najdłuższy czas wykonania.

Różnice w czasie wykonania są stosunkowo niewielkie między różnymi wersjami kodu, co może wynikać z niewielkich rozmiarów danych i prostej logiki operacji w pętli.

Wartości sumy są zbliżone do oczekiwanej sumy, ale różnice mogą występować z powodu błędów reprezentacji liczb zmiennoprzecinkowych.

Skuteczność buforów może zależeć od wielu aspektów, gdzie głównym czynnikiem będzie urządzenie z którego korzystamy, największy wpływ ma architektura procesora ponieważ 8 rdzeniowy procesor o wiele

lepiej poradzi sobie z podziałem pracy 8 buforów od 16 buforów.

Kolejnym aspektem będzie pamięć RAM, nie powinna mieć ona wpływu na zależność ilości buforów od czasu ale zdecydowanie będzie miała ona wpływ na szybkość działania programu.