Przetwarzanie współbieżne

Laboratorium 6

Jan Bartula

# Przygotowanie do zadania

Aby móc przystąpić do wykonywania zadań zostały przygotowane klasy pomocnicze:

* CSVFile
* Matrix

Pierwsza umożliwia łatwiejsze wypisywanie do plików .csv w celu zaimportowania wyników do EXCEL, który umożliwi prezentację wyników w formie wykresów

Druga obejmująca tworzenie macierzy o określonej wielkości, wraz z generowaniem jej wartości.

## CSVFile

Zawiera podstawowe funkcje obsługi strumieni, domyślny znak ograniczający kolumny, a także zakańczanie wierzy. Dodatkowo możliwość „spłukania” strumienia pisania do pliku.

Taka klasa umożliwia proste pisanie do plików CSV

CSVFile csv("MatrixTiming.csv");

csv << "test1" << "test2" << "test3" << nextl;

csv << 22 << 22 << 22 << nextl;

Czego wynikiem jest plik:

>MatrixTiming.csv

"test1","test2","test3"

22,22,22

Nie musimy dbać o znaki ograniczające, a także o wrapowanie tekstu w cudzysłowie.

## Matrix

Jest to prosta klasa zawierająca wskaźnik na dane, wymiary macierzy i inicjalizator.  
Dodatkowo została napisana funkcja losująca liczby korzystająca z równomiernego rozkładu (na forach internetowych istnieje opinia że jest to najlepszy z tych krótkich generatorów liczb)

static double random\_number(double min = -99, double max = 99) {  
 std::random\_device rd{};  
 std::mt19937 gen{rd()};  
 std::uniform\_real\_distribution<double> distribution(min, max);  
 return distribution(gen);  
}

## Zadanie

W ramach zadania przygotowano 8 funkcji, która wykonuje mnożenie dla metod zrównoleglenia:

Program działa wedle wzoru na mnożenie macierzy:

Matrix multiply\_nnn(const Matrix &A, const Matrix &B) {  
 double sum;  
 if (A.M != B.N) {  
 throw std::invalid\_argument("Size is not equal");  
 }  
  
 Matrix C(A.N, B.M);  
 for (int y = 0; y < A.N; y++) {  
 for (int x = 0; x < B.M; x++) {  
 sum = 0;  
 for (int i = 0; i < A.M; i++) {  
 sum += A.data[y][i] \* B.data[i][x];  
 }  
 C.data[y][x] = sum;  
 }  
 }  
 return C;  
}

Sumuje się wartości poszczególnych kolejnych kolumn wierszy przez kolejne wiersze w kolumnie drugiej macierzy o indeksie wiersza macierzy pierwszej.

### Zrównoleglenie

Zrównoleglenie pętli:

* Iterującej po wierszach macierzy pierwszej
* Iterującej po kolumnach macierzy pierwszej
* Iterującej po wierszach macierzy drugiej

Co daje nam 8 różnych metod, wliczając w to brak zrównoleglenia.

W wynikach zastosowałem następujące oznaczenia:

**XXX** – gdzie każda litera oznacza poszczególną pętlę wymienioną wyżej w punktach

* R – Oznacza zrównoleglenie przez redukcję
* N – oznacza brak zrównoleglenia

Z czego wynika

**RNR:** oznacza zrównoleglenie pętli iterujących po wierszach macierzy pierwszej i macierzy drugiej i brak zrównoleglenia po kolumnach macierzy pierwszej.

Wyniki były generowane w środowisku Linux pod wirtualizacją **WSL2. Ubuntu 20.04.**   
Wykresy przedstawiają zależności czasu wykonywania mnożenia, w zależności od wielkości wygenerowanych macierzy, a także wybranej metody zrównoleglenia. Wykresy są podzielone także na ustawione porcje.

## Wyniki

### Schedule(static)

### Schedule(static,4)

### Schedule(dynamic)

### Schedule(dynamic,4)

## Podsumowanie

Analizując uzyskane wyniki mogę stwierdzić, że: operacje przy braku zrównoleglenia są o połowę wolniejsze, im większe wartości tym jest to eksponencjalnie większe. Przy N=1000 jest to np. prawie 4 razy szybciej.

Zrównoleglenie tylko pętli iterującej po wierszach drugiej macierzy nie przynosi dobrego rezultatu, użycie samego zrównoleglenia dla tej pętli jest losowo wolniejsze od braku zrównoleglenia.

W przypadku pozostałych zrównolegleń różnice są niewielkie, czasami występują „piki” czasu wykonywania, ale jest to związane z wykonywaniem przez procesor dodatkowych operacji w tle. Przy czasie wykonywania 1sek, „niewielkie” zadania przekładają się na wzrosty czasu.

Dla największych wielkości macierzy najszybsze wydają się być tylko zrównoleglenie pętli iterującej po wierszach macierzy pierwszej. Podobny wynik daje zrównoleglenie pętli iterującej po kolumnach macierzy pierwszej.

Zrównoleglenie wszystkich pętli nie daje „oczekiwanego” najszybszego rezultatu

## Cel dodatkowy

Moim celem była próba dla wielkości N = 1000 zejść poniżej 1 sekundy

### Wyniki dla N = 1000

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Podział porcji** | **RRR** | **RRN** | **RNR** | **NRR** | **RNN** | **NRN** | **NNR** | **NNN** |
| static (T:12) | 1.47863 | 1.49596 | 1.33791 | 1.47429 | 1.39084 | 1.42727 | 2.71049 | 4.01588 |
| static4 (T:12) | 1.3294 | 1.30029 | 1.21698 | 1.50701 | 1.2408 | 1.31069 | 2.6256 | 4.15502 |
| dynamic (T:12) | 3.15256 | 1.76224 | 2.88321 | 3.87111 | 1.37692 | 1.71273 | 2.465 | 4.04912 |
| dynamic4 (T:12) | 1.7197 | 1.39382 | 1.57094 | 1.64607 | 1.36967 | 1.32363 | 7.49492 | 4.13644 |
| dynamic3 (T:8) | 2.12352 | 1.4129 | 2.00013 | 2.34112 | 1.42282 | 1.42109 | 9.05243 | 5.31303 |
| static (T:12) | 1.65204 | 1.56163 | 1.51833 | 1.82825 | 1.48926 | 1.43693 | 3.56061 | 4.55615 |

### Podsumowanie

Jednak dla wielu prób, najniższą uzyskaną wartością było: 1.21 sekundy.

Uzyskane przez statyczny podział porcji o rozmiarze 4 dla 12 wątków.

### Dodatek

Uruchomiłem też testy dla N=2000

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| rrr | rrn | rnr | nrr | rnn | rnr | nnr |
| 16.9848 | 16.2481 | 15.0193 | 17.3307 | 16.2316 | 18.4109 | 21.5588 |

Dla większej ilości danych: rozmiar macierzy N=2000, wynika, że najszybsze jest zrównoleglenie pętli iterującej po wierszach macierzy pierwszej i kolumnach macierzy drugiej. (to potwierdza też wcześniejsze testy)

Ponieważ ten test trwa dość długo, ciężko jest wygenerować większą ilość pomiarów.