

Akademia Nauk Stosowanych w Nowym Sączu Programowanie Współbieżne i Rozproszone			
Temat: Metoda Monte Carlo, zrównoleglenie obliczeń.			spr nr 7
Nazwisko i imię: Ciapała Tadeusz		Ocena sprawozdania	Zaliczenie:
Data wykonania ćwiczenia: 4.04.2023	Grupa: P3		

1) Zadanie nr 1 - metoda Monte Carlo:

Treść zadania: "Zrównoleglj obliczenia metody Monte-Carlo, umożliwiające obliczenie przybliżonej wartości π ."

Metoda Monte Carlo - jest to metoda, która wykorzystuje liczby pseudolosowe losowane zgodnie z zadany rozkładem do obliczenia wyniku / rozwiązania danego problemu (np. całkowanie). Losowanie to jest nazywane wyborem przypadkowym. Dokładność wyniku tej metody jest zależna od jakości generatora liczb pseudolosowych oraz od liczby sprawdzeń wylosowanych liczb.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <chrono>
#include <vector>
#include <thread>

double piNiezrownoleglone() {
    const int num_points = 1000000;
    int num_inside_circle = 0;
    std::srand(std::time(nullptr));

    for (int i = 0; i < num_points; i++) {
        double x = ((double)std::rand() / RAND_MAX) * 2 - 1;
        double y = ((double)std::rand() / RAND_MAX) * 2 - 1;
        if (std::pow(x, 2) + std::pow(y, 2) <= 1) {
            num_inside_circle++;
        }
    }

    return 4.0 * num_inside_circle / num_points;
}

double piZrownoleglone(int num_threads) {
    const int num_points = 1000000;
    int num_inside_circle = 0;
    std::srand(std::time(nullptr));
    std::vector<std::thread> threads(num_threads);

    for (int t = 0; t < num_threads; t++) {
        threads[t] = std::thread([&] {
            int temp_points_circle = 0;
            for (int i = 0; i < num_points / num_threads; i++) {
                double x = ((double)std::rand() / RAND_MAX) * 2 - 1;
                double y = ((double)std::rand() / RAND_MAX) * 2 - 1;
                if (std::pow(x, 2) + std::pow(y, 2) <= 1) {
                    temp_points_circle++;
                }
            }
            num_inside_circle += temp_points_circle;
        });
    }
}
```

```

        for (int t = 0; t < num_threads; t++) {
            threads[t].join();
        }

        return 4.0 * num_inside_circle / num_points;
    }

int main() {
    auto start = std::chrono::steady_clock::now();
    double pi1 = piNiezrownoleglone();
    auto end = std::chrono::steady_clock::now();
    printf("Czas trwania piNiezrownoleglone() [MS]: %llu\n",
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());
    std::cout << "Approximate value of Pi (1) : " << pi1 << std::endl;

    start = std::chrono::steady_clock::now();
    double pi2 = piZrownoleglone(4);
    end = std::chrono::steady_clock::now();
    printf("Czas trwania piZrownoleglone() [MS]: %llu\n",
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());
    std::cout << "Approximate value of Pi (2) : " << pi2 << std::endl;

    return 0;
}

```

Listing nr 1: kod do zadania pierwszego

Mamy tutaj funkcję `main()`, w której są zwracane wyniki π oraz obliczane czasy wykonywania obliczeń. Ponad funkcją główną są dwie funkcje: `piNiezrownoleglone()` oraz `piZrownoleglone()`. Pierwsza z nich, to kod dostarczony do wykonania zadania, natomiast druga, to realizacja polecenia w zadaniu. Wykorzystałem tutaj bibliotekę `<thread>`, aby móc wykonać generowanie liczb na kilku wątkach jednocześnie.

```

WybierzKonsola debugowania programu Microsoft Visual Studio
Czas trwania piNiezrownoleglone() [MS]: 150
Approximate value of Pi (1) : 3.1397
Czas trwania piZrownoleglone() [MS]: 41
Approximate value of Pi (2) : 3.14405

C:\Users\Tadek\Documents\PROJECTS\pwir05\x64\Debug\pwir05.exe (proces 12188) zakończono z kodem 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...

```

Zrzut nr 1: porównanie dwóch funkcji używających metody Monte Carlo

Przy wywołaniu funkcji `piZrownoleglone()` podałem jako argument liczbę 4. Oznacza to, że generowanie liczb będzie wykonywane na 4 wątkach. Na zrzucie nr 1 widać znaczną różnicę w czasie - po zrównolegleniu operacji funkcja wykonała się ponad trzykrotnie szybciej. Przybliżenia liczby π są zależne od wygenerowanych liczb.

2) Zadanie nr 2 - Własny algorytm:

Treść zadania: "Przygotuj własny przykład algorytmu, który poddasz zrównolegleniu. Opisz go i udokumentuj."

Wybrany przeze mnie algorytmem było mnożenie macierzowe dwóch macierzy. Do zrównoleglenia tej operacji posłużyłem się biblioteką <thread> tak samo jak w poprzednim zadaniu. Rozwiązanie zadania polega na mnożonych macierzy na bloki, które można przypisać do osobnych wątków.

```
#include <iostream>
#include <thread>
#include <vector>
#include <chrono>

using namespace std;

// funkcja do mnożenia dwóch macierzy
void mnozenieMacierzowe(const vector<vector<int>>& A, const vector<vector<int>>& B,
vector<vector<int>>& C, int start, int koniec) {
    int n = A.size();
    int m = B[0].size();
    int p = B.size();

    for (int i = start; i < koniec; i++) {
        for (int j = 0; j < m; j++) {
            C[i][j] = 0;
            for (int k = 0; k < p; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

// funkcja do mnożenia dwóch macierzy z użyciem wielu wątków
vector<vector<int>> zrownoleglenieMnozenia(const vector<vector<int>>& A, const
vector<vector<int>>& B, int num_threads) {
    int n = A.size();
    int m = B[0].size();
    vector<vector<int>> C(n, vector<int>(m, 0)); // utwórz macierz wynikową
    int wiersze_na_thread = n / num_threads; // podziel macierz na równe bloki
    vector<thread> threads(num_threads);

    for (int t = 0; t < num_threads; t++) {
        int start = t * wiersze_na_thread;
        int koniec = (t == num_threads - 1) ? n : start + wiersze_na_thread;
        threads[t] = thread(mnozenieMacierzowe, ref(A), ref(B), ref(C), start, koniec);
    }

    for (int t = 0; t < num_threads; t++) {
        threads[t].join();
    }

    return C;
}

// funkcja do wyświetlania macierzy
void wyswietlMacierz(const vector<vector<int>>& matrix) {
    for (const auto& row : matrix) {
        for (const auto& elem : row) {
            cout << elem << " ";
        }
        cout << endl;
    }
}

int main() {
    //vector<vector<int>> macierz_A = {{1, 2, 3},{4, 5, 6},{7, 8, 9}};
    //vector<vector<int>> macierz_B = {{9, 8, 7},{6, 5, 4},{3, 2, 1}};
    vector<vector<int>> macierz_A = {{11, 22, 33},{44, 55, 66},{77, 88, 99}};
    vector<vector<int>> macierz_B = {{99, 88, 77},{66, 55, 44},{33, 22, 11}};
```

```

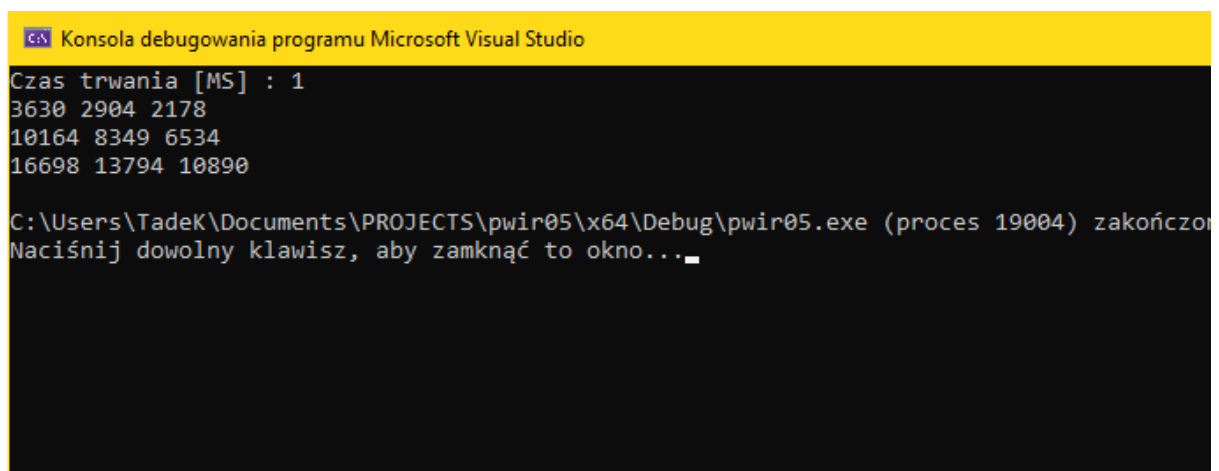
    auto start = chrono::steady_clock::now();
    vector<vector<int>> wynik = zrownoleglenieMnozenia(macierz_A, macierz_B, 4); //
    wykonaj mnożenie macierzowe z użyciem 4 wątków
    auto end = chrono::steady_clock::now();
    printf("Czas trwania [MS] : %llu\n", chrono::duration_cast<chrono::milliseconds>(end
- start).count());
    wyswietlMacierz(wynik); // wyświetl macierz wynikową

    return 0;
}

```

Listing nr 2: rozwiązanie zadania drugiego

Kod składa się z trzech funkcji - jedna służy do wykonania pożądanej operacji, druga do podziału obliczeń na podaną ilość wątków, a trzecia do wyświetlenia wyniku w funkcji głównej programu. Jest to dosyć prosty algorytm, więc po zrównolegleniu mnożenie macierzowe dla macierzy rzędu trzeciego czas wykonania jest bliski 1 milisekundy.



```

C:\> Konsola debugowania programu Microsoft Visual Studio

Czas trwania [MS] : 1
3630 2904 2178
10164 8349 6534
16698 13794 10890

C:\Users\TadeK\Documents\PROJECTS\pwir05\x64\Debug\pwir05.exe (proces 19004) zakończon
Naciśnij dowolny klawisz, aby zamknąć to okno..._

```

Zrzut nr 2: uruchomienie programu z zadania nr 2