

| | | |
|--|--------|------------------|
| Programowanie współbieżne i rozproszone | PWIR03 | |
| 12.04.2022 | P2 | Szymon Zwoliński |

KOD 4

wielowątkowe dodawanie dwóch tablic do siebie indeks po indeksie:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <thread.h>

#define SIZE 40

void add(int id, int* a, int* b, int* c) {
    c[id] = a[id] + b[id];
}

int main() {
    srand(time(NULL));
    int a[SIZE];
    int b[SIZE];
    int c[SIZE];

    for (int i = 0; i < SIZE; i++) {
        a[i] = rand() % 100 + 1; //1 do 100
        b[i] = rand() % 100 + 1;
    }

    //wypisanie na ekranie A
    for (int i = 0; i < SIZE; i++) {
        printf("%u ", a[i]);
    }
    printf("\n");

    //wypisanie na ekranie B
    for (int i = 0; i < SIZE; i++) {
        printf("%u ", b[i]);
    }
    printf("\n");

    std::thread** threads = new std::thread * [SIZE];
    for (int i = 0; i < SIZE; i++) {
        threads[i] = new std::thread(add, i, a, b, c); //wykorzystuje i jako id danego wÅtku
    }

    for (int i = 0; i < SIZE; i++) {
        threads[i]->join();
    }

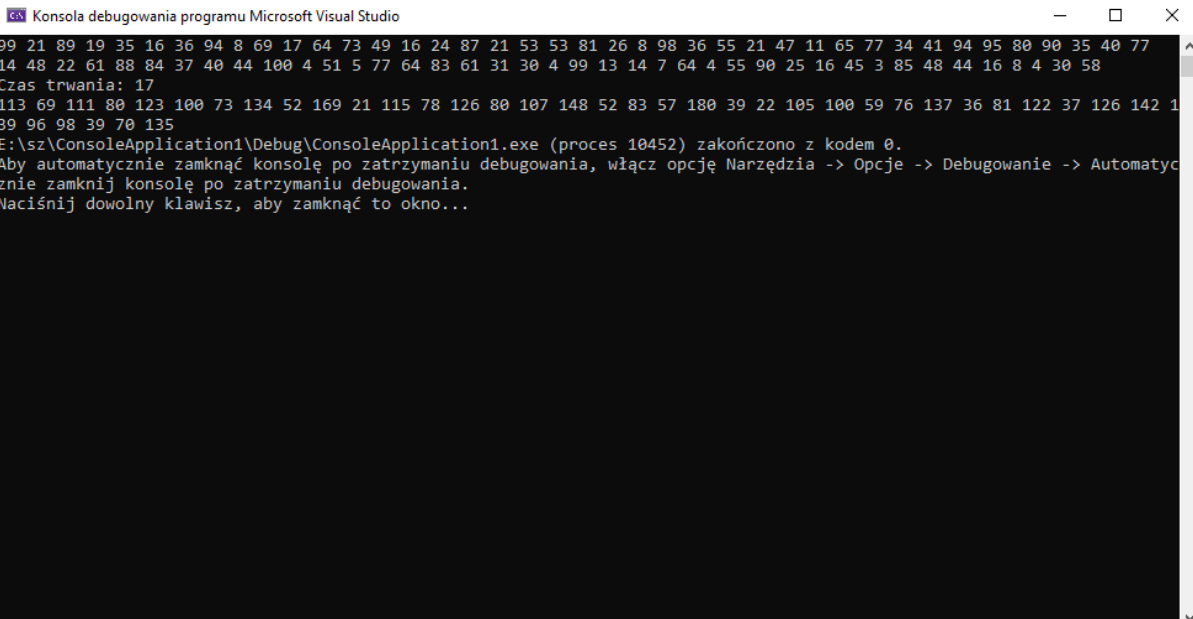
    for (int i = 0; i < SIZE; i++) {
        delete threads[i];
    }
    delete[] threads;

    //wypisanie na ekranie C
    for (int i = 0; i < SIZE; i++) {
        printf("%u ", c[i]);
    }

    return 0;
}
```

Powyższy program tworzy tablice o zadanym rozmiarze. Wypełnia losowymi danymi z zakresu 1 do 100. Otwiera tyle wątków ile jest elementów w tablicy. Wątki wykonują dodawanie każdy pod swoim indeksem.

4.1. Dodaj pomiar czasu dla powyższego kodu (tak by pomiar obejmował tylko faktyczne dodawanie)



Konsola debugowania programu Microsoft Visual Studio

```
99 21 89 19 35 16 36 94 8 69 17 64 73 49 16 24 87 21 53 53 81 26 8 98 36 55 21 47 11 65 77 34 41 94 95 80 90 35 40 77
14 48 22 61 88 84 37 40 44 100 4 51 5 77 64 83 61 31 30 4 99 13 14 7 64 4 55 90 25 16 45 3 85 48 44 16 8 4 30 58
Czas trwania: 17
113 69 111 80 123 100 73 134 52 169 21 115 78 126 80 107 148 52 83 57 180 39 22 105 100 59 76 137 36 81 122 37 126 142 1
39 96 98 39 70 135
E:\sz\ConsoleApplication1\Debug\ConsoleApplication1.exe (proces 10452) zakończono z kodem 0.
Aby automatycznie zamknąć konsolę po zatrzymaniu debugowania, włącz opcję Narzędzia -> Opcje -> Debugowanie -> Automatyc
znie zamknij konsolę po zatrzymaniu debugowania.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

```
std::thread** threads = new std::thread * [SIZE];
auto start = std::chrono::steady_clock::now();
for (int i = 0; i < SIZE; i++) {
    threads[i] = new std::thread(add, i, a, b, c); //wykorzystuje i jako id danego wątku
}
auto end = std::chrono::steady_clock::now();
printf("Czas trwania: %llu\n", std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());
```

dodawanie 40 liczb w 40 wątkach wykonuje się aż 17 sekund.

4.2. Zmodyfikuj program tak by każdy wątek dodawał 10 komórek.

Wątek o id 0 dodaje indeksy 0 z 0, 1 z 1, ... 9 z 9

Wątek o id 1 dodaj indeksy 10 z 10, ... 19 z 19

Itd.

Dostosuj SIZE i ilość wątków tak by to miało sens.

Dodaj pomiar czasu

```
Konsola debugowania programu Microsoft Visual Studio

1 12
1 13
1 14
1 15
1 16
1 17
1 18
1 19
2 20
2 21
2 22
2 23
2 24
2 25
2 26
2 27
2 28
2 29
3 30
3 31
3 32
3 33
3 34
3 35
3 36
3 37
3 38
3 39
Czas trwania: 37
89 101 151 98

std::thread** threads = new std::thread * [SIZE];
auto start = std::chrono::steady_clock::now();
for (int i = 0; i < (SIZE); i++) {
    for (int j = i * 10; j < ((i * 10) + 10); j++)
    {
        std::cout << i << " " << j<< std::endl;
        threads[i] = new std::thread(add, i, a, b, c); //wykorzystuje i jako id danego wÅtku
    }
}
auto end = std::chrono::steady_clock::now();
printf("Czas trwania: %llu\n", std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());
```

Dodanie 10 liczb do jednego wątku wykonuje się znacznie dłużej niż dodawanie wykonane w zadaniu 4.1. Zdefiniowana stała liczba SIZE określa ilość wątków.

4.3. Odpowiedz na pytanie co trzeba zmienić i dlaczego by rozmiar a, b,c można było pobierać od użytkownika.

```
int rozmiar;
std::cout << "Podaj rozmiar: ";
std::cin >> rozmiar;
int *a = (int*)malloc(rozmiar);
int *b= (int*)malloc(rozmiar);
int *c = (int*)malloc(rozmiar);
```

Aby zadeklarować rozmiar przypisany od użytkownika, należy zarezerwować odpowiednio duże miejsce w pamięci, oraz tablice traktować jako wskaźniki. W przypadku kodu programu 4, należy wprowadzić rozmiar mniejszy niż rozmiar zadeklarowany jako SIZE.

KOD 5

```
#include <stdio>
#include <thread>
#include <chrono>

double sum;

void calculatePI(int id, double step, unsigned long long steps_per_thread) {
    double x = 0;
    double temp_sum = 0;

    for (unsigned long long i = id * steps_per_thread; i < (id + 1) * steps_per_thread; i++) {
        x = (i + 0.5) * step;
        temp_sum = temp_sum + 4.0 / (1.0 + x * x);
    }

    sum += temp_sum;
}

int main() {
    int threads_count = 10;
    unsigned long long steps = 90000000000;
    double step = 1.0 / steps;

    auto start = std::chrono::high_resolution_clock::now();

    std::thread** threads = new std::thread * [threads_count];

    for (uint32_t i = 0; i < threads_count; i++) {
        threads[i] = new std::thread(calculatePI, i, step, steps / threads_count);
    }

    for (uint32_t i = 0; i < threads_count; i++) {
        threads[i] -> join();
    }

    double PI = step;
    PI *= sum;

    auto end = std::chrono::high_resolution_clock::now();

    printf("PI: %f in time: %llu ms\r\n", PI,
        std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());

    for (uint32_t i = 0; i < threads_count; i++) {
        delete threads[i];
    }
    delete[] threads;

    return 0;
}
```

W powyższym przykładzie wszystkie wątki robią inkrementacje zmiennej sum, która jest wspólna dla wszystkich wątków. Sytuacja ta może dać poprawny wynik, ale może dać też zły.

Problem rozwiązany za pomocą redukcji poniżej. Wątki nie korzystają z obliczeń innych wątków więc nie ma problemu.

Każdy wątek liczy swoją porcję iteracji i zapisuje do bufora pod indeks == id wątku. Po synchronizacji w wątku głównym wyniki pod problemów są sumowane i wykorzystywane w finalnych obliczeniach

5.1. Przetestuj poprawność wywołując kod wiele razy.

Ilość kroków wyliczających Pi, ze względu na ograniczenia sprzętowe została znacznie ograniczona, przez co program wykonuje się bardzo szybko.

5.2. Przetestuj czasy wykonania dla różnych ilości wątków i kroków.

Dla 100000000 kroków i 10 wątków

Dla 100000000 kroków i 20 wątków

dla 200000000 i 10 wątków

dla 200000000 i 20 wątków

Przy mniejszej ilości kroków, program wykonuje się szybciej przy mniejszej ilości wątków. Zwiększając ilość kroków, program wykonuje się szybciej przy większej ilości wątków

KOD 6

```

#include <cstdio>
#include <thread>
#include <chrono>

double* sums;

void calculatePI(int id, double step, unsigned long long steps_per_thread) {
    double x = 0;
    double temp_sum = 0;

    for (unsigned long long i = id * steps_per_thread; i < (id + 1) * steps_per_thread; i++) {
        x = (i + 0.5) * step;
        temp_sum = temp_sum + 4.0 / (1.0 + x * x);
    }

    sums[id] = temp_sum;
}

int main() {
    int threads_count = 10;
    unsigned long long steps = 90000000000;
    double step = 1.0 / steps;

    auto start = std::chrono::high_resolution_clock::now();

    std::thread** threads = new std::thread * [threads_count];
    sums = new double[threads_count];

    for (uint32_t i = 0; i < threads_count; i++) {
        threads[i] = new std::thread(calculatePI, i, step, steps / threads_count);
    }

    for (uint32_t i = 0; i < threads_count; i++) {
        threads[i] -> join();
    }

    double PI = step;
    double s = 0;
    for (int i = 0; i < threads_count; i++) s += sums[i]; //redukcja

    PI *= s;

    auto end = std::chrono::high_resolution_clock::now();

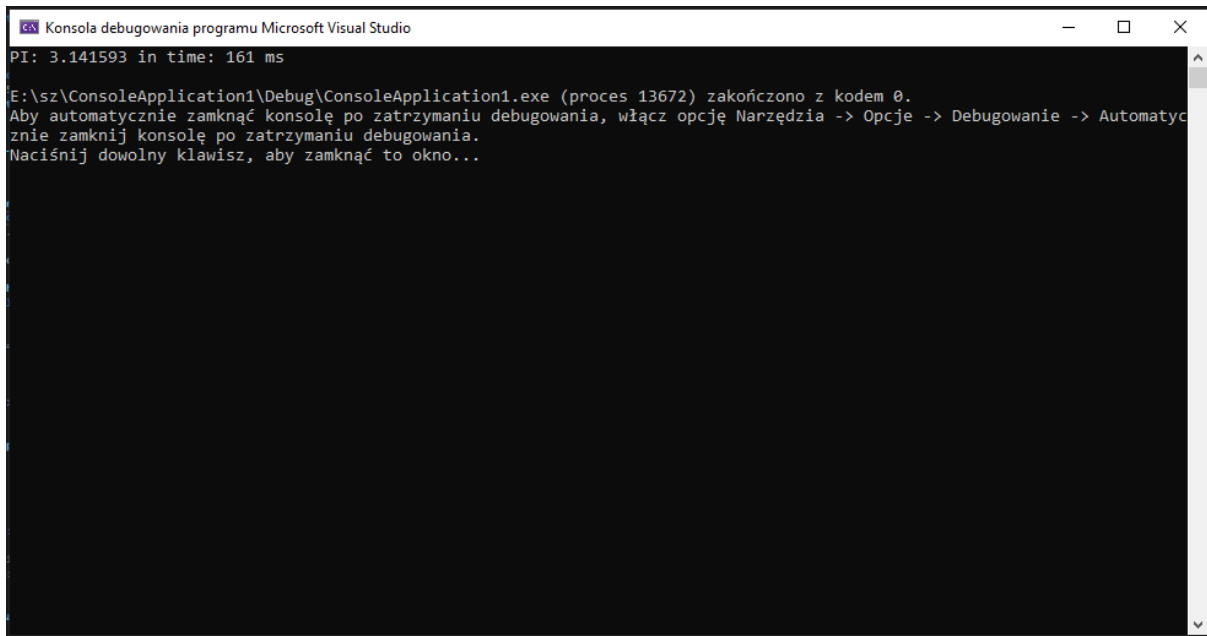
    printf("PI: %f in time: %llu ms\r\n", PI,
        std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());

    for (uint32_t i = 0; i < threads_count; i++) {
        delete threads[i];
    }
    delete[] threads;
    delete[] sums;

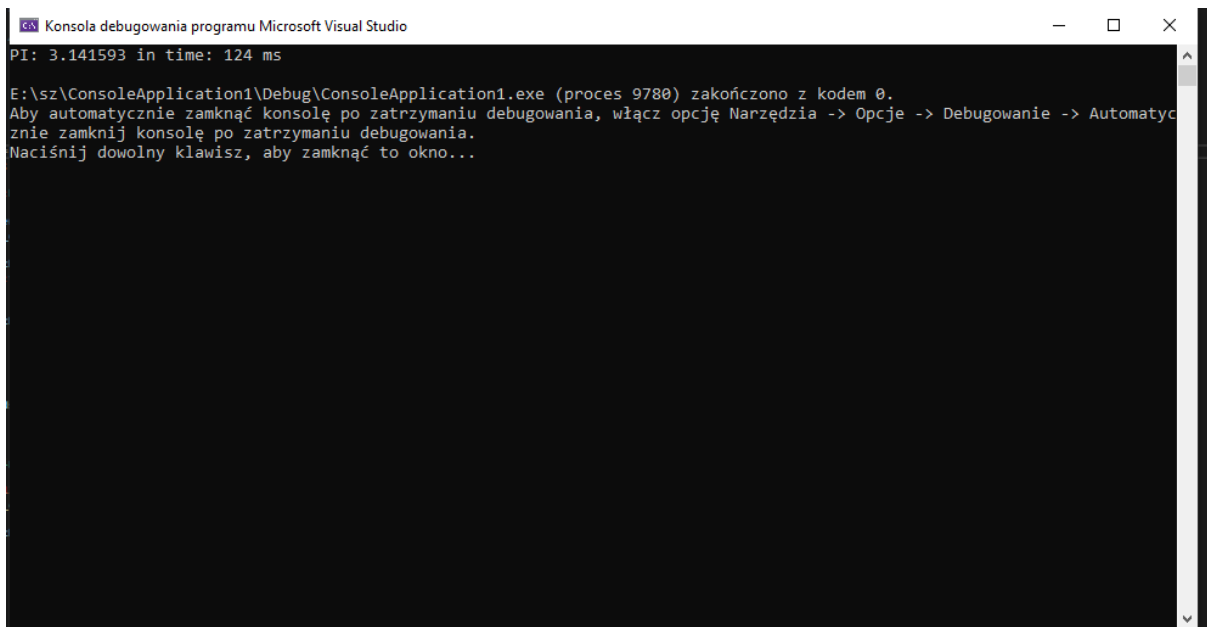
    return 0;
}

```

6.1 Przetestuj poprawność wywołując kod wiele razy.



```
Konsola debugowania programu Microsoft Visual Studio
PI: 3.141593 in time: 161 ms
E:\sz\ConsoleApplication1\Debug\ConsoleApplication1.exe (proces 13672) zakończono z kodem 0.
Aby automatycznie zamknąć konsolę po zatrzymaniu debugowania, włącz opcję Narzędzia -> Opcje -> Debugowanie -> Automatycznie zamknij konsolę po zatrzymaniu debugowania.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```



```
Konsola debugowania programu Microsoft Visual Studio
PI: 3.141593 in time: 124 ms
E:\sz\ConsoleApplication1\Debug\ConsoleApplication1.exe (proces 9780) zakończono z kodem 0.
Aby automatycznie zamknąć konsolę po zatrzymaniu debugowania, włącz opcję Narzędzia -> Opcje -> Debugowanie -> Automatycznie zamknij konsolę po zatrzymaniu debugowania.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

Ilość kroków wyliczających Pi, ze względu na ograniczenia sprzętowe została znacznie ograniczona, przez co program wykonuje się bardzo szybko. Program oblicza wartość Pi poprawnie.

6.2

Dla 100000000 i 10 wątków



```
Konsola debugowania programu Microsoft Visual Studio
PI: 3.141593 in time: 133 ms
E:\sz\ConsoleApplication1\Debug\ConsoleApplication1.exe (proces 8536) zakończono z kodem 0.
```

Dla 100000000 i 20 wątków



```
Konsola debugowania programu Microsoft Visual Studio
PI: 3.141593 in time: 136 ms
E:\sz\ConsoleApplication1\Debug\ConsoleApplication1.exe (proces 8536) zakończono z kodem 0.
```

Dla 200000000 i 10 wątków

A screenshot of a Visual Studio debug console window. The title bar reads "Konsola debugowania programu Microsoft Visual Studio". The console output shows "PI: 3.141593 in time: 255 ms".

```
Konsola debugowania programu Microsoft Visual Studio
PI: 3.141593 in time: 255 ms
```

Dla 200000000 i 20 wątków

A screenshot of a Visual Studio debug console window. The title bar reads "Konsola debugowania programu Microsoft Visual Studio". The console output shows "PI: 3.141593 in time: 239 ms".

```
Konsola debugowania programu Microsoft Visual Studio
PI: 3.141593 in time: 239 ms
```

Program Podobnie jak w kodzie 5, dla małej liczby kroków wykonuje się szybciej na mniejszej ilości wątków, jednak zwiększając liczbę kroków, program wykonuje się szybciej na większej liczbie wątków.