

Akademia Nauk Stosowanych w Nowym Sączu Programowanie Współbieżne i Rozproszone		
Temat: Problem pisania do tej samej zmiennej.		
Nazwisko i imię: Ciapała Tadeusz	Ocena sprawozdania	Zaliczenie:
Data wykonania ćwiczenia: 14.03.2023	Grupa: P3	

### 1) KOD 5:

Na tych zajęciach zajmowaliśmy się problemem jednoczesnego pisania do tej samej zmiennej przez kilka wątków jednocześnie. Aby zaprezentować i rozwiązać problem, posłużyliśmy się metodą Wallic'a do obliczania wartości liczby  $\pi$ . Im więcej iteracji, to tym bardziej precyzyjne są nasze przybliżenia wartości  $\pi$  po przecinku.

```
#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 = 9000000000;
    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;
}

```

Listing nr 1: kod nr 5 dostarczony do wykonania zadań

Pierwszy kod uruchamiamy w takiej postaci jaki otrzymaliśmy. Kod działa w taki sposób, że wszystkie utworzone w nim wątki i podpięte do procesu głównego inkrementują sumę zmiennej. Zmienna ta jest wspólna dla wszystkich pracujących na niej wątków. Największą wadą tego rozwiązania jest to, że nie mamy gwarancji poprawności wykonanych obliczeń.

Zadaniami do wykonania na tym kodzie były testy jego poprawności wywołując go wiele razy oraz testy czasowe wykonania dla różnych ilości kroków i wątków.

```

5_pi_bad.cpp ConsoleApplication2.cpp
application2 (Globalny zasięg) main()
    temp_sum = temp_sum + 4.0 / (1.0 + x * x);
    }
    sum += temp_sum;
    }
int main() {
    int threads_count = 10;
    unsigned long long steps = 9000000000;
    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, step, i);
    }

    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;

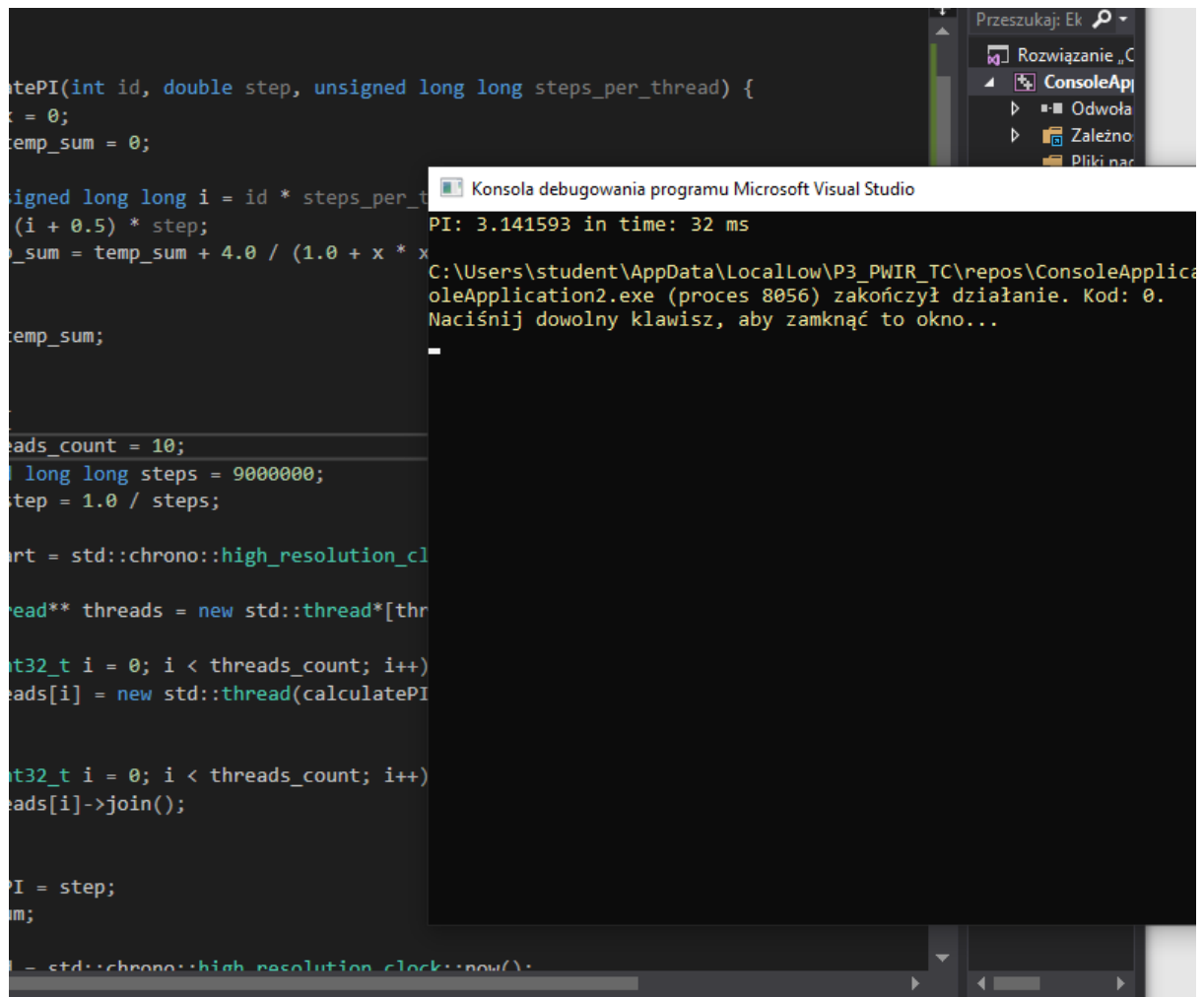
    return 0;
}

Wybierz konsolę debugowania programu Microsoft Visual Studio
PI: 3.141593 in time: 302965 ms
C:\Users\student\AppData\LocalLow\P3_PWIR_TC\repos\ConsoleApplication2\Debug\ConsoleApplication2.exe (proces 10080) zakończył działanie. Kod: 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...

wyjściowe: Kompilacja
Kompilacja rozpoczęta: Projekt: ConsoleApplication2, Konfiguracja: Debug Win32 -----
Application2.vcxproj -> C:\Users\student\AppData\LocalLow\P3_PWIR_TC\repos\ConsoleApplication2\Debug\ConsoleApplication2.exe
== Kompilacja: 1 zakończono powodzeniem, 0 zakończono niepowodzeniem, 0 zaktualizowano, 0 pominięto =====

```

Zrzut nr 1: pierwsze uruchomienie kodu



The screenshot shows the Microsoft Visual Studio IDE. On the left, a C++ source file is open, displaying a function `calculatePI` and a `main` function. The `main` function sets `threads_count = 10`, `steps = 9000000`, and `step = 1.0 / steps`. It then creates an array of `std::thread` objects and launches them to calculate the value of  $\pi$  using the `calculatePI` function. The `main` function waits for all threads to finish using `join()` and then prints the result.

On the right, the 'Console' window is open, showing the output of the program. The output is: `PI: 3.141593 in time: 32 ms`. Below this, a message indicates that the application has finished execution with code 0 and prompts the user to press a key to close the window.

```
calculatePI(int id, double step, unsigned long long steps_per_thread) {  
    double sum = 0;  
    double temp_sum = 0;  
  
    for (unsigned long long i = id * steps_per_thread; i < (id + 1) * steps_per_thread; i++)  
    {  
        temp_sum = temp_sum + 4.0 / (1.0 + x * x);  
    }  
  
    sum = sum + temp_sum;  
  
    return sum;  
}  
  
int main()  
{  
    const int threads_count = 10;  
    const unsigned long long steps = 9000000;  
    const double step = 1.0 / steps;  
  
    auto start = std::chrono::high_resolution_clock::now();  
  
    std::thread** threads = new std::thread*[threads_count];  
  
    for (int i = 0; i < threads_count; i++)  
    {  
        threads[i] = new std::thread(calculatePI, i, step, steps);  
    }  
  
    for (int i = 0; i < threads_count; i++)  
    {  
        threads[i] -> join();  
    }  
  
    double PI = step * sum;  
    auto end = std::chrono::high_resolution_clock::now();  
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count();  
    std::cout << "PI: " << PI << " in time: " << duration << " ms" << std::endl;  
    return 0;  
}
```

Konsola debugowania programu Microsoft Visual Studio  
PI: 3.141593 in time: 32 ms  
C:\Users\student\AppData\LocalLow\P3\_PWIR\_TC\repos\ConsoleApplication2.exe (proces 8056) zakończył działanie. Kod: 0.  
Naciśnij dowolny klawisz, aby zamknąć to okno...

Zrzut nr 2: ponowne uruchomienie kodu, z tym, że ilość kroków wynosi 9000000

```
0;

g long i = id * steps_per_thread; i < (id + 1)*steps_per_thread; i++)
* step
mp_sum
```

Konsola debugowania programu Microsoft Visual Studio

```
PI: 3.142426 in time: 1 ms

C:\Users\student\AppData\LocalLow\P3_PWIR_TC\repos\ConsoleApplication2
oleApplication2.exe (proces 9264) zakończył działanie. Kod: 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

```
= 1;
g steps
/ step

:chrono

eads =

0; i <
new std

0; i <
oin();
```

Zrzut nr 3: uruchomienie kodu, gdzie ilość wątków = 1, ilość kroków = 10

```
um = 0;

long long
0.5) * step
= temp_sum

um;

ount = 100;
long steps
1.0 / step

std::chrono

threads =

i = 0; i <
= new std
```

Konsola debugowania programu Microsoft Visual Studio

PI: 0.000000 in time: 24 ms

C:\Users\student\AppData\LocalLow\P3\_PWIR\_TC\repos\ConsoleApplication2\Debug\ConsoleApplication2.exe (proces 10252) zakończył działanie. Kod: 0.  
Naciśnij dowolny klawisz, aby zamknąć to okno...

Zrzut nr 4: uruchomienie kodu, gdzie ilość wątków = 100, ilość kroków = 50

Na zrzucie ekranu nr 4 widzimy ciekawą sytuację, gdzie obliczenia totalnie zawiodły. Dzieje się tak w momencie, gdy podamy większą ilość wątków (nawet o 1) od ilości wykonywanych kroków.

```
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);
}

sum += temp_sum;

int main() {
    int threads_count = 5;
    unsigned long long steps = 5;
    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++)
```

Konsola debugowania programu Microsoft Visual Studio

PI: 3.144926 in time: 2 ms

C:\Users\student\AppData\LocalLow\P3\_PWIR\_TC\repos\ConsoleApplication2\Debug\ConsoleApplication2.exe (proces 6112) zakończył działanie. Kod: 0.  
Naciśnij dowolny klawisz, aby zamknąć to okno...

Zrzut nr 5: ilość wątków = 5, ilość kroków = 5

```
double 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 = 5;
    unsigned long long steps = 9;
    double step = 1.0 / steps;

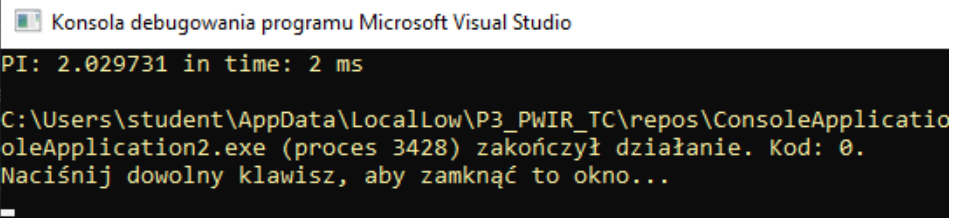
    auto start = std::chrono::high_resolution_clock::now();

    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);
    }

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

    double PI = step;
}
```



Zrzut nr 6: ilość wątków = 5, ilość kroków = 9

Po przeanalizowaniu dotychczasowych zrzutów, można wysunąć wniosek, że ilość kroków powinna być wielokrotnością liczby wątków. W przypadku małych liczb ma to bardzo duże znaczenie. Przykładem jest zrzut nr 5 i nr 6: na zrzucie piątym przybliżenie liczby  $\pi$  wynosi 3.14. Natomiast na szóstym wynosi ono 2.03. Jednak obserwacja tych wielokrotności wydaje się nie mieć znaczenia przy bardzo dużych wartościach

```
x = (i + 0.5) * step;
temp_sum = temp_sum + 4.0 / (1.0 + x * x);
}

sum += temp_sum;

int main() {
    int threads_count = 500;
    unsigned long long steps = 9999999;
    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);
    }

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

    auto end = std::chrono::high_resolution_clock::now();
    double PI = sum;
    std::cout << "PI: " << PI << " in time: " << std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << " ms" << std::endl;
}
```

Konsola debugowania programu Microsoft Visual Studio

PI: 3.141493 in time: 72 ms

C:\Users\student\AppData\LocalLow\P3\_PWIR\_TC\repos\ConsoleApp\oleApplication2.exe (proces 9380) zakończył działanie. Kod: 0

Naciśnij dowolny klawisz, aby zamknąć to okno...

Zrzut nr 7: ilość wątków = 500, ilość kroków = 9999999

Oczywiście przy każdym tego typu testach należy pamiętać o roli sprzętowej. Różne procesory będą dawać różne czasy. Prawdopodobnie powinno się dopasowywać ilość wątków w programie do ilości wątków procesora: np tworzyć więcej wątków niż 12 jeżeli procesor ma 12 logicznych rdzeni.

## 2) KOD 6:

Ten kod ma rozwiązywać problem występujący w kodzie 5. Dlatego, że wątki nie korzystają z obliczeń pozostałych, to możemy zastosować redukcję. Każdy wątek wtedy będzie liczył swoją porcję danych, a wyniki będą sumowane w wątku głównym.

```
#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;
}

```

*Listing nr 2: dostarczony kod nr 6*

Warto tu zwrócić uwagę na zmianę, która zaszła w funkcji “calculatePI(..)”.



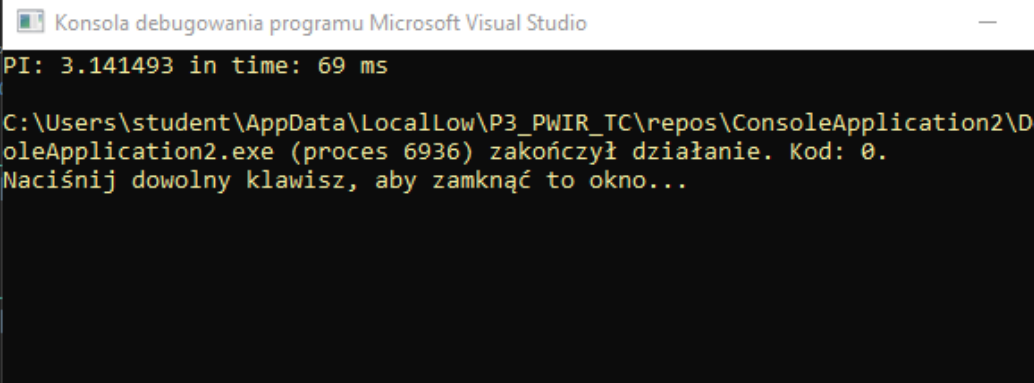
```
main() {
    int threads_count = 10;
    unsigned long long steps = 9000000000;
    double step = 1.0 / steps;

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

    std::thread
    sums = new (
    for (uint32_t i = 0; i < threads_count; i++) {
        threads[i] = new std::thread(
    }

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

    double PI = 0;
    double s = 0;
    for (int i = 0; i < threads_count; i++) {
        PI += s;
    }
}
```



```
PI: 3.141493 in time: 69 ms
C:\Users\student\AppData\LocalLow\P3_PWIR_TC\repos\ConsoleApplication2\Debug\ConsoleApplication2.exe (proces 6936) zakończył działanie. Kod: 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

Zrzut nr 8: pierwsze uruchomienie kodu nr 6

Jak widać po wyliczonym czasie wykonania obliczeń, kod nr 6 poradził sobie znacznie lepiej niż kod nr 5. W przypadku poprzedniego rozwiązania czas po pierwszym uruchomieniu wyniósł około 302 sekund, a w tym rozwiązaniu było to zaledwie 69 milisekund. Natomiast warto zwrócić uwagę na poprawność obliczeń. Obliczenia są dokładniejsze dla pierwszej metody. Mimo wszystko przybliżony wynik liczby  $\pi$  w obu przypadkach wynosił 3.14, więc druga metoda jest po prostu lepsza.

```
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;
}

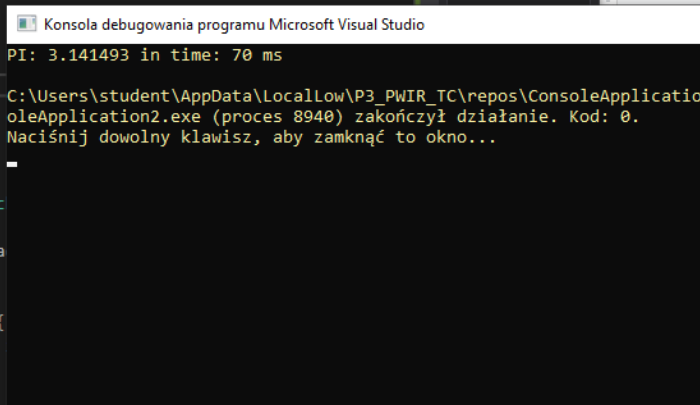
int main() {
    int threads_count = 10;
    unsigned long long steps = 9;
    double step = 1.0 / steps;

    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);
    }

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



```
PI: 3.141493 in time: 70 ms
C:\Users\student\AppData\LocalLow\P3_PWIR_TC\repos\ConsoleApplication2\Debug\ConsoleApplication2.exe (proces 8940) zakończył działanie. Kod: 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

Zrzut nr 9: ilość wątków = 10, ilość kroków = 9

```
double* sums;

int 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 = 5;
    unsigned long long steps = 5;
    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
```

[illegible]

I na sam koniec, bardziej przyziemny test, czyli ilość wątków równa ilości rdzeni logicznych procesora komputera udostępnionego na zajęciach - 4, oraz ilość kroków równa maksymalnej całkowitej wartości dla typu unsigned long long - 18446744073709551615.

```
double sums[4];

double 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++)
        temp_sum = temp_sum + 4.0 / (1.0 + x * x);

    sums[id] = temp_sum;
}

int main() {
    int threads_count = 4;
    unsigned long long steps = 18446744073709551615;
    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);
    }

    for (int i = 0; i < threads_count; i++)
        threads[i].join();

    auto end = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(end - start);

    std::cout << "PI: " << sums[0] << " in time: " << duration.count() << " ms" << std::endl;
}
```

Konsola debugowania programu Microsoft Visual Studio

PI: 3.141493 in time: 71 ms

C:\Users\student\AppData\LocalLow\P3\_PWIR\_TC\repos\oleApplication2.exe (proces 12324) zakończył działanie. Naciśnij dowolny klawisz, aby zamknąć to okno...

Zrzut nr 12: ilość wątków = 4, ilość kroków = 18446744073709551615