Problem pisania do tej samej zmiennej

Metoda Wallic'a obliczania wartości PI. Im więcej iteracji tym bardziej precyzyjne wyliczenie.

KOD5:

```cpp
#include <cstdio>
#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;
}
```

```cpp
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 porcje iteracji i zapisuje do bufora pod indeks == id wątku. Po synchronizacji w wątku głównym wyniki pod problemów są sumowane i wykorzystywane w finalnych obliczeniach.

KOD6:

```cpp
#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;
}
```

```cpp
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;
}
```

Zadanie do KOD5 i KOD6:

1. Przetestuj poprawność wywołując kod wiele razy.
2. Przetestuj czasy wykonania dla różnych ilości wątków i kroków.