Programowanie współbieżne i rozproszone	PWIR05	
10.05.2022	P2	Szymon Zwoliński

2.1. Zmodyfikuj program tak by mnożenie macierzy wykonywało się kilka razy, następnie zrównoleglij ten proces i wyświetl czas. Porównaj czasy przed zrównolegleniem i po modyfikacji. :

przed:

```
for (i = 0; i < MATRIX_H; i++) {
    for (k = 0; k < MATRIX_W; k++) {
        matrix[i][k] = (uint16_t)(rand() % 100);
    }
}
for (i = 0; i < MATRIX_H; i++) {
    for (k = 0; k < MATRIX_W; k++) {
        matrix[i][k] = (uint16_t)(rand() % 100);
    }
}
for (i = 0; i < MATRIX_H; i++) {
    for (k = 0; k < MATRIX_W; k++) {
        matrix[i][k] = (uint16_t)(rand() % 100);
    }
}
for (i = 0; i < MATRIX_H; i++) {
    for (k = 0; k < MATRIX_W; k++) {
        matrix[i][k] = (uint16_t)(rand() % 100);
    }
}
```

```
Konsola debugowania programu Microsoft Visual Studio
Fill in 6570 miliseconds
Calculated normal way in 227 miliseconds
Fill parallel way in 25875 miliseconds
Calculated parallel way in 225 miliseconds
```

Po:

```
#pragma omp parallel for num_threads(4)  shared(matrix) private(i, k)
    for (i = 0; i < MATRIX_H; i++) {
        for (k = 0; k < MATRIX_W; k++) {
            matrix[i][k] = (uint16_t)(rand() % 100);
        }
    }
#pragma omp parallel for num_threads(4)  shared(matrix) private(i, k)
    for (i = 0; i < MATRIX_H; i++) {
        for (k = 0; k < MATRIX_W; k++) {
            matrix[i][k] = (uint16_t)(rand() % 100);
        }
    }
#pragma omp parallel for num_threads(4)  shared(matrix) private(i, k)
    for (i = 0; i < MATRIX_H; i++) {
        for (k = 0; k < MATRIX_W; k++) {
            matrix[i][k] = (uint16_t)(rand() % 100);
        }
    }
#pragma omp parallel for num_threads(4)  shared(matrix) private(i, k)
    for (i = 0; i < MATRIX_H; i++) {
        for (k = 0; k < MATRIX_W; k++) {
            matrix[i][k] = (uint16_t)(rand() % 100);
        }
    }

#pragma omp parallel for num_threads(4) shared(vector) private(i)
    for (i = 0; i < VECTOR_S; i++) {
```

```
Konsola debugowania programu Microsoft Visual Studio
Fill in 6527 miliseconds
Calculated normal way in 227 miliseconds
Fill parallel way in 25827 miliseconds
Calculated parallel way in 225 miliseconds
```

Czas uzyskany poprzez zastosowanie zrównoleglenie są nieznacznie szybsze, przy 10000 iteracji.

3.1. Przetestuj każdą z opcji schedule. Wykonaj między 5 a 10 pomiarów (w zależności od wydajności) dla każdej z opcji a następnie wyciągnij z nich średnią. Wyniki zapisać i umieścić wraz z kodem w repozytorium.

```
Calculated parallel static way in 5652 miliseconds
Calculated parallel static N(MATRIX_H/10) way in 5644 miliseconds
Calculated parallel dynamic N(MATRIX_H/10) way in 5643 miliseconds
Calculated parallel guided N(MATRIX_H/10) way in 5636 miliseconds
Calculated parallel runtime way in 5650 miliseconds
```

```
Calculated parallel static way in 5652 miliseconds
Calculated parallel static N(MATRIX_H/10) way in 5652 miliseconds
Calculated parallel dynamic N(MATRIX_H/10) way in 5638 miliseconds
Calculated parallel guided N(MATRIX_H/10) way in 5635 miliseconds
Calculated parallel runtime way in 5647 miliseconds
```

```
Calculated parallel static way in 5643 miliseconds
Calculated parallel static N(MATRIX_H/10) way in 5648 miliseconds
Calculated parallel dynamic N(MATRIX_H/10) way in 5690 miliseconds
Calculated parallel guided N(MATRIX_H/10) way in 5653 miliseconds
Calculated parallel runtime way in 5645 miliseconds
```

```
Calculated parallel static way in 5647 miliseconds
Calculated parallel static N(MATRIX_H/10) way in 5649 miliseconds
Calculated parallel dynamic N(MATRIX_H/10) way in 5644 miliseconds
Calculated parallel guided N(MATRIX_H/10) way in 5640 miliseconds
Calculated parallel runtime way in 5643 miliseconds
```

```
Calculated parallel static way in 5636 miliseconds
Calculated parallel static N(MATRIX_H/10) way in 5662 miliseconds
Calculated parallel dynamic N(MATRIX_H/10) way in 5643 miliseconds
Calculated parallel guided N(MATRIX_H/10) way in 5639 miliseconds
Calculated parallel runtime way in 5639 miliseconds
```

```
Calculated parallel static way in 5640 miliseconds
Calculated parallel static N(MATRIX_H/10) way in 5685 miliseconds
Calculated parallel dynamic N(MATRIX_H/10) way in 5666 miliseconds
Calculated parallel guided N(MATRIX_H/10) way in 5644 miliseconds
Calculated parallel runtime way in 5645 miliseconds
```

średnia static: 5656,66666667ms
średnia dynamic:5654 ms
średnia guided: 5641,166666667ms

Średnio najkrócej wykonywał się tryb dynamic

4.1.Usuń dyrektywę reduction i porównaj wyniki. Dlaczego wyniki się różnią?
Przed usunięciem reduction:

```
#pragma omp parallel for shared(matrix) private(i, k) reduction(+ : sum)
    for (i = 0; i < MATRIX_H; i++) {
        for (k = 0; k < MATRIX_W; k++) {
            sum = sum + matrix[i][k];
        }
    }

    return sum;
}
```

```
Sum calculated normal way: 449970518 in time: 146 ms
Sum calculated parralel way: 449970518 in time: 144 ms
```

Po usunięciu reduction:

```
#pragma omp parallel for shared(matrix) private(i, k)
    for (i = 0; i < MATRIX_H; i++) {
        for (k = 0; k < MATRIX_W; k++) {
            sum = sum + matrix[i][k];
        }
    }

    return sum;
}
```

```
Konsola debugowania programu Microsoft Visual Studio
Sum calculated normal way: 449921302 in time: 143 ms
Sum calculated parralel way: 104879500 in time: 362 ms

C:\Users\student\source\repos\4,0\Debug\4,0.exe (proces 39064) 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...
```

Uzyskane wyniki różnią się, ze względu na brak sumowania kopii stworzonej dla każdego z wątków osobno zmiennej sum na koniec działania programu

4.2. Napisz funkcje tworzącą wektor jednowymiarowy o wielkości 10.000 elementów. Uzupełnij go RAND w zakresie 0-10, a następnie oblicz długość wektora. Porównaj i zapisz wyniki z wykorzystaniem zrównoleglenia oraz bez

```
uint32_t WektorLen()
{
    int32_t length = 0;
    for (int i = 0; i < MATRIX ;i++)
    {
        length += wektor[i];
    }
    return length;
}
uint32_t WektorLenpar()
{
    int32_t length = 0;
    int i;
#pragma omp parallel for shared(matrix) private(i) reduction(+:length)
    for (i = 0; i < MATRIX; i++)
    {
        length += wektor[i];
    }
    return length;
}
```

```
Konsola debugowania programu Microsoft Visual Studio
Sum calculated normal way: 449995947 in time: 143 ms
Sum calculated parralel way: 82992811 in time: 293 ms
Wektor normalnie: 49739 in time: 0 ms
Wektor parralel: 49739 in time: 0 ms
```

Sumowanie jednowymiarowego wektora, wykonuje się tak samo szybko z wykorzystaniem zrównoleglenia jak i bez jego zastosowania.