

Wydział Nauk Inżynieryjnych ANS w Nowym Sączu		
Metody numeryczne – laboratorium		
Temat: P10		
Nazwisko i imię: Dominik Żuchowicz	Ocena sprawozdania	Zaliczenie:
Data wykonania ćwiczenia: 06.05.2025	Grupa: P3	

## Wprowadzenie

Celem niniejszego laboratorium było zapoznanie się z wybranymi dyrektywami oraz mechanizmami programowania równoległego w środowisku OpenMP. W trakcie ćwiczeń przeanalizowano działanie dyrektyw takich jak `if`, `num_threads`, `barrier`, `section` oraz `nowait`, które umożliwiają efektywne zarządzanie współbieżnością oraz synchronizacją wątków. Poznane mechanizmy pozwalają na optymalizację wykonywania obliczeń poprzez dynamiczne dostosowywanie liczby wątków, synchronizację punktów wykonania oraz podział zadań pomiędzy wątki. W ramach zadań praktycznych przeprowadzono testy wydajnościowe oraz analizę wpływu poszczególnych dyrektyw na czas wykonania programów, co pozwoliło na lepsze zrozumienie zasad działania programów równoległych oraz potencjalnych korzyści płynących z ich stosowania.

## 1 Zadania

### 1.1 PWIR\_10\_01.cpp

#### 1.1.1 Zadanie 1

Dopisz dyrektywę `num_threads` do programu z `PWIR_08_00.cpp`. Przetestuj czas wykonywania programu dla dwóch i więcej wątków.

```

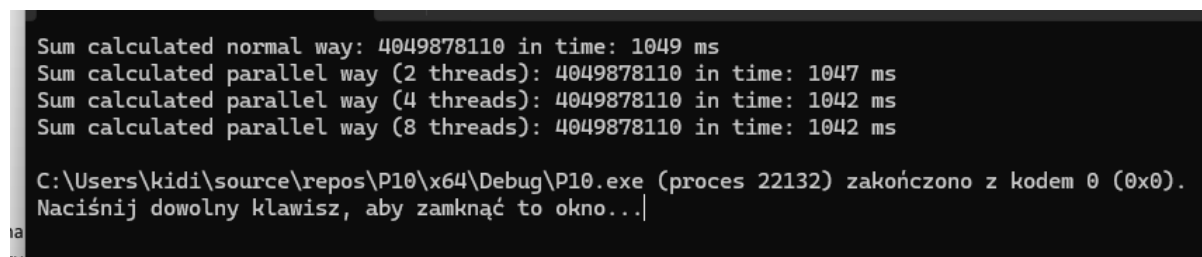
1  #include <stdio>
2  #include <stdint>
3  #include <stdlib>
4  #include <chrono>
5  #include <assert.h>
6  #include <omp.h>
7  #include <time.h>
8
9  #define MATRIX_H 30000
10 #define MATRIX_W 30000
11
12 //operators
13 //+
14 //-
15 //*
16 //&
17 //|
18 //~
19 //&&
20 //||
21
22 uint8_t** matrix;
23
24 uint32_t sumMatrix(int p, int threads) {
25     uint32_t sum = 0;
26     int32_t i;
27     int32_t k;
28
29     #pragma omp parallel for \
30         if (p) num_threads(threads) \
31         shared(matrix) private(k, i) \
32         reduction(+ : sum)
33     for (uint32_t i = 0; i < MATRIX_H; i++) {
34         for (uint32_t k = 0; k < MATRIX_W; k++) {
35             sum = sum + matrix[i][k];
36         }
37     }
38 }
```

```

37     }
38
39     return sum;
40 }
41
42 int main() {
43     srand(time(NULL));
44
45     //alloc matrix
46     matrix = (uint8_t**)new uint8_t * [MATRIX_H];
47     for (uint32_t i = 0; i < MATRIX_H; i++)
48         matrix[i] = new uint8_t[MATRIX_W];
49
50     //fill matrix random data normal way
51     for (uint32_t i = 0; i < MATRIX_H; i++) {
52         for (uint32_t k = 0; k < MATRIX_W; k++) {
53             matrix[i][k] = (uint16_t)(rand() % 10);
54         }
55     }
56
57     auto start = std::chrono::high_resolution_clock::now();
58     uint32_t sum = sumMatrix(0, 0);
59     auto end = std::chrono::high_resolution_clock::now();
60
61     printf("Sum calculated normal way: %u in time: %llu ms\r\n", sum,
62           std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());
63
64     // Test dla różnych liczby wątków
65     for (int threads : {2, 4, 8}) {
66         start = std::chrono::high_resolution_clock::now();
67         sum = sumMatrix(1, threads);
68         end = std::chrono::high_resolution_clock::now();
69
70         printf("Sum calculated parallel way (%d threads): %u in time: %llu ms\r\n",
71               threads, sum,
72               std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());
73     }
74
75     for (uint32_t i = 0; i < MATRIX_H; i++) delete[] matrix[i];
76     delete[] matrix;
77
78     return 0;
79 }

```

Listing 1: Zadanie 1



```

Sum calculated normal way: 4049878110 in time: 1049 ms
Sum calculated parallel way (2 threads): 4049878110 in time: 1047 ms
Sum calculated parallel way (4 threads): 4049878110 in time: 1042 ms
Sum calculated parallel way (8 threads): 4049878110 in time: 1042 ms

C:\Users\kidi\source\repos\P10\x64\Debug\P10.exe (proces 22132) zakończono z kodem 0 (0x0).
Naciśnij dowolny klawisz, aby zamknąć to okno...

```

Rysunek 1: Zrzut ekranu 2025-05-06 102647

## 1.2 PWIR\_10\_03.cpp

### 1.2.1 Zadanie 1

Przetestuj działanie PWIR\_08\_00 z klauzulą `nowait` oraz `bez`. Sprawdź również działanie na większej ilości wątków.

```
1 #include <cstdio>
2 #include <cstdlib>
3 #include <stdlib.h>
4 #include <chrono>
5 #include <assert.h>
6 #include <windows.h>
7 #include <omp.h>
8
9 void DoSomethingFast() {
10     Sleep(1000);
11 }
12
13 void DoSomethingLong() {
14     Sleep(6000);
15 }
16
17
18 int main() {
19     uint8_t id;
20
21     auto start = std::chrono::high_resolution_clock::now();
22     DoSomethingFast();
23     auto end = std::chrono::high_resolution_clock::now();
24
25     printf("Fast in time: %llu ms\r\n",
26         std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());
27
28     start = std::chrono::high_resolution_clock::now();
29     DoSomethingLong();
30     end = std::chrono::high_resolution_clock::now();
31
32     printf("Long in time: %llu ms\r\n",
33         std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());
34
35     int thread_counts[] = {2, 4, 8};
36     for (int t = 0; t < 3; ++t) {
37         int threads = thread_counts[t];
38         start = std::chrono::high_resolution_clock::now();
39 #pragma omp parallel num_threads(threads) private(id)
40         {
41             id = omp_get_thread_num();
42             if (id % 2) {
43                 DoSomethingLong();
44             } else {
45                 DoSomethingFast();
46             }
47             printf("Thread %d done work and wait on barrier\n", id);
48 #pragma omp barrier
49             printf("Thread %d done work and already finished\n", id);
50         }
51         end = std::chrono::high_resolution_clock::now();
52         printf("Parallel normal way (%d threads): %llu ms\r\n",
53             threads,
54             std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());
55     }
56
57     return 0;
58 }
```

Listing 2: Zadanie1

```
Konsola debugowania progra X + v

Sections - Thread 0 working...
Sections - Thread 0 working...
Iteration 0 execute thread 0.
Iteration 1 execute thread 0.
Iteration 2 execute thread 0.
Iteration 3 execute thread 0.
Iteration 4 execute thread 0.
Iteration 5 execute thread 0.
Iteration 6 execute thread 0.
Iteration 7 execute thread 0.
Iteration 8 execute thread 0.
Iteration 9 execute thread 0.
Parallel normal way 10074 ms
[nowait] Sections - Thread 0 working...
[nowait] Sections - Thread 0 working...
[nowait] Iteration 0 execute thread 0.
[nowait] Iteration 1 execute thread 0.
[nowait] Iteration 2 execute thread 0.
[nowait] Iteration 3 execute thread 0.
[nowait] Iteration 4 execute thread 0.
[nowait] Iteration 5 execute thread 0.
[nowait] Iteration 6 execute thread 0.
[nowait] Iteration 7 execute thread 0.
[nowait] Iteration 8 execute thread 0.
[nowait] Iteration 9 execute thread 0.
Parallel (nowait, 2 threads): 10090 ms
```

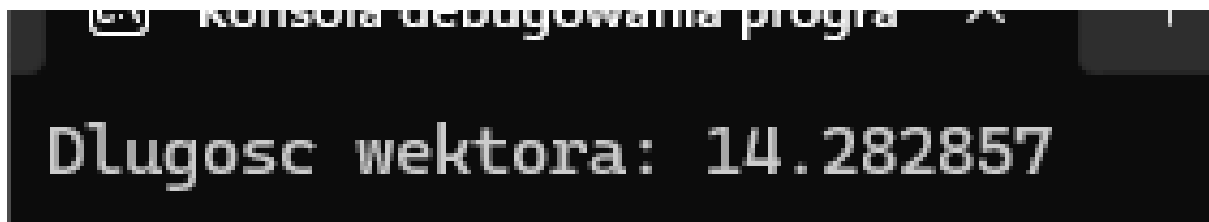
Rysunek 2: Zrzut ekranu 2025-05-06 103617

### 1.2.2 Zadanie 2

Napisz program liczący długość wektora na czterech wątkach, używając sekcji.

```
1 #include <stdio>
2 #include <cmath>
3 #include <omp.h>
4
5 int main() {
6     const int N = 8;
7     double v[N] = {1, 2, 3, 4, 5, 6, 7, 8};
8     double sum[4] = {0, 0, 0, 0};
9
10    #pragma omp parallel sections num_threads(4)
11    {
12        #pragma omp section
13        for (int i = 0; i < 2; ++i) sum[0] += v[i] * v[i];
14        #pragma omp section
15        for (int i = 2; i < 4; ++i) sum[1] += v[i] * v[i];
16        #pragma omp section
17        for (int i = 4; i < 6; ++i) sum[2] += v[i] * v[i];
18        #pragma omp section
19        for (int i = 6; i < 8; ++i) sum[3] += v[i] * v[i];
20    }
21    double total = sum[0] + sum[1] + sum[2] + sum[3];
22    printf("Dlugosc wektora: %f\n", sqrt(total));
23    return 0;
24 }
```

Listing 3: z4



Rysunek 3: Zrzut ekranu 2025-05-06 103711

## 2 Wnioski

Podczas realizacji ćwiczenia zapoznałem się z zaawansowanymi dyrektywami OpenMP, takimi jak `if`, `num_threads`, `barrier`, `section` oraz `nowait`. Zastosowanie dyrektywy `if` pozwala na dynamiczne decydowanie o równoległym wykonaniu kodu w zależności od rozmiaru danych, co może poprawić efektywność programu. Dyrektywa `num_threads` umożliwia kontrolę liczby wątków, co pozwala na optymalizację wykorzystania zasobów sprzętowych. Dyrektywa `barrier` zapewnia synchronizację wątków, co jest istotne przy współdzieleniu danych. Konstrukcja `sections` umożliwia podział pracy na niezależne zadania, które mogą być wykonywane równoległe przez różne wątki. Klauzula `nowait` pozwala na dalsze wykonywanie kodu bez oczekiwania na pozostałe wątki, co może skrócić czas wykonania programu, ale wymaga ostrożności przy dostępie do współdzielonych danych. Przeprowadzone testy wykazały, że odpowiednie wykorzystanie tych dyrektyw pozwala na zwiększenie wydajności programów równoległych, jednak wymaga świadomego zarządzania synchronizacją i podziałem pracy.