

Akademia Nauk Stosowanych w Nowym Sączu Programowanie Współbieżne i Rozproszone			
Temat: Pomiar czasu oraz zrównoleglenie pętli.			spr nr 8
Nazwisko i imię: Ciapała Tadeusz		Ocena sprawozdania	Zaliczenie:
Data wykonania ćwiczenia: 18.04.2023		Grupa: P3	

1) KOD PWIR_02_01:

Na tych zajęciach zajmowaliśmy się zrównolegleniem operacji przy pomocy biblioteki Open MP. Do tego posłużyliśmy się dyrektywą OpenMP - #pragma omp parallel, po której podajemy pętlę, zmienne wspólne oraz zmienne prywatne. Do zmiennych wspólnych dostęp ma każdy tworzony przez tę metodę wątek, natomiast do zmiennych prywatnych dostęp ma tylko ten wątek, który wykonuje na nich operacje.

```
#include <cstdio>
#include <cstdlib>
#include <stdlib.h>
#include <chrono>
#include <assert.h>
#include <time.h>

//matrix * vector

#define MATRIX_H 30000
#define MATRIX_W 30000
#define VECTOR_S 30000

uint16_t** matrix;
uint16_t* vector;
uint16_t* result;

int32_t i;
int32_t k;

int main() {
    srand(time(NULL));

    //check if vector size == matrix width
    assert(MATRIX_W == VECTOR_S);

    //alloc matrix
    matrix = (uint16_t**)new uint16_t * [MATRIX_H];
    for (i = 0; i < MATRIX_H; i++)
        matrix[i] = new uint16_t[MATRIX_W];

    //alloc vectors
    vector = (uint16_t*)new uint16_t[VECTOR_S];
    result = (uint16_t*)new uint16_t[VECTOR_S];

    //fill matrix random data normal way
    auto start = std::chrono::high_resolution_clock::now();
    for (i = 0; i < MATRIX_H; i++) {
        for (k = 0; k < MATRIX_W; k++) {
            matrix[i][k] = (uint16_t)(rand() % 100);
        }
    }

    //fill vector random data
    for (i = 0; i < VECTOR_S; i++) {
        vector[i] = (uint16_t)(rand() % 100);
    }
}
```

```

        result[i] = 0;
    }

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

    printf("Fill in %llu milliseconds\n",
        std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());

    //normal execution
    start = std::chrono::high_resolution_clock::now();
    for (i = 0; i < MATRIX_H; i++) {
        for (k = 0; k < MATRIX_W; k++) {
            result[i] += matrix[i][k] * vector[k];
        }
    }
    end = std::chrono::high_resolution_clock::now();

    printf("Calculated normal way in %llu milliseconds\n",
        std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());

    start = std::chrono::high_resolution_clock::now();
#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++) {
        vector[i] = (uint16_t)(rand() % 100);
    }
    end = std::chrono::high_resolution_clock::now();

    printf("Fill parallel way in %lu milliseconds\n",
        std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());

    start = std::chrono::high_resolution_clock::now();
#pragma omp parallel for num_threads(4) shared(matrix, vector, result) private(i, k)
    for (i = 0; i < MATRIX_H; i++) {
        for (k = 0; k < MATRIX_W; k++) {
            result[i] += matrix[i][k] * vector[k];
        }
    }
    end = std::chrono::high_resolution_clock::now();

    printf("Calculated parallel way in %lu milliseconds\n",
        std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());

    //free memory
    delete[] vector;
    delete[] result;

    for (i = 0; i < MATRIX_H; i++)
        delete[] matrix[i];

    delete[] matrix;

    return 0;
}

```

Listing nr 1: kod dostarczony do wykonania zadania

Wyżej pokazany kod wykonuje mnożenie macierzy przez wektor. Rozmiary macierzy definiowane są za pomocą dyrektywy `#define` - mamy zdefiniowane dwie stałe w programie, które odpowiadają za wysokość i szerokość macierzy (w uruchomionym przykładzie jest to 30000 na 30000). Rozmiar wektora jest zdefiniowany w taki sam sposób.

```
matrix[i][k] = (uint16_t)(rand() % 100);
}

Konsola debugowania programu Microsoft Visual Studio
Fill in 68898 milliseconds
Calculated normal way in 2966 milliseconds
Fill parallel way in 69610 milliseconds
Calculated parallel way in 2341 milliseconds

nd = C:\Users\student\AppData\LocalLow\TC\P3_PWIR_TC\repos\ConsoleApplication3\Debug\ConsoleApplication3.exe (proces 13716) zakończył działanie. Kod: 0.
printf Naciśnij dowolny klawisz, aby zamknąć to okno...
scanf %c;
system("pause");
return 0;
}
```

Zrzut nr 1: pierwsze uruchomienie kodu

Po uruchomieniu na pierwszy rzut oka można zauważyć pewne różnice w czasie wykonania zadanych operacji, jednak wypełnienie macierzy sposobem zrównoleglonym jest podejrzanie dłuższe pod względem czasu niż normalną metodą. Mimo, że wypisało nam czas obliczeń niższy przy zrównolegleniu, to jednak wniosek jest taki, że operacja ta nie została zrównoleglona. Aby umożliwić pełne działanie Open MP należy wejść w ustawienia rozwiązania, wybrać zakładkę dotyczącą naszego języka (C/C++) i tam w podzakładce "Język" trzeba włączyć obsługę biblioteki OpenMP. Tylko wtedy mamy gwarancję poprawnego działania tej metody zrównoleglania obliczeń.

Kod do wykonania zadania:

```
#include <cstdio>
#include <cstdint>
#include <cstdlib>
#include <chrono>
#include <assert.h>
#include <time.h>

//matrix * vector

#define MATRIX_H 5000
#define MATRIX_W 5000
#define VECTOR_S 5000
#define MUL_VAR 1000

uint16_t** matrix;
uint16_t* vector;
uint16_t* result;

int32_t i;
int32_t k;
int32_t j;

int main() {
    srand(time(NULL));

    //check if vector size == matrix width
    assert(MATRIX_W == VECTOR_S);

    //alloc matrix
    matrix = (uint16_t**)new uint16_t * [MATRIX_H];
    for (i = 0; i < MATRIX_H; i++)
        matrix[i] = new uint16_t[MATRIX_W];

    //alloc vectors
    vector = (uint16_t*)new uint16_t[VECTOR_S];
    result = (uint16_t*)new uint16_t[VECTOR_S];

    //fill matrix random data normal way
    auto start = std::chrono::high_resolution_clock::now();
    for (i = 0; i < MATRIX_H; i++) {
        for (k = 0; k < MATRIX_W; k++) {
            matrix[i][k] = (uint16_t)(rand() % 100);
        }
    }

    //fill vector random data
    for (i = 0; i < VECTOR_S; i++) {
        vector[i] = (uint16_t)(rand() % 100);
        result[i] = 0;
    }

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

    printf("Fill in %llu miliseconds\n",
        std::chrono::duration_cast<std::chrono::milliseconds>(end -
start).count());

    //normal execution
    start = std::chrono::high_resolution_clock::now();
    for (i = 0; i < MATRIX_H; i++) {
        for (k = 0; k < MATRIX_W; k++) {
            result[i] += matrix[i][k] * vector[k];
        }
    }
    end = std::chrono::high_resolution_clock::now();

    printf("Calculated normal way in %llu miliseconds\n",
        std::chrono::duration_cast<std::chrono::milliseconds>(end -
start).count());

    start = std::chrono::high_resolution_clock::now();
#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++) {
        vector[i] = (uint16_t)(rand() % 100);
    }
    end = std::chrono::high_resolution_clock::now();

    printf("Fill parallel way in %lu milliseconds\n",
        std::chrono::duration_cast<std::chrono::milliseconds>(end -
start).count());

    start = std::chrono::high_resolution_clock::now();
#pragma omp parallel for num_threads(4) shared(matrix, vector, result) private(i, k)
    for (i = 0; i < MATRIX_H; i++) {
        for (k = 0; k < MATRIX_W; k++) {
            result[i] += matrix[i][k] * vector[k];
        }
    }
    end = std::chrono::high_resolution_clock::now();
    printf("Calculated parallel way in %lu milliseconds\n",
        std::chrono::duration_cast<std::chrono::milliseconds>(end -
start).count());

//=====

    //reset result
    for (i = 0; i < VECTOR_S; i++) {
        result[i] = 0;
    }

    //wielokrotne mnozenie macierzy - jeden watek
    start = std::chrono::high_resolution_clock::now();
#pragma omp parallel for num_threads(1) shared(matrix, vector, result) private(i,k,j)
    for (j = 0; j < MUL_VAR; j++) {
        for (i = 0; i < MATRIX_H; i++) {
            for (k = 0; k < MATRIX_W; k++) {
                result[i] += matrix[i][k] * vector[k];
            }
        }
    }
    end = std::chrono::high_resolution_clock::now();
    printf("Zad - normal way in %llu milliseconds\n",
        std::chrono::duration_cast<std::chrono::milliseconds>(end -
start).count());

    //reset result
    for (i = 0; i < VECTOR_S; i++) {
        result[i] = 0;
    }

    //wielokrotne mnozenie macierzy - zrownoleglone
    start = std::chrono::high_resolution_clock::now();
#pragma omp parallel for num_threads(4) shared(matrix, vector, result) private(i, k, j)
    for (j = 0; j < MUL_VAR; j++) {
        for (i = 0; i < MATRIX_H; i++) {
            for (k = 0; k < MATRIX_W; k++) {
                result[i] += matrix[i][k] * vector[k];
            }
        }
    }

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

    printf("Zad - Calculated parallel way in %lu milliseconds\n",
        std::chrono::duration_cast<std::chrono::milliseconds>(end -
start).count());

```

```

//free memory
delete[] vector;
delete[] result;

for (i = 0; i < MATRIX_H; i++)
    delete[] matrix[i];

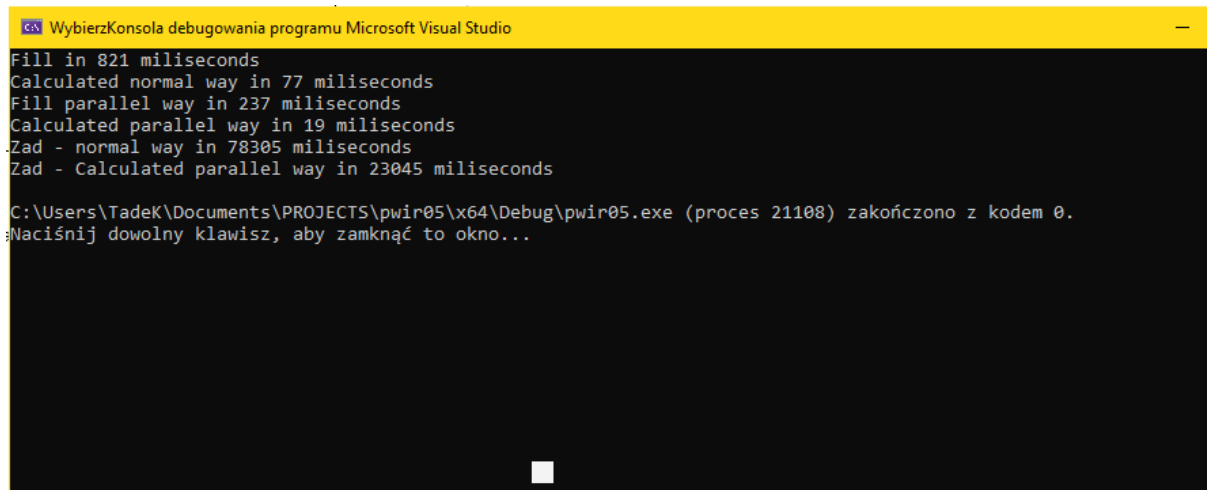
delete[] matrix;

return 0;
}

```

Listing nr 2: kod do wykonania zadania

Powyższy kod został uzupełniony o dodatkowe testy. Do tego obliczenia zostały powielone, to znaczy zagnieżdżone w innych pętlach. Ma to na celu spowodowanie sztucznego utrudnienia wykonywanych obliczeń a co za tym idzie uzyskania dłuższych pomiarów czasu bez zmiany wielkości macierzy i jej wartości.



```

WybierzKonsola debugowania programu Microsoft Visual Studio
Fill in 821 milliseconds
Calculated normal way in 77 milliseconds
Fill parallel way in 237 milliseconds
Calculated parallel way in 19 milliseconds
Zad - normal way in 78305 milliseconds
Zad - Calculated parallel way in 23045 milliseconds

C:\Users\TadeK\Documents\PROJECTS\pwir05\x64\Debug\pwir05.exe (proces 21108) zakończono z kodem 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...

```

Zrzut nr 2: uruchomienie programu z listingu nr 2 - wykonanie zadania

Gdy mamy ustawioną pełną obsługę Open MP i uruchomione testy, to tym razem widzimy znaczną różnicę czasową pomiędzy obliczeniami na jednym wątku a obliczeniami równoległymi. Wyliczony wynik dla OpenMP jest kilkukrotnie niższy pod względem zajętego czasu. W rozwiązaniu zadania obliczenia zostały podzielone na 4 wątki.