Łukasz Oprych Gr. Lab. 5 Informatyka Techniczna	T: OpenMP Zmienne	15.12.2023r.

Cel ćwiczenia:

Dalsze zapoznawanie się z pisaniem programów równoległych w OpenMP, zależnościami oraz wykorzystywaniem dyrektyw i klauzul.

Przebieg ćwiczenia:

Po przygotowaniu programu zgodnie z instrukcją prowadzącego wywołano domyślny program openmp_watki_zmienne dla ustawionych niedomyślnie 6 wątków:

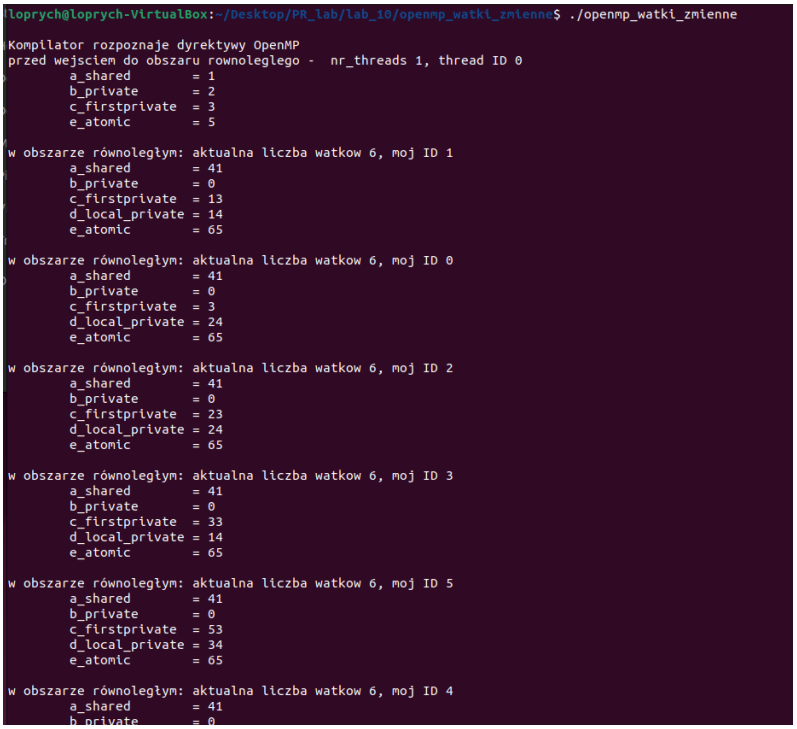
```
w obszarze równoległym: aktualna liczba watkow 6, moj ID 0
        a_shared        = 51
        b_private       = 0
        c_firstprivate  = 3
        d_local_private = 14
        e_atomic        = 155

w obszarze równoległym: aktualna liczba watkow 6, moj ID 3

w obszarze równoległym: aktualna liczba watkow 6, moj ID 1
        a_shared        = 51
        b_private       = 0
        c_firstprivate  = 13
        d_local_private = 34
        e_atomic        = 155
        a_shared        = 51
        b_private       = 0
        c_firstprivate  = 33
        d_local_private = 24
        e_atomic        = 155
po zakonczeniu obszaru rownoleglego:
        a_shared        = 51
        b_private       = 2
        c_firstprivate  = 3
        e_atomic        = 155
loprych@loprych-VirtualBox:~/Desktop/PR_lab/lab_10/openmp_watki_zmienne$
```

Jak widać wynik, zmienne dla wątku ID3 wypisane wypisały się w wątku ID1, co jest niepożądanym zjawiskiem.

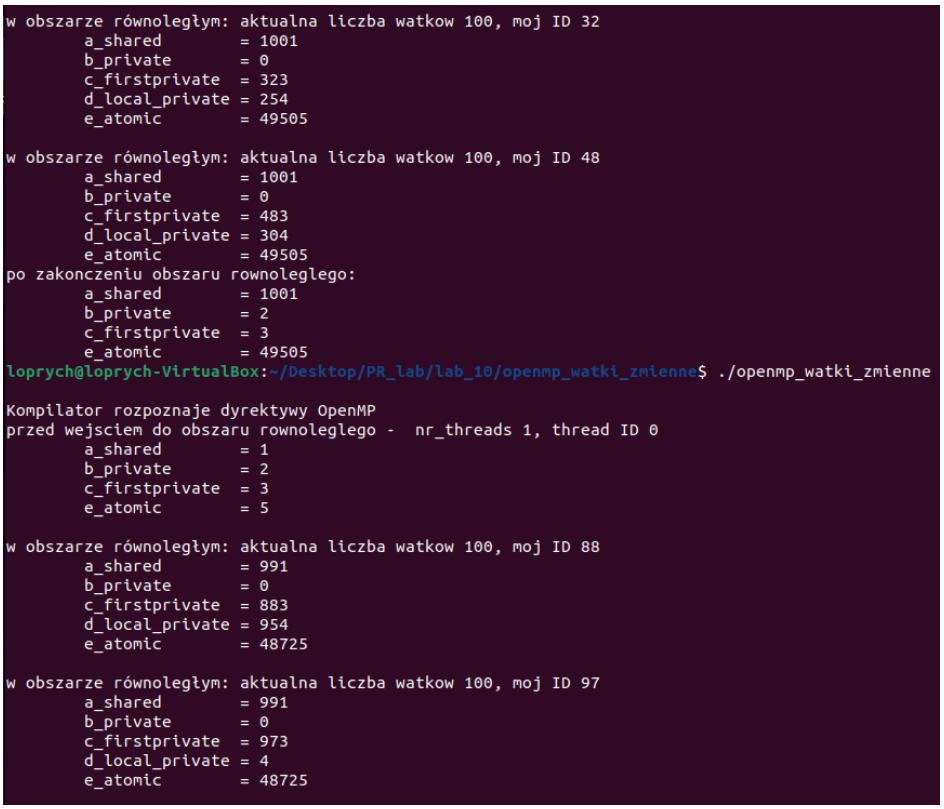
W celu poprawienia czytelności wydruku zastosowano dyrektywy bezpośrednio przed wypisaniem wyników poprzez dodanie bariery, aby przed wypisaniem wszystkie wątki wykonały swoje obliczenia, oraz dyrektywę critical, aby kod był wykonywany pojedynczo dla każdego wątku:

```
43      }
44      #pragma omp barrier
45      #pragma omp critical
46      {
47
48        printf("\nw obszarze równoległym: aktualna liczba watkow %d, moj ID %d\n",
49               omp_get_num_threads(), omp_get_thread_num());
50
```

Jak widać zmienne wyświetlane są już prawidłowo dla każdego wątku:



Kolejnym etapem było poprawienie wypisywania wartości zmiennych globalnych, w tym wypadku a_shared i e_atomic, ponieważ przy każdym wywołaniu zmienne zmieniały swoją wartość:



Dodano klauzulę critical dla zmiennej a_shared w celu ochrony sekcji krytycznej, czyli pętli z a_shared.

Dodano klazulę atomic w pętli ze zmienną e_atomic w celu zapewnienia poprawności działania przez użycie operacji atomowych.

Dodano klauzulę barrier, aby to co już zostało nadpisanie nie było zmienione przez inny wątek wykonujące.

```
33      #pragma omp barrier
34      #pragma omp critical(a_shared)
35      for(i=0;i<10;i++){
36        a_shared ++;
37      }
38
39      for(i=0;i<10;i++){
40        c_firstprivate += omp_get_thread_num();
41      }
42
43      for(i=0;i<10;i++){
44      #pragma omp atomic
45        e_atomic+=omp_get_thread_num();
46      }
47      #pragma omp barrier
48      #pragma omp critical
49      {
```

W 33 linii za pomocą bariery eliminujemy zależność WAR

W 36 linii zależność WAR ze względu na zmienną a_shared

W 45 linii zależność RAW i WAR ze względu na e_atomic

```
34      d_local_private = a_shared + c_firstprivate;
35      f_threadprivate = omp_get_thread_num();
```

W linii 34 zależność RAW i WAR ze względu na a_shared

```
55      printf("\nw obszarze równoległym: aktualna liczba watkow %d, moj ID %d\n",
56             omp_get_num_threads(), omp_get_thread_num());
57
58      printf("\ta_shared \t= %d\n", a_shared);
59      printf("\tb_private \t= %d\n", b_private);
60      printf("\tc_firstprivate \t= %d\n", c_firstprivate);
61      printf("\td_local_private = %d\n", d_local_private);
62      printf("\te_atomic \t= %d\n", e_atomic);
63
64      printf("\tf_threadprivate = %d\n", f_threadprivate);
65    }
66  }
```

W 58 i 62 linii zależność RAW ze względu na zmienne a_shared i e_atomic

Jak widać wartości zmiennych współdzielonych nie zmieniają się po kolejnym wykonaniu programu:

```
w obszarze równoległym: aktualna liczba watkow 100, moj ID 36
        a_shared        = 1001
        b_private       = 0
        c_firstprivate  = 363
        d_local_private = 4
        e_atomic        = 49505

w obszarze równoległym: aktualna liczba watkow 100, moj ID 95
        a_shared        = 1001
        b_private       = 0
        c_firstprivate  = 953
        d_local_private = 4
        e_atomic        = 49505
po zakonczeniu obszaru rownoleglego:
        a_shared        = 1001
        b_private       = 2
        c_firstprivate  = 3
        e_atomic        = 49505
loprych@loprych-VirtualBox:~/Desktop/PR_lab/lab_10/openmp_watki_zmienne$ ./openmp_watki_zmienne

Kompilator rozpoznaje dyrektywy OpenMP
przed wejsciem do obszaru rownoleglego -  nr_threads 1, thread ID 0
        a_shared        = 1
        b_private       = 2
        c_firstprivate  = 3
        e_atomic        = 5

w obszarze równoległym: aktualna liczba watkow 100, moj ID 61
        a_shared        = 1001
        b_private       = 0
        c_firstprivate  = 613
        d_local_private = 4
        e_atomic        = 49505

w obszarze równoległym: aktualna liczba watkow 100, moj ID 11
        a_shared        = 1001
        b_private       = 0
        c_firstprivate  = 113
        d_local_private = 4
        e_atomic        = 49505
```

Kolejnym poleceniem było utworzenie kolejnego obszaru równoległego oraz przetestowanie w nim dyrektywy threadprivate poprzez utworzenie zmiennej f_threadprivate oraz dodanie jej w pierwszym obszarze równoległym, która ma zachować swoją wartość w obu obszarach równoległych.

Definicja zmiennej:

```
5 int f_threadprivate;
6 #pragma omp threadprivate(f_threadprivate)
```

Umieszczenie zmiennej w pierwszym obszarze równoległym:

```
34      d_local_private = a_shared + c_firstprivate;
35      f_threadprivate = omp_get_thread_num();
```

Zależność RAW i WAR w linii 34 ze względu na zmienną a_shared.

Utworzenie drugiego obszaru równoległego, w którym zostaje wypisana wartość zmiennej f_threadprivate:

```
#pragma omp parallel default(none)
{
	#pragma omp critical
	{
	printf("\nw drugim obszarze równoległym: aktualna liczba watkow %d, moj ID %d\n",
	omp_get_num_threads(), omp_get_thread_num());

	printf("\tf_threadprivate = %d\n", f_threadprivate);
	}
}
```

Wynik dla 5 wątków zdefiniowanych za pomocą funkcji bibliotecznej omp_set_num_threads(5):

```
w obszarze równoległym: aktualna liczba watkow 5, moj ID 2
        a_shared        = 51
        b_private       = 0
        c_firstprivate  = 23
        d_local_private = 4
        e_atomic        = 105
        f_threadprivate = 2

w obszarze równoległym: aktualna liczba watkow 5, moj ID 4
        a_shared        = 51
        b_private       = 0
        c_firstprivate  = 43
        d_local_private = 4
        e_atomic        = 105
        f_threadprivate = 4

w obszarze równoległym: aktualna liczba watkow 5, moj ID 0
        a_shared        = 51
        b_private       = 0
        c_firstprivate  = 3
        d_local_private = 4
        e_atomic        = 105
        f_threadprivate = 0

w obszarze równoległym: aktualna liczba watkow 5, moj ID 1
        a_shared        = 51
        b_private       = 0
        c_firstprivate  = 13
        d_local_private = 4
        e_atomic        = 105
        f_threadprivate = 1

w obszarze równoległym: aktualna liczba watkow 5, moj ID 3
        a_shared        = 51
        b_private       = 0
        c_firstprivate  = 33
        d_local_private = 4
        e_atomic        = 105
        f_threadprivate = 3
```

```
w drugim obszarze równoległym: aktualna liczba watkow 5, moj ID 2
        f_threadprivate = 2

w drugim obszarze równoległym: aktualna liczba watkow 5, moj ID 1
        f_threadprivate = 1

w drugim obszarze równoległym: aktualna liczba watkow 5, moj ID 0
        f_threadprivate = 0

w drugim obszarze równoległym: aktualna liczba watkow 5, moj ID 4
        f_threadprivate = 4

w drugim obszarze równoległym: aktualna liczba watkow 5, moj ID 3
        f_threadprivate = 3
```

Jak widać zmienne threadprivate zachowały swoje wartości dla poszczególnych wątków w drugim obszarze równoległym.

Następnie skonfigurowano program openmp_zaleznosci i uruchomiono.

```
loprych@loprych-VirtualBox:~/Downloads$ gcc openmp_zaleznosci.c -fopenmp -lm -o
openmp_zaleznosci
loprych@loprych-VirtualBox:~/Downloads$ ./openmp_zaleznosci
suma 1459701.114868, czas obliczen 0.009812
suma 500001.500001, czas obliczen rownoleglych 0.000001
loprych@loprych-VirtualBox:~/Downloads$
```

Jak widać obliczenia sekwencyjne i równoległe różnią się wynikiem.

Analiza:

W wersji sekwencyjnej programu nie ma możliwości zmiany wartości elementów kolejnych zanim nie policzymy poprzednich.

W równoległej wersji jeżeli weźmiemy pod uwagę pierwsze 2 watki, to zakładając, ze pierwszy watek dostanie elementy 0 i 1 tablicy, drugi wątek dostanie elementy 2 i 3 tablicy. Problem pojawia się, gdy drugi watek wykonuje swoje instrukcje jako pierwszy, ponieważ obliczy on z podanego wzoru wartości 2 i 3 elementu w tablicy A, a dopiero po nim wątek pierwszy będzie liczył 0 i 1 element w tablicy, wówczas w oparciu o zmienione wartości tablicy A. W linii 37 zachodzi zależność WAR, A[i] odczytuje dane, do zapisu danych dochodzi w całej operacji A[i] += A[i+2] + sin(B[i]).

Wersja równoległa:

```
35  #pragma omp parallel for default(none) shared(A
36  for(i=0; i<N; i++){
37    A[i] += A[i+2] + sin(B[i]);
38  }
39  t1 = omp_get_wtime() - t1;
```

Modyfikacja:

Dodanie tablicy C

```
double* C = malloc((N+2)*sizeof(double));
```

Uzupełnienie wersji równoległej:

```
31  for(i=0;i<N+2;i++) C[i] = (double)i/N;
32  t1 = omp_get_wtime();
33
34  // wersja równoległa
35  #pragma omp parallel for default(none) shared(A, B, C) num_threads(2)
36  for(i=0; i<N; i++){
37    A[i] += C[i+2] + sin(B[i]);
38  }
39  t1 = omp_get_wtime() - t1;
40
41  suma = 0.0;
42  for(i=0;i<N+2;i++) suma+=A[i];
43  printf("suma %lf, czas obliczen rownoleglych %lf\n", suma, t1);
44
45
46 }
```

Wnioski:

W powyższym ćwiczeniu zapoznano się z zależnościami danych RAW (Read After Write, zależność rzeczywista) oraz WAR (Write after Read, antyzależność). W RAW przykładowo, gdy mamy 2 instrukcje, instrukcja 2 próbuje odczytać dane zanim instrukcja 1 zdąży zapisać dane. Zależności rzeczywiste uniemożliwiają nam zrównoleglenie algorytmu, w celu poprawienia rozwiązania należy zmodyfikować kod, aby uzyskać wersję pozwalającą na zrównoleglenie. W WAR ponownie na przykładzie 2 instrukcji, instrukcja 2 zapisuje dane zanim instrukcja 1 zanim zdąży odczytać dane.W przypadku antyzależności rozwiązaniem problemu będzie przemianowanie zmiennych, wykorzystanie dodatkowej zmiennej.