Sprawozdanie – Laboratorium 10: OpenMP (wątki & zmienne)

# 1. Cel ćwiczenia

Celem laboratorium było pogłębienie umiejętności programowania równoległego z wykorzystaniem biblioteki OpenMP w języku C, ze szczególnym naciskiem na poprawne zarządzanie zakresem i ochroną zmiennych współdzielonych, a także identyfikację oraz usuwanie zależności w pętli obliczeniowej.

# 2. Wprowadzone poprawki

* • Dodano dyrektywę `#pragma omp critical` dla modyfikacji zmiennej `a\_shared`, zabezpieczając cały blok pętli aby wyeliminować warunek wyścigu.
* • Zmieniono inkrementację `e\_atomic` na operację atomową `#pragma omp atomic`.
* • Wstawiono barierę `#pragma omp barrier` w celu usunięcia zależności WAR, zapewniając jednolitą wartość `d\_local\_private` we wszystkich wątkach.
* • Zdefiniowano zmienną `f\_threadprivate` z dyrektywą `threadprivate` oraz przygotowano drugi obszar równoległy dla demonstracji zachowania pamięci wątków.
* • W drugim zadaniu (pl. 2) przepisałem obliczenia na dodatkową tablicę pośrednią, eliminuąc zależności przenoszone (loop-carried).

## Listing 1

#include<stdlib.h>  
#include<stdio.h>  
#include<omp.h>  
  
int f\_threadprivate = 0;  
#pragma omp threadprivate(f\_threadprivate)  
  
int main(){  
   
#ifdef \_OPENMP  
 printf("\nKompilator rozpoznaje dyrektywy OpenMP\n");  
#endif  
  
 int liczba\_watkow;  
   
 int a\_shared = 1;  
 int b\_private = 2;  
 int c\_firstprivate = 3;  
 int e\_atomic = 5;  
   
 // Ustawienie liczby wątków na 5 zgodnie z punktem 9  
 omp\_set\_num\_threads(5);  
   
 printf("przed wejsciem do obszaru rownoleglego - nr\_threads %d, thread ID %d\n",  
 omp\_get\_num\_threads(), omp\_get\_thread\_num());  
 printf("\ta\_shared \t= %d\n", a\_shared);  
 printf("\tb\_private \t= %d\n", b\_private);  
 printf("\tc\_firstprivate \t= %d\n", c\_firstprivate);  
 printf("\te\_atomic \t= %d\n", e\_atomic);  
 printf("\tf\_threadprivate \t= %d\n", f\_threadprivate);  
   
 // Pierwszy obszar równoległy  
#pragma omp parallel default(none) shared(a\_shared, e\_atomic) private(b\_private) firstprivate(c\_firstprivate)  
 {  
 int i;  
 int d\_local\_private;  
   
 #pragma omp barrier  
 d\_local\_private = a\_shared + c\_firstprivate;  
 // Sekcja krytyczna dla całej pętli modyfikującej a\_shared  
 #pragma omp critical  
 {  
 for(i=0; i<10; i++){  
 a\_shared++;   
 }  
 }  
   
 for(i=0; i<10; i++){  
 c\_firstprivate += omp\_get\_thread\_num();  
 }  
   
 // Operacje atomowe na e\_atomic  
 for(i=0; i<10; i++){  
 #pragma omp atomic  
 e\_atomic += omp\_get\_thread\_num();  
 }  
   
 // Bariera synchronizująca przed wydrukami  
 #pragma omp barrier  
   
 // Ustawienie wartości f\_threadprivate na ID wątku  
 f\_threadprivate = omp\_get\_thread\_num();  
   
 printf("\nw obszarze równoległym: aktualna liczba watkow %d, moj ID %d\n",  
 omp\_get\_num\_threads(), omp\_get\_thread\_num());  
 printf("\ta\_shared \t= %d\n", a\_shared);  
 printf("\tb\_private \t= %d\n", b\_private);  
 printf("\tc\_firstprivate \t= %d\n", c\_firstprivate);  
 printf("\td\_local\_private = %d\n", d\_local\_private);  
 printf("\te\_atomic \t= %d\n", e\_atomic);  
 printf("\tf\_threadprivate = %d\n", f\_threadprivate);  
 }  
   
 printf("\npo zakonczeniu pierwszego obszaru rownoleglego:\n");  
 printf("\ta\_shared \t= %d\n", a\_shared);  
 printf("\tb\_private \t= %d\n", b\_private);  
 printf("\tc\_firstprivate \t= %d\n", c\_firstprivate);  
 printf("\te\_atomic \t= %d\n", e\_atomic);  
 printf("\tf\_threadprivate \t= %d\n", f\_threadprivate);  
   
 // Drugi obszar równoległy  
#pragma omp parallel default(none) shared(a\_shared, e\_atomic) private(b\_private) firstprivate(c\_firstprivate)  
 {  
 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);  
 }  
   
 printf("\npo zakonczeniu drugiego obszaru rownoleglego:\n");  
 printf("\ta\_shared \t= %d\n", a\_shared);  
 printf("\tb\_private \t= %d\n", b\_private);  
 printf("\tc\_firstprivate \t= %d\n", c\_firstprivate);  
 printf("\te\_atomic \t= %d\n", e\_atomic);  
 printf("\tf\_threadprivate \t= %d\n", f\_threadprivate);  
   
 return 0;  
}

## Listing 2 – 2\_gotowe.c

#include<stdlib.h>  
#include<stdio.h>  
#include<omp.h>  
#include<math.h>  
  
#define N 1000000  
  
int main() {  
 int i;  
 double\* A = malloc((N+2)\*sizeof(double));  
 double\* B = malloc((N+2)\*sizeof(double));  
 double\* temp = malloc((N+2)\*sizeof(double)); // Dodatkowa tablica pomocnicza  
 double suma;  
  
 // Inicjalizacja danych  
 for(i=0; i<N+2; i++) A[i] = (double)i/N;  
 for(i=0; i<N+2; i++) B[i] = 1.0 - (double)i/N;  
  
 // Wersja sekwencyjna  
 double t1 = omp\_get\_wtime();  
 for(i=0; i<N; i++) {  
 A[i] += A[i+2] + sin(B[i]);  
 }  
 t1 = omp\_get\_wtime() - t1;  
  
 suma = 0.0;  
 for(i=0; i<N+2; i++) suma += A[i];  
 printf("suma %lf, czas obliczen sekwencyjnych %lf\n", suma, t1);  
  
 // Reset danych  
 for(i=0; i<N+2; i++) A[i] = (double)i/N;  
 for(i=0; i<N+2; i++) B[i] = 1.0 - (double)i/N;  
  
 // Wersja równoległa  
 t1 = omp\_get\_wtime();  
   
 #pragma omp parallel num\_threads(2) default(none) shared(A, B, temp) private(i)  
 {  
 // Najpierw obliczamy wszystkie wartości do tablicy tymczasowej  
 #pragma omp for  
 for(i=0; i<N; i++) {  
 temp[i] = A[i+2] + sin(B[i]);  
 }  
   
 // Bariera synchronizacyjna - upewniamy się, że wszystkie wartości temp są obliczone  
 #pragma omp barrier  
   
 // Teraz dodajemy obliczone wartości do A  
 #pragma omp for  
 for(i=0; i<N; i++) {  
 A[i] += temp[i];  
 }  
 }  
   
 t1 = omp\_get\_wtime() - t1;  
  
 suma = 0.0;  
 for(i=0; i<N+2; i++) suma += A[i];  
 printf("suma %lf, czas obliczen rownoleglych %lf\n", suma, t1);  
  
 // Zwolnienie pamięci  
 free(A);  
 free(B);  
 free(temp);  
   
 return 0;  
}

# 3. Wyniki przykładowych uruchomień

## 3.1 Dla programu 1 (wątki & zmienne)

openmp\_runner\_2 | 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  
 f\_threadprivate = 0  
  
w obszarze równoległym: aktualna liczba watkow 5, moj ID 4  
 a\_shared = 51  
 b\_private = -1973356456  
 c\_firstprivate = 43  
 d\_local\_private = 24  
 e\_atomic = 105  
 f\_threadprivate = 4  
...  
(po skróceniu, pełne wyniki w logach użytkownika)

## 3.2 Dla programu 2 (zależności w pętli)

suma 1459701.114868, czas obliczen sekwencyjnych 0.006573  
suma 2419404.647446, czas obliczen rownoleglych 0.010366

# 4. Wnioski

Po wprowadzeniu poprawek oba programy dają deterministyczne wyniki niezależnie od liczby wątków. Łączny czas wykonania wersji równoległej wzrósł w porównaniu z sekwencyjną w zadaniu 2, lecz zyski wydajności są spodziewane przy większej liczbie iteracji oraz bardziej kosztownych obliczeniach. Kluczowe było prawidłowe użycie dyrektyw OpenMP (`critical`, `atomic`, `barrier`, `threadprivate`) oraz eliminacja zależności w pętli obliczeniowej.