Łukasz Oprych Gr.5 ITE	Programowanie Równoległe	Laboratorium 5, Dekompozycja

Cel ćwiczenia:

Zapoznanie się z mechanizmami dekompozycji, blokowej oraz cyklicznej w programach równoległych.

Przebieg ćwiczenia:

Po skonfigurowaniu struktury katalogowej i plików źródłowych zgodnie z poleceniami prowadzącego, skompilowano plik pthreads_całka.c i dokonano pomiarów czasów obliczeń dla danej ilości wątków i rozmiaru danych w trzech wersjach: sekwencyjnej, ż użyciem mutexów, z użyciem tablicy.

Przykładowy wynik z konsoli:

```
loprych@loprych-VirtualBox:~/Desktop/PR_lab/lab_5/zad_1/pthreads_suma$ ./pthread
s_suma
Obliczenia sekwencyjne
suma = 50000000.500000
Czas obliczen sekwencyjnych = 0.159124
Poczatek tworzenia watkow
suma = 50000000.500000
Czas obliczen 1 wątków = 0.171452
Poczatek tworzenia watkow
suma = 50000000.500000
Czas obliczen 1 wątków (globalna tablica zamiast mutex'a) = 0.171428
loprych@loprych-VirtualBox:~/Desktop/PR_lab/lab_5/zad_1/pthreads_suma$
```

Czasy dla wersji z tablicą o rozmiarze 1000 i 100000000 i ilości wątków 1,2,4, wersja zoptymalizowana -O3 :

Rozmiar tablicy = 1000, obliczenia sekwencyjne	
	czas obliczeń sekwencyjnych
t[sekundy]	0,000002

Rozmiar tablicy = 100000000,obliczenia sekwencyjne	
	czas obliczeń sekwencyjnych
t[sekundy]	0,159124

Rozmiar tablicy = 1000, liczba wątków = 2, obliczenia równoległe		
	czas mutexy	czas tablicą
t[sekundy]	0,000649	0,000228

Rozmiar tablicy = 100000000, liczba wątków = 2, obliczenia równoległe		
	czas mutexy	czas tablicą
t[sekundy]	0,84475	0,077588

Rozmiar tablicy = 1000, liczba wątków = 4, obliczenia równoległe		
	czas mutexy	czas tablicą
t[sekundy]	0,000810	0,000172

Rozmiar tablicy = 100000000, liczba wątków = 4, obliczenia równoległe		
	czas mutexy	czas tablicą
t[sekundy]	0,075216	0,083931

Czasy dla wersji z tablicą o rozmiarze 1000 i 100000000 i ilości wątków 1,2,4, wersja niezoptymalizowana -g ddebug :

Rozmiar tablicy = 1000,obliczenia sekwencyjne	
	czas obliczeń sekwencyjnych
t[sekundy]	0,000004

Rozmiar tablicy = 100000000, obliczenia sekwencyjne	
	czas obliczeń sekwencyjnych
t[sekundy]	0,355895

Rozmiar tablicy = 1000, liczba wątków = 2, obliczenia równoległe		
	czas mutexy	t z tablicą
t[sekundy]	0,001663	0,000694

Rozmiar tablicy = 100000000, liczba wątków = 2, obliczenia równoległe		
	czas mutexy	czas tablicą
t[sekundy]	0,16491	0,201277

Rozmiar tablicy = 1000, liczba wątków = 4, obliczenia równoległe		
	czas mutexy	t z tablicą
t[sekundy]	0,002049	0,000794

Rozmiar tablicy = 100000000, liczba wątków = 4, obliczenia równoległe		
	czas mutexy	czas tablicą
t[sekundy]	0,146538	0,108698

Kolejnym krokiem było odkomentowanie informacji na temat liczby wątków używanych w programie w pliku obliczanie_calki.c ukrytych pod zmienną l_w.

Fragment kodu wraz z sekwencyjnym obliczaniem całki prezentuje się w następujący sposób:

```
  int l_w = 0;
  printf("\nPodaj liczbę wątków:  "); scanf("%d", &l_w);

  printf("\nPoczatek obliczeń sekwencyjnych\n");
  t1 = czas_zegara();

  calka = calka_sekw(a, b, dx);

  t1 = czas_zegara() - t1;
  printf("\nKoniec obliczen sekwencyjnych\n");
  printf("\tCzas wykonania %lf. \tObliczona całka = %.15lf\n", t1, calka);

  printf("\nPoczatek obliczeń równoległych (zrównoleglenie pętli)\n");
  t1 = czas_zegara();

  calka = calka_zrownoleglenie_petli(a, b, dx, l_w);

  t1 = czas_zegara() - t1;
  printf("\nKoniec obliczen  równoległych (zrównoleglenie pętli) \n");
  printf("\tCzas wykonania %lf. \tObliczona całka = %.15lf\n", t1, calka);

  printf("\nPoczatek obliczeń równoległych (dekompozycja obszaru)\n");
  t1 = czas_zegara();

  calka = calka_dekompozycja_obszaru(a, b, dx, l_w);

  t1 = czas_zegara() - t1;
  printf("\nKoniec obliczen  równoległych (dekompozycja obszaru) \n");
  printf("\tCzas wykonania %lf. \tObliczona całka = %.15lf\n", t1, calka);

}

double calka_sekw(double a, double b, double dx){

  int N = ceil((b-a)/dx);
  double dx_adjust = (b-a)/N;

  printf("Obliczona liczba trapezów: N = %d, dx_adjust = %lf\n", N, dx_adjust);
  //printf("a %lf, b %lf, n %d, dx %.12lf (dx_adjust %.12lf)\n", a, b, N, dx, dx_adjust);
  int i;
  double calka = 0.0;
  for(i=0; i<N; i++){

    double x1 = a + i*dx_adjust;
    calka += 0.5*dx_adjust*(funkcja(x1)+funkcja(x1+dx_adjust));
    //printf("i %d, x1 %lf, funkcja(x1) %lf, całka = %.15lf\n",
    //      i, x1, funkcja(x1), calka);

  }

  return(calka);
}
```

Następnym krokiem było zedytowanie pliku dekompozycja pętli.c.

Tworzenie mutexa oraz przyrównanie zmiennych globalnych:

```
pthread_mutex_t mutex; // tworzymy mutex
double calka_zrownoleglenie_petli(double a, double b, double dx, int l_w){

  int N = ceil((b-a)/dx);
  double dx_adjust = (b-a)/N;
  a_global = a;
  b_global = b;
  dx_global = dx_adjust;
  N_global = N;
  l_w_global = l_w;
  printf("Obliczona liczba trapezów: N = %d, dx_adjust = %lf\n", N, dx_adjust);
  //printf("a %lf, b %lf, n %d, dx %.12lf (dx_adjust %.12lf)\n", a, b, N, dx, dx_adjust);
```

Utworzenie struktur danych do obsługi wielowątkowości, tworzenie tablicy globalnej do wersji bez sekcji krytycznej, tworzenie wątków w pętli, zdefiniowanie joina w celu oczekiwania na zakończenie pracy wątków, oraz sumowanie tablicy calka_global, w której każdy wątek ma osobny element i jest przekazywany do wyniku przez wątek główny.

```
  pthread_mutex_init(&mutex, NULL);
  int index [l_w];
  for (int i=0; i < l_w; i++){
        index [i] = i;
  }

  tab_calka_global = (double *) malloc(l_w*sizeof(double));

  pthread_t watki [l_w];
  for (int i=0; i<l_w ; i++){
        pthread_create(&watki[i], NULL, calka_fragment_petli_w, (void*)(&index[i]));
  }

  for (int i=0; i<l_w ; i++){
        pthread_join(watki[i], NULL);
  }

    for (int i=0; i < l_w; i++){
        calka_global += tab_calka_global[i];

  return(calka_global);
}
```

Następnie w funkcji calka_fragment_petli zdefiniowanie dekompozycji blokowej. Dekompozycja blokowa to technika podziału danych na mniejsze bloki w celu łatwiejszego zarządzania i implementacji. Jeżeli rozmiar danych jest podzielny przez liczbę wątków, każdy wątek dostaje blok o tym samym rozmiarze danych.

```
int j = ceil((float)(N/l_w));

if (j*l_w > N) {
      printf("Error!");
}
// dekompozycja blokowa
int my_start = j * my_id;
int my_end = j * (my_id + 1);
if(my_id == l_w - 1){
      my_end = N;
}
int my_stride = 1;
```

Zmienna J odpowiada za obliczenie ilości danych przypisanych do każdego wątku bazując na ilości zadań oraz ilości wątków, funkcja ceil zaokrągla wynik dzielenia w góre, w celu optymalnego rozkładu bloków danych między wątkami, aby ostatni wątek nie został obciążony nadmiernie. Jako start dekompozycji bierzemy nr wątku * ilość danych, końcem danego bloku jest nr wątku + 1 * ilość danych, każdy krok w dekompozycji blokowej my_stride wynosi 1. Mamy też tu warunek dla ostatniego wątku w przypadku niepodzielnego rozmiaru danych względem wątków.

Zdefiniowanie dekompozycji cyklicznej, w przypadku tej dekompozycji dane między wątkami są rozmieszczane co n-ty wątek, czyli np. jeżeli mamy n wątków, to co n-ty element jest przypisywany do danego wątku (1 element do pierwszego wątku, 2 element do drugiego wątku, 3 element do trzeciego wątku, 4 element do pierwszego wątku, 5 element do 5 wątku itd.)

```
int my_start = my_id;
int my_end = N;
int my_stride = l_w;
```

Zmienna my_start określa pierwszy element wątku jako numer wątku, my_id, końcem jest N, czyli rozmiar danych, kolejnym krokiem my_stride jest liczba wątków.

Dokończenie funkcji calka_fragment_petli

```
int i;
double calka = 0.0;
//liczenie calki
for(i=my_start; i<my_end; i+=my_stride){

  double x1 = a + i*dx;
  calka += 0.5*dx*(funkcja(x1)+funkcja(x1+dx));
}
//pthread_mutex_lock(&mutex);
//calka_global += calka;
//pthread_mutex_unlock(&mutex);
tab_calka_global [my_id] = calka;

}
```

Dokonanie pomiarów oraz testowanie dokładności rozwiązania dla różnych wysokości trapezu (dx) na podstawie dekompozycji blokowej.

Przykładowy wynik z konsoli:

```
loprych@loprych-VirtualBox:~/Desktop/PR_lab/lab_5/zad_2/pthreads_calka$ ./oblicz
anie_calki

Program obliczania całki z funkcji (sinus) metodą trapezów.

Podaj wysokość pojedynczego trapezu:  0.01

Podaj liczbę wątków:  4

Poczatek obliczeń sekwencyjnych
Obliczona liczba trapezów: N = 315, dx_adjust = 0.009973

Koniec obliczen sekwencyjnych
        Czas wykonania 0.000027.        Obliczona całka = 1.999983422153753

Poczatek obliczeń równoległych (zrównoleglenie pętli)
Obliczona liczba trapezów: N = 315, dx_adjust = 0.009973

Wątek 1: my_start 1, my_end 315, my_stride 4

Wątek 2: my_start 2, my_end 315, my_stride 4

Wątek 3: my_start 3, my_end 315, my_stride 4

Wątek 0: my_start 0, my_end 315, my_stride 4

Koniec obliczen  równoległych (zrównoleglenie pętli)
        Czas wykonania 0.000744.        Obliczona całka = 1.999983422153754

Poczatek obliczeń równoległych (dekompozycja obszaru)

Koniec obliczen  równoległych (dekompozycja obszaru)
        Czas wykonania 0.000000.        Obliczona całka = 0.000000000000000
```

Średnie wyniki pomiaru dla dx = 0.1 dla kolejno 1,2,4 wątków

Wysokość trapezu h = 0.1, obliczanie sekwencyjne	
Wyniki programu: N=32, dx_adjust=0,098175	
Obliczona całka = 1,99839393360970144	
Pomiary czasu:	
	czas obliczeń sekwencyjnych
t[sekundy]	0,000117

Wysokość trapezu h = 0.1, ilość wątków = 2	
Wyniki programu: N=32, dx_adjust=0,098175	
Obliczona całka równolegle = 1,99839393360970145	
Pomiary czasu:	
	czas obliczeń równoległych
t[sekundy]	0,000521

Wysokość trapezu h = 0.1, ilość wątków = 4	
Wyniki programu: N=32, dx_adjust=0,098175	
Obliczona całka równolegle = 1,99839393360970145	
Pomiary czasu:	
	czas obliczeń równoległych
t[sekundy]	0,0001926

Średnie wyniki pomiaru dla dx = 0.01 dla kolejno 1,2,4 wątków

Wysokość trapezu h = 0.01, obliczanie sekwencyjne	
Wyniki programu: N=315, dx_adjust=0,009973	
Obliczona całka = 1,999983422153753	
Pomiary czasu:	
	czas obliczeń sekwencyjnych
t[sekundy]	0,000032

Wysokość trapezu h = 0.01, ilość wątków = 2	
Wyniki programu: N=315, dx_adjust=0,009973	
Obliczona całka równolegle = 1,999983422153753	
Pomiary czasu:	
	czas obliczeń równoległych
t[sekundy]	0,000819

Wysokość trapezu h = 0.01, ilość wątków = 4	
Wyniki programu: N=315, dx_adjust=0,009973	
Obliczona całka równolegle = 1,999983422153754	
Pomiary czasu:	
	czas obliczeń równoległych
t[sekundy]	0,001524

Średnie wyniki pomiaru dla dx = 0.0001 dla kolejno 1,2,4 wątków

Wysokość trapezu h = 0.0001, obliczanie sekwencyjne	
Wyniki programu: N=31416, dx_adjust=0,0001	
Obliczona całka = 1,999999998333344	
Pomiary czasu:	
	czas obliczeń sekwencyjnych
t[sekundy]	0,001017

Wysokość trapezu h = 0.0001, ilość wątków = 2	
Wyniki programu: N=31416, dx_adjust=0,0001	
Obliczona całka równolegle = 1,999999998333348	
Pomiary czasu:	
	czas obliczeń równoległych
t[sekundy]	0,001339

Wysokość trapezu h = 0.0001, ilość wątków = 4	
Wyniki programu: N=31416, dx_adjust=0,0001	
Obliczona całka równolegle = 1,999999998333347	
Pomiary czasu:	
	czas obliczeń równoległych
t[sekundy]	0,001521

Średnie wyniki pomiaru dla dx = 0.0000001 dla kolejno 1,2,4 wątków

Wysokość trapezu h = 0.0000001, obliczanie sekwencyjne	
Wyniki programu: N=31415927, dx_adjust=0,0000	
Obliczona całka = 1,999999999999808	
Pomiary czasu:	
	czas obliczeń sekwencyjnych
t[sekundy]	0,687007

Wysokość trapezu h = 0.0000001, ilość wątków = 2	
Wyniki programu: N=31415927, dx_adjust=0,0000	
Obliczona całka równolegle = 2,000000000000044	
Pomiary czasu:	
	czas obliczeń równoległych
t[sekundy]	0,426585

Wysokość trapezu h = 0.0000001, ilość wątków = 4	
Wyniki programu: N=31415927, dx_adjust=0,0000	
Obliczona całka równolegle = 2,000000000000162	
Pomiary czasu:	
	czas obliczeń równoległych
t[sekundy]	0,315995

Wnioski:

Wykorzystanie obliczeń równoległych pozwala zwiększenie wydajności oraz skrócenie czasu wykonywania złożonych obliczeniowo programów.

W przypadku obliczania sumy w programie pthreads_suma dla tablicy o rozmiarze 100000000 można zauważyć wykonanie programu nie raz dwukrotnie szybciej, przy wykorzystaniu dla 2 wątków. W przypadku małej tablicy o rozmiarze 1000 program sekwencyjnie wykonuje się szybciej z tego względu, iż zarządzanie wątkami oraz uruchamianie wątków zajmuje dużo czasu w stosunku do obliczeń sekwencyjnych na tak małej tablicy, w tym przypadku obliczenia sekwencyjne okazują się szybsze, lecz jak widać na większym rozmiarze tablicy, użycie n wątków już skraca czas obliczeń około n-krotnie.

W przypadku obliczania całki w programie obliczanie_calki widać, że zrównoleglenie również skraca czas obliczeń kilkukrotnie w przypadku niskiego dx (wysokości trapezu), rzędu 0,000001 ze względu na dużą ilość obliczeń do wykonania. W przypadku wyższych trapezów, przy czym mniejszej ilości obliczeń, program wykonywany sekwencyjnie ma krótszy czas wykonania. Działa to na podobnej zasadzie jak przy programie pthreads_suma, ponieważ zarządzanie wątkami oraz ich inicjowanie wymaga trochę dodatkowego czasu, który zwraca się w szybszym wykonaniu dużych obliczeń. Wynik całki jest zaokrąglany do podobnej nieomal identycznych wartości w przypadku sekwencyjnym oraz zrównoleglonym, ponieważ mamy tę samą liczbę iteracji oraz wysokość trapezu dx, różnice co najwyżej mogą wynikać z powodu maszyny, na której wykonywany jest program. Zwiększenie liczby trapezów (zmniejszenie dx) pozwala nam na uzyskanie dokładniejszych wyników całkowania (np. ok. 1,99 przy dx = 0.1 oraz ok. 1,999999999999 przy dx = 0,000001), odbija się to na czasie wykonywania programu.