Tomasz Ligęza

Programowanie równoległe. Przetwarzanie równoległe i rozproszone.

Sprawozdanie z laboratorium 5.

Cel zajęć:

Nabycie umiejętności tworzenia i implementacji programów równoległych w środowisku Pthreads.

W ramach zajęć zrealizowałem następujące kroki:

1. Utworzyłem katalog roboczy lab_5.
2. Pierwszym zadaniem było pobranie przygotowanego wcześniej programu liczącego sumę elementów tablicy, uruchomienie go oraz przetestowanie działania wersji sekwencyjnej oraz równoległej w zależności od rozmiaru sumowanej tablicy:

Prosty wzorzec zrównoleglenia pętli wygląda następująco:

```
int indeksy[LICZBA_W];
for (i = 0; i < LICZBA_W; i++) indeksy[i] = i;

pthread_mutex_init(&muteks, NULL);

for (i = 0; i < LICZBA_W; i++)
   pthread_create(&watki[i], NULL, suma_w, (void *) &indeksy[i]);
for (i = 0; i < LICZBA_W; i++) pthread_join(watki[i], NULL);

void *suma_w(void *arg_wsk) {

   int i, j, moj_id;
   double moja_suma = 0;
   moj_id = *((int *) arg_wsk);
   j = ceil((float) ROZMIAR / LICZBA_W);
   for (i = j * moj_id; i < j * (moj_id + 1); i++) {
       moja_suma += tab[i];
   }

   pthread_mutex_lock(&muteks);
   suma += moja_suma;
   pthread_mutex_unlock(&muteks);

   pthread_exit(NULL);
}
```

Praca każdego z wątków polega na obliczeniu sumy zadanego mu zakresu tablicy. Zakres tablicy obliczamy w ten sposób:
start_index = `ceil((float) ROZMIAR / LICZBA_W) * moj_id;`

end_index = `ceil((float) ROZMIAR / LICZBA_W) * (moj_id + 1);`

Iterując od indeksu początkowego do ostatniego dodajemy do lokalnej sumy wartości elementów tablicy. Ostatecznym krokiem jest zablokowanie globalnego muteksa, oraz dodanie lokalnej sumy do sumy globalnej.

Jest to przykład dekompozycji blokowej.

3. Kolejnym zadaniem było pobranie programu obliczającego całkę metodą trapezów, jego uruchomienie oraz przetestowanie dla różnych dokładności całkowania (wartość dx).
4. Następnie należało uzupełnić program o wariant obliczania całki w sposób wielowątkowy - najpierw wykorzystując dekompozycję cykliczną, a później blokową:

Program wywołuje funkcję, która ma zająć się uruchomieniem wątków i podzieleniem zakresu problemu na wątki:

```
double calka_zrownoleglenie_petli(double a, double b, double dx) {

   int N = ceil((b - a) / dx);
   double dx_adjust = (b - a) / N;
   printf("Obliczona liczba trapezów: N = %d\n", N);
   int l_w = l_w_global;
   a_global = a;
   b_global = b;
   dx_global = dx_adjust;
   N_global = N;
   int indeksy[LICZBA_W_MAX];
   pthread_mutex_init(&muteks, NULL);

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

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

Utworzone wątki otrzymują jako argument swój numer id, na podstawie którego obliczają zakres całkowania jaki mają zrealizować (zakres dla dekompozycji blokowej, a poszczególne indeksy dla dekompozycji cyklicznej).

```
void *calka_fragment_petli_w(void *arg_wsk) {

    int my_id = *((int *) arg_wsk);

    double a, b, dx;
    int N, l_w;
    a = a_global;
    b = b_global;
    dx = dx_global;
    N = N_global;
    l_w = l_w_global;
        // dekompozycja CYKLICZNA
//      int my_start = my_id;
//      int my_end = N;
//      int my_stride = l_w;

        // dekompozycja BLOKOWA
    int el_na_watek = ceil((float) N / l_w_global);
    int my_start = el_na_watek * my_id;
    int my_end = el_na_watek * (my_id + 1);
    if (my_end > N) my_end = N;
    int my_stride = 1;

    int i;
    double calka = 0.0;
    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(&muteks);
    calka_global += calka;
    pthread_mutex_unlock(&muteks);
}
```

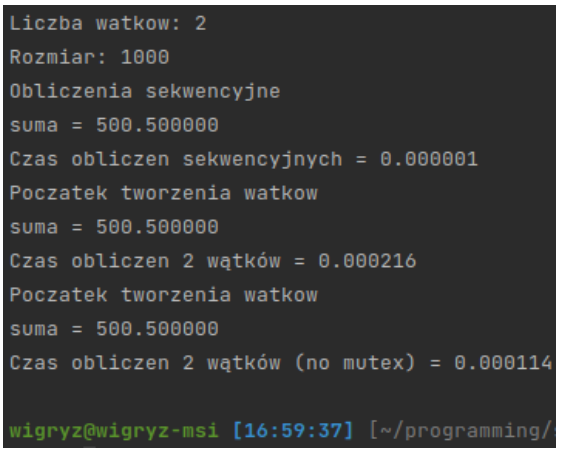
Dekompozycja cykliczna polega na przydzieleniu wątkom wybiórczych indeksów tabeli dla których mają zrealizować dane obliczenia. Pierwszym indeksem jest id wątku, kolejnym i każdym następnym jest id wątku pomnożone przez liczbę wątków aż do momentu, kiedy indeks byłby większy od liczby indeksów tablicy.

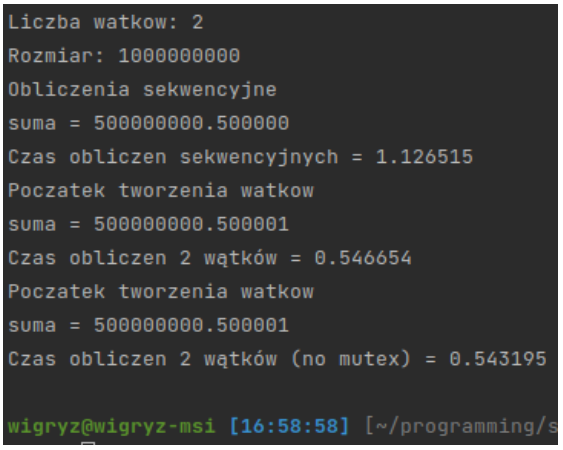
Dekompozycja blokowa polega na przydzieleniu wątkom bloków indeksów tabeli dla których mają zrealizować dane obliczenia. Opis obliczeń dla dekompozycji blokowej zawarłem w punkcie drugim sprawozdania.

Finalny wynik lokalny całki dodawany jest do globalnego wyniku po zablokowaniu muteksa.

Wnioski:

- Wyniki programu pthreads_suma:





	rozmiar tabeli		
l. wątków	1000	1000000	1000000000
sekwencyjnie			
1	0.000001	0.001172	1.126515
równolegle (muteks)			
2	0.000216	0.000793	0.546654
4	0.001487	0.000571	0.348690
równolegle (bez muteksu)			
2	0.000114	0.000603	0.543195
4	0.000067	0.000351	0.345546

Wyniki wersji równoległych zgadzają się z wersją sekwencyjną. Czas wykonywania sekwencyjnej wersji programu jest krótszy w przypadku tabeli o rozmiarze 1000, lecz w każdym innym przypadku ustępuje szybkością wykonania wersjom równoległym. Wersja bez muteksu jest najszybsza.

- Wyniki programu pthreads_calka:

```
Podaj wysokość pojedynczego trapezu:  0.00001

Podaj liczbę wątków:  2

Poczatek obliczeń sekwencyjnych
Obliczona liczba trapezów: N = 314160

Koniec obliczen sekwencyjnych
        Czas wykonania 0.010386.        Obliczona całka = 1.999999999983388

Poczatek obliczeń równoległych (zrównoleglenie pętli)
Obliczona liczba trapezów: N = 314160

Koniec obliczen  równoległych (zrównoleglenie pętli)
        Czas wykonania 0.005974.        Obliczona całka = 1.999999999983346

Poczatek obliczeń równoległych (dekompozycja obszaru)

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

wigryz@wigryz-msi [17:49:12] [~/programming/studies/parallel-programming/lab_5]
```

	wysokość pojedynczego trapezu			
l. wątków	0.1	0.01	0.001	0.0001
sekwencyjnie				
1	0.000084	0.000110	0.000331	0.002422
dekompozycja cykliczna				
2	0.000355	0.000298	0.000401	0.001545
4	0.000526	0.000694	0.000526	0.001810
dekompozycja blokowa				
2	0.000089	0.000338	0.000702	0.001684
4	0.000609	0.000444	0.000539	0.000887

Obliczenia dla wersji równoległej mają przewagę dopiero dla wysokości trapezu równej 10^{-4}. Obliczenia prowadzone przez 4 wątki trwały dłużej w większości przypadków. Czasy dla dekompozycji blokowej i cyklicznej są do siebie mocno zbliżone i można uznać,

że żadna z dekompozycji nie ma przewagi nad drugą. Wynik dla dekompozycji blokowej dla wysokości trapezu równej 10^{-4} otrzymany w wyniku pracy 4 wątków odbiegają od reszty - wykorzystanie 4 wątków zamiast dwóch dało prawie dwukrotne przyspieszenie pracy. Mogło to być spowodowane np. mniejszym zużyciem zasobów komputera w chwili uruchomienia programu.