Tomasz Ligęza

Programowanie równoległe. Przetwarzanie równoległe i rozproszone.

Sprawozdanie z laboratorium 10.

Cel zajęć:

Pogłębienie umiejętności pisania programów równoległych w środowisku OpenMP

W ramach zajęć zrealizowałem następujące kroki:

1. Utworzyłem katalog roboczy lab_10.
2. Skopiowałem pliki ze strony przedmiotu, skompilowałem je oraz uruchomiłem.
3. Poprawiłem czytelność wydruków oraz użyłem dyrektyw OpenMP tak, aby każdy wątek drukował bez ingerencji innych. Sprawdziłem również program w wersji wielowątkowej, i poprawiłem problem ze zmienną a_shared, e_atomic oraz d_local_private (gdzie wymagane było użycie bariery):

```
   static int f_thread_private;
#pragma omp threadprivate(f_thread_private)

#pragma omp parallel default(none) shared(a_shared, e_atomic)
      private(b_private) firstprivate(c_firstprivate )
   {
       int i;
       int d_local_private;
       d_local_private = a_shared + c_firstprivate;
       f_thread_private = omp_get_thread_num();
#pragma omp barrier

#pragma omp critical(a_shared)
       for (i = 0; i < 10; i++) {
           a_shared++;
       }

       for (i = 0; i < 10; i++) {
           c_firstprivate += omp_get_thread_num();
       }

       for (i = 0; i < 10; i++) {
#pragma omp atomic
           e_atomic += omp_get_thread_num();
       }
#pragma omp barrier
#pragma omp critical(print)
       {
```

```
            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);
        }
```

4. Uruchomiłem program z domyślną liczbą wątków i sprawdziłem poprawność działania programu:

```
wigryz@wigryz-msi [17:49:28] [~/programming/studies/parallel-programming/lab
-> % ./run.sh
rm -f *.o openmp_watki_zmienne
gcc -c -g -DDEBUG  -fopenmp openmp_watki_zmienne.c
gcc -g -DDEBUG  -fopenmp openmp_watki_zmienne.o -o openmp_watki_zmienne -lm

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 8, moj ID 6
        a_shared        = 81
        b_private       = 0
        c_firstprivate  = 63
        d_local_private = 4
        e_atomic        = 285

w obszarze równoległym: aktualna liczba watkow 8, moj ID 3
        a_shared        = 81
        b_private       = 0
        c_firstprivate  = 33
        d_local_private = 4
```

[...]

5. Następnym krokiem było napisanie drugiego obszaru równoległego w którym testuję działanie dyrektywy threadprivate:

```
   printf("Thread private:\n");
#pragma omp parallel default(none)
   {
       printf("\twatek nr= %d, f_thread_private= %d\n",
              omp_get_thread_num(), f_thread_private);
   }
```

```
Thread private:
        watek nr= 3, f_thread_private= 3
        watek nr= 2, f_thread_private= 2
        watek nr= 0, f_thread_private= 0
        watek nr= 1, f_thread_private= 1
        watek nr= 4, f_thread_private= 4
        watek nr= 6, f_thread_private= 6
        watek nr= 5, f_thread_private= 5
        watek nr= 7, f_thread_private= 7
```

6. Utworzyłem kolejny podkatalog roboczy pde.
7. Uruchomiłem program i przeanalizowałem zależności w pętli z obliczeniami.

W pętli do obliczenia jednego elementu tablicy A[i] używamy elementu A[i+2], więc nie możemy tego w prosty sposób zrównoleglić. Rozwiązaniem tego problemu jest utworzenie tablicy pomocniczej zawierającej te same elementu co tablica A, dzięki czemu do obliczania elementów tablicy A będziemy mogli używać tablicy pomocniczej.

```
   double suma;
   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;
   for (i = 0; i < N + 2; i++) aux[i] = A[i];

[...]

   // wersja równoległa
#pragma omp parallel for default(none) shared(A, aux, B) num_threads(2)
   for (i = 0; i < N; i++) {
       A[i] += aux[i + 2] + sin(B[i]);
   }
```

```
wigryz@wigryz-msi [18:16:02] [~/programming/studies/parallel-programming/l
-> % ./run.sh
rm -f *.o openmp_zaleznosci
gcc -c -g -DDEBUG  -fopenmp openmp_zaleznosci.c
gcc -g -DDEBUG  -fopenmp openmp_zaleznosci.o -o openmp_zaleznosci -lm
suma 1459701.114868, czas obliczen 0.009877
suma 1459701.114868, czas obliczen rownoleglych 0.005224
```

Wnioski:

Po raz kolejny przekonałem się jak wiele ułatwia środowisko OpenMP w aspekcie programowania równoległego. Ostatni zrealizowany przeze mnie problem pokazał, że czasem trzeba zastosować jakiś “trick”, zamiast na siłę używać różnych dyrektyw środowiska OpenMP, żeby osiągnąć zrównoleglenie.