Przemysław Szymoniak

Przetwarzanie współbieżne. Programowanie równoległe i rozproszone.

Sprawozdanie z laboratorium 9.
Celem laboratorium było nabycie umiejętności tworzenia oraz implementacji programów równoległych z wykorzystaniem OpenMP.

W ramach zajęć zrealizowałem następujące kroki:

- Pobrałem i rozpakowałem wszystkie wymagane pliki.

- Napisałem 4 wersje zrównoleglenia pętli:

1. Wierszowe – zrównoleglenie pętli zewnętrznej (klauzula reduction)

```
double suma_parallel=0.0;
#pragma omp parallel for private(j) reduction(+:suma_parallel)
for(i=0;i<WYMIAR;i++) {
    int id_w = omp_get_thread_num();
    for(j=0;j<WYMIAR;j++) {
        suma_parallel += a[i][j];
        printf("(%2d,%2d)-W(%1d,%1d) ",i,j,id_w,omp_get_thread_num());
    }
    printf("\n");
}
printf("Suma wyrazów tablicy równolegle: %lf\n", suma_parallel);
```

2. Wierszowe – zrównoleglenie pętli wewnętrznej (tablica wartości dla wątków)

```
double *tab_sum = malloc(sizeof *tab_sum * number_of_threads);
for ( i = 0; i < number_of_threads; i++ )//tablica do sumowania
    tab_sum[i] = 0.0;

for(i=0;i<WYMIAR;i++) {
    int id_w = omp_get_thread_num();
    #pragma omp parallel for firstprivate(i) num_threads(3)
    for(j=0;j<WYMIAR;j++) {
        tab_sum[omp_get_thread_num()] += a[i][j];
        printf("(%2d,%2d)-W(%1d,%1d) ",i,j,id_w,omp_get_thread_num());
    }
    printf("\n");
}

int k;
for ( k = 0; k < number_of_threads; k++ )
        suma_parallel += tab_sum[k];

printf("Suma wyrazów tablicy równolegle: %lf\n", suma_parallel);
```

3. Kolumnowe – zrównoleglenie pętli zewnętrznej (klauzula reduction)

```
for(i=0;i<WYMIAR;i++) {
    int id_w = omp_get_thread_num();
    #pragma omp parallel for firstprivate(i) reduction(+:suma_parallel)
    for(j=0;j<WYMIAR;j++) {
    suma_parallel += a[i][j];
    printf("(%2d,%2d)-W(%1d,%1d) ",i,j,id_w,omp_get_thread_num());
    }
    printf("\n");
}
printf("Suma wyrazów tablicy równolegle: %lf\n", suma_parallel);
```

4. Kolumnowe – zrównoleglenie pętli zewnętrznej

```
double suma_czesciowa = 0.0;
#pragma omp parallel for  private(i,j, suma_czesciowa) num_threads(3)
for(i=0;i<WYMIAR;i++) {
    suma_czesciowa=0.0;
    int id_w = omp_get_thread_num();
    for(j=0;j<WYMIAR;j++) {
        suma_czesciowa += a[i][j];
        printf("(%2d,%2d)-W(%1d,%1d) ",i,j,id_w,omp_get_thread_num());
    }
    #pragma omp critical
    {
        suma_parallel += suma_czesciowa;
    }
    printf("\n");
}
printf("Suma wyrazów tablicy równolegle: %lf\n", suma_parallel);
```

- Tworzenie podziału wariantu dekompozycji tworzymy poprzez zmiane **//#pragma omp parallel for** na:
    - **#pragma omp parallel for ordered schedule(static, 3)**
    - **#pragma omp parallel for ordered schedule(static)**
    - **#pragma omp parallel for ordered schedule(dynamic, 2)**
    - **#pragma omp parallel for ordered schedule(dynamic)**

- Dodatkowo by uzyskać większą czytelność wydruku dodajemy **#pragma omp ordered** w środkowej pętli tuż przed **printf("(%2d, %2d) - W(%1d, %1d)", i, j, id, omp_get_thread_num());**

- W załączniku obecne są 4 pliki będące wynikiem testowania program dla zadania 5 (4 warianty w jednym pliku)

Wnioski:

- Interfejs programowania OpenMP składa się z dyrektyw kompilatora, biblioteki i zmiennych środowiskowych
- Pełna kontrola nad zrównolegleniem jest osiągana przez konieczność jawnego definiowania, które fragmenty kody mają wykonać się równolegle
- Dyrektywa **#pragma omp critical** pozwala wykorzystać sekcję krytyczną do której dostęp ma tylko jeden wątek w danym czasie
- Dostępnymi klauzulami współdzielenia zmiennych są:
    1. Shared – zmienna jest współdzielona między wątkami
    2. Private – każdy wątek otrzymuje zmienna dostępną tylko dla siebie i nie osiągalną dla innych
    3. Firstprivate – jest to zmienna typu Private jednak dla wszystkich wątków została zainicjowana tą samą wartością
    4. Lastprivate – również analogiczna do Private jednak wartość końcowa ma być równa wartości z wariantu sekwencyjnego

- **Schedule(static, liczba)** jest klauzulą dzięki której przydział zadań odbywa się za pomocą bloków o rozmiarze **„liczba”**, bloki iteracji przydzielane są cyklicznie. W momencie nie określenia **„liczby”** wielkość bloku jest określana wzorem (liczbaIteracji + liczbaWątów -1)/liczbaWątków
- Przy wykorzystaniu klauzuli **Schedule(dynamic, liczba)**, każdy wątek otrzymuje do przetworzenia blok iteracji o rozmiarze określonym przez **„liczba”**, w momencie zakończenia go przydzielony zostaje następny blok iteracji. W momencie nie określenia **„liczby”** blok równy jest 1
- Wykorzystanie **#pragma omp ordered** powoduje, że w danej chwili wykonywany jest tylko 1 wątek w porządku zgodnym z ruchem w pętli sekwencyjnej
- Po wykorzystanie klauzuli **reduction** oczekujemy wyniku zgodnego z jego sekwencyjnym odpowiednikiem