Antoni Pawlak

401480

Laboratorium 9: OpenMP

Cel ćwiczenia

- Tworzenie programów równoległych przy użyciu OpenMP
- Poznanie składni OpenMP
- Testowanie implementacji znanych wzorców w bibliotece OMP

Wykonanie

Przygotowania

1. Utworzenie katalogu i pobranie plików ze strony
2. Skompilowanie kodu i uruchomienie programu

   gcc -fopenmp openmp_petle_simple.c -o openmp_petle_simple)

Zrównoleglanie pętli simple

3. Statyczny przydział, rozmiar porcji 3

```
#pragma omp parallel for default(none) schedule(static,3)
shared(suma_parallel, a) private(i) ordered
```

4. Statyczny przydział, domyślny rozmiar porcji

```
#pragma omp parallel for default(none) schedule(static)
shared(suma_parallel, a) private(i) ordered
```

5. Dynamiczny przydział, rozmiar porcji 2

```
#pragma omp parallel for default(none) schedule(dynamic,2)
shared(suma_parallel, a) private(i) ordered
```

6. Dynamiczny przydział, domyślny rozmiar pętli

```
#pragma omp parallel for default(none) schedule(dynamic)
shared(suma_parallel, a) private(i) ordered
```

7. Zrównoleglana pętla

```
for(i=0;i<WYMIAR;i++) {
    int id_w = omp_get_thread_num();
    #pragma omp atomic //sekcja krytyczna
    suma_parallel += a[i];
    #pragma omp ordered //wyniki po kolei
    printf("a[%2d]->W_%1d \n",i,id_w);
    }
```

Zrównoleglanie pętli

8. Dekompozycja wierszowa, reduction, static 2

```
#pragma omp parallel for default(none) schedule(static,2) private(i,j) shared(a)
ordered 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];
            #pragma omp ordered
            printf("(%1d,%1d)-W_%1d ",i,j,omp_get_thread_num());
            }
    #pragma omp ordered
    printf("\n");
    }
```

9. Dekompozycja kolumnowa – pętla wewnętrzna, reduction, dynamic default

```
for(i=0;i<WYMIAR;i++) {
    int id_w = omp_get_thread_num();
    #pragma omp parallel for default(none) schedule(dynamic) private(j)
    shared(i, a) ordered reduction(+:suma_parallel)
            for(j=0;j<WYMIAR;j++) {
            suma_parallel += a[i][j];
            #pragma omp ordered
            printf("(%1d,%1d)-W_%1d ",i,j,omp_get_thread_num());
            }
    #pragma omp ordered
    printf("\n");
    }
```

10. Dekompozycja kolumnowa – pętla zewnętrzna, ręczne sterowanie, static default

```
#pragma omp parallel for default(none) schedule(static) private(i,j) shared(a,
suma_parallel) ordered firstprivate(tmp_suma)
    for(i=0;i<WYMIAR;i++) {
    int id_w = omp_get_thread_num();
    tmp_suma = 0.0;
            for(j=0;j<WYMIAR;j++) {
            tmp_suma += a[j][i];
            #pragma omp ordered
            printf("(%1d,%1d)-W_%1d ",j,i,omp_get_thread_num());
            }
    #pragma omp ordered
    printf("\n");
    #pragma omp atomic
    suma_parallel += tmp_suma;
    }
```

Wnioski

Zrównoleglanie simple

Statyczna | 3	Statyczna | domyślny	Dynamiczna | 2	Dynamiczna | domyślny
a[0]->W_0	a[0]->W_0	a[0]->W_2	a[0]->W_1
a[1]->W_0	a[1]->W_0	a[1]->W_2	a[1]->W_2
a[2]->W_0	a[2]->W_0	a[2]->W_1	a[2]->W_0
a[3]->W_1	a[3]->W_0	a[3]->W_1	a[3]->W_3
a[4]->W_1	a[4]->W_0	a[4]->W_0	a[4]->W_1
a[5]->W_1	a[5]->W_1	a[5]->W_0	a[5]->W_2
a[6]->W_2	a[6]->W_1	a[6]->W_3	a[6]->W_0
a[7]->W_2	a[7]->W_1	a[7]->W_3	a[7]->W_3
a[8]->W_2	a[8]->W_1	a[8]->W_2	a[8]->W_1
a[9]->W_3	a[9]->W_1	a[9]->W_2	a[9]->W_2
a[10]->W_3	a[10]->W_2	a[10]->W_1	a[10]->W_0
a[11]->W_3	a[11]->W_2	a[11]->W_1	a[11]->W_3
a[12]->W_0	a[12]->W_2	a[12]->W_0	a[12]->W_1
a[13]->W_0	a[13]->W_2	a[13]->W_0	a[13]->W_2
a[14]->W_0	a[14]->W_3	a[14]->W_3	a[14]->W_0
a[15]->W_1	a[15]->W_3	a[15]->W_3	a[15]->W_3
a[16]->W_1	a[16]->W_3	a[16]->W_2	a[16]->W_1
a[17]->W_1	a[17]->W_3	a[17]->W_2	a[17]->W_2

Jaka jest domyślna porcja dla każdej z wersji?

- Static – dzieli pętle na możliwie równe bloki
- Dynamic – dzieli pętle na chunki o wielkości 1

Jaka jest kolejność przydzielania iteracji wątkom?

- Static – idzie w kolejności wraz z iteratorem pętli
- Dynamic – procesor sam przydziela wątki bez wpywu programisty

Która z wersji jest przyjmowana dla dyrektywy bez klauzuli schedule?

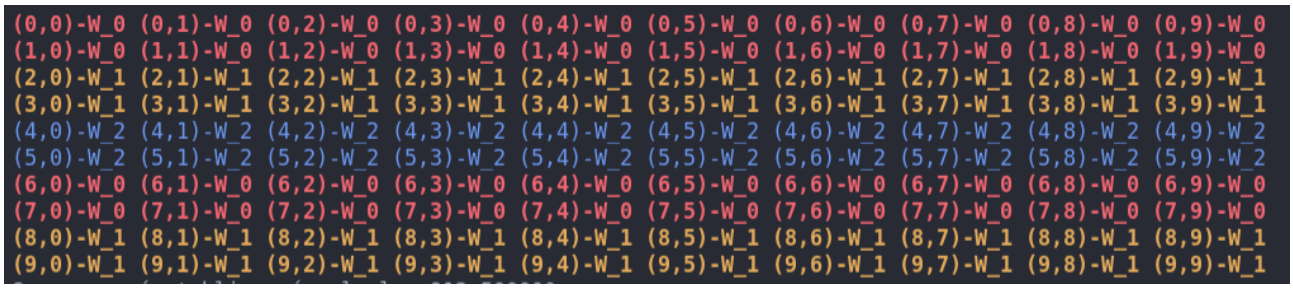
- Static default, domyślnie omp zrównolegla blokowo

Czy przy każdym uruchomieniu wątki dostają te same iteracje?

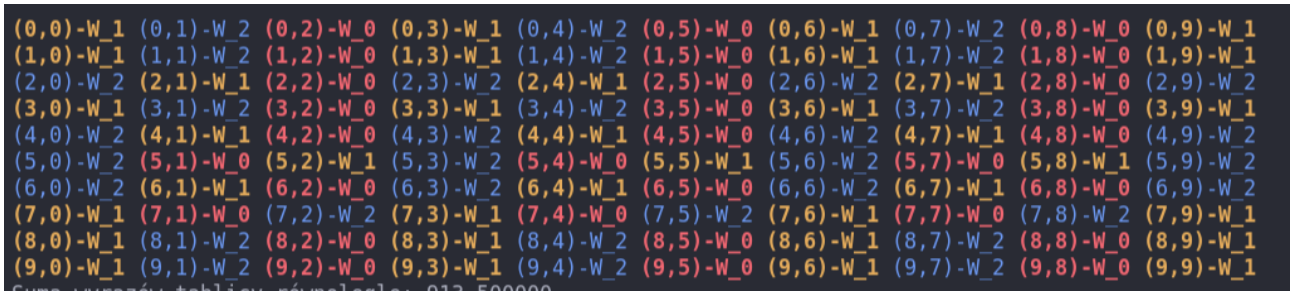
- Static, TAK, za każdym razem przydziela tym samym wątkow
- Dynamic, NIE, za każdym razem mogą być przydzielone inne wątki

Zrównoleglanie

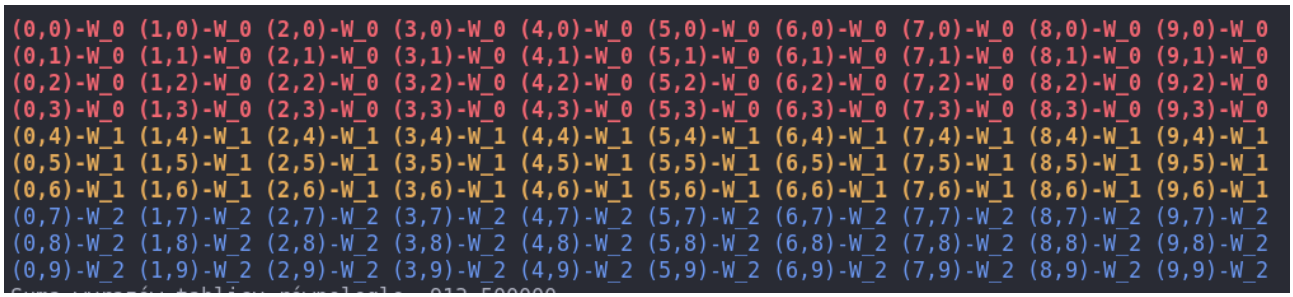
Wierszowe, static 2



Kolumnowe, dynamic



Kolumnowe zewnętrzne, static



Jakie przypisanie wierszy dają różne wersje dyrektywy schedule?

- Static przydziela wiersze blokowo zgodnie z iteratorem
  - domyślnie dzieli na równe bloki
- Dynamic przydziela cyklicznie zgodnie z chwilowym stanem CPU
  - domyślnie dzieli na chunki o rozmiarze 1

SCHEDULE – dynamic vs static

- OMP domyślnie używa **static default**
- Static jest bardziej deterministyczne i przewidywalne dla programisty
- Dynamic jest bardziej elastyczne dla CPU

Kiedy używać static?

- Kiedy posiadamy pętle, gdzie w kolejnych iteracjach będziemy mieli podobnej wielkości obliczenia
- Kiedy zależy nam na kolejność przydzielania obliczeń do wątków

Kiedy używać dynamic?

- Kiedy posiadamy pętle o wysoce asymetrycznym rozkładzie złożoności obliczeń w poszczególnych iteracjach pętli
- Kiedy chcemy aby program działał najwydajniej jak tylko się da

Czy można powiedzieć, że jedno jest lepsze od drugiego?

- NIE, obie klauzule mają swoje zastosowania, należy traktować je jak narzędzia, których należy w odpowiedni sposób używać