Използване на OpenMP. Част 2. for, barrier, section, master, single

Курс „Паралелно програмиране“

ИНСТИТУТ *за* СЪВРЕМЕННИ
ФИЗИЧЕСКИ ИЗСЛЕДВАНИЯ

Стоян Мишев

План

Задача

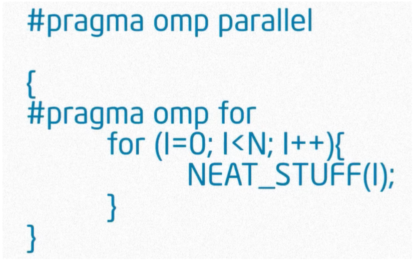
$$\int_0^1 \frac{4}{1+x^2}dx = \pi$$

SPMD <-> Worksharing

Loop
Section
Single
Task

Loop

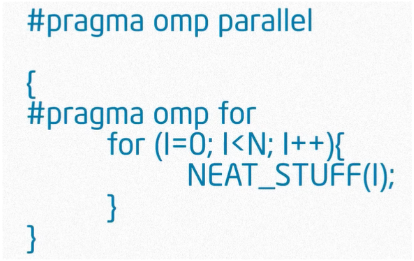
Нишките изпълняват итерации с различен номер.



```
#pragma omp parallel

{
#pragma omp for
        for (I=0; I<N; I++){
                NEAT_STUFF(I);
        }
}
```

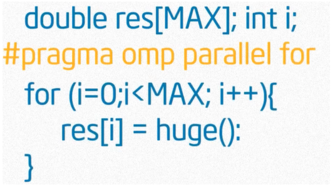
Loop

Нишките изпълняват итерации с различен номер.



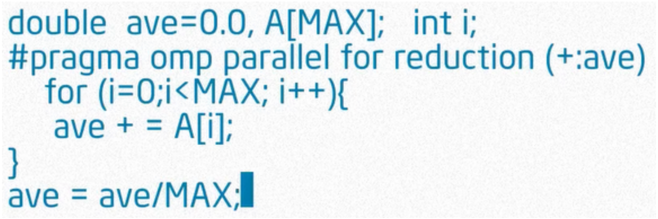
```
#pragma omp parallel

{
#pragma omp for
        for (I=0; I<N; I++){
                NEAT_STUFF(I);
        }
}
```

Съкратен запис:



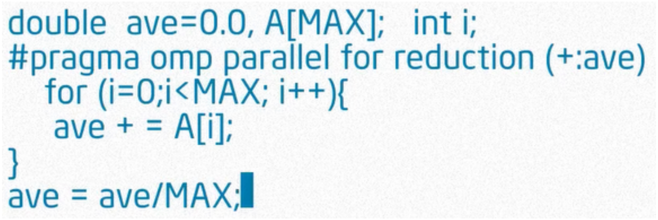
```
 double res[MAX]; int i;
#pragma omp parallel for
 for (i=0;i<MAX; i++){
     res[i] = huge():
 }
```

Reduction

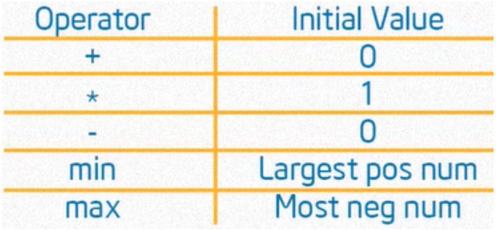


```
double  ave=0.0, A[MAX];   int i;
#pragma omp parallel for reduction (+:ave)
   for (i=0;i<MAX; i++){
    ave + = A[i];
}
ave = ave/MAX;
```

Reduction



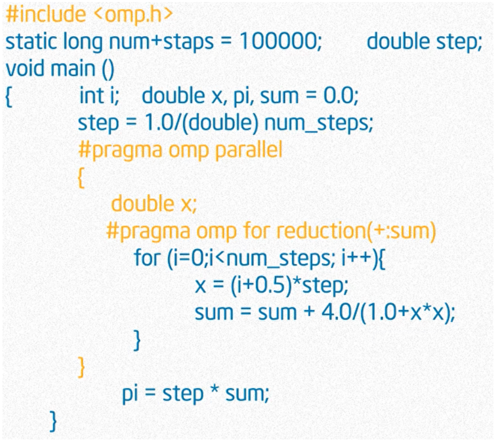
```
double  ave=0.0, A[MAX];   int i;
#pragma omp parallel for reduction (+:ave)
   for (i=0;i<MAX; i++){
    ave + = A[i];
}
ave = ave/MAX;
```



Operator	Initial Value
+	0
*	1
-	0
min	Largest pos num
max	Most neg num

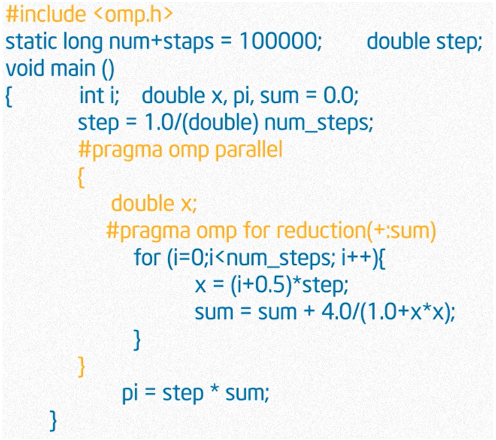
```c
  int tnumber;
  int i = 10, j = 10, k = 10;
  printf("Before parallel region: i=%i, j=%i, k=%i\n",
      i, j, k);
#pragma omp parallel default(none) private(tnumber)
    reduction(+:i) reduction(*:j) reduction(^:k)
  {
    tnumber = omp_get_thread_num() + 1;
    i = tnumber;
    j = tnumber;
    k = tnumber;
    printf("Thread %i: i=%i, j=%i, k=%i\n", tnumber, i,
        j, k);
  }
  printf("After parallel region: i=%d, j=%d, k=%d\n", i
      , j, k);
```

Integral

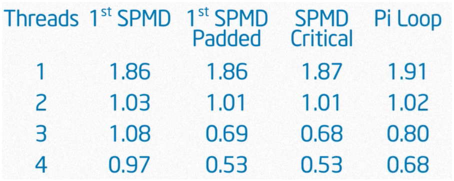


```
#include <omp.h>
static long num+staps = 100000;        double step;
void main ()
{          int i;    double x, pi, sum = 0.0;
           step = 1.0/(double) num_steps;
           #pragma omp parallel
           {
                double x;
               #pragma omp for reduction(+:sum)
                   for (i=0;i<num_steps; i++){
                            x = (i+0.5)*step;
                            sum = sum + 4.0/(1.0+x*x);
                   }
           }
                 pi = step * sum;
       }
```

Integral

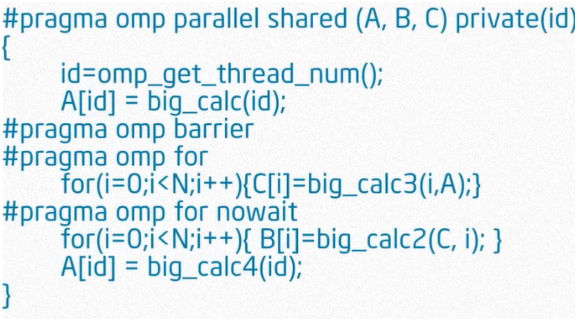


```
#include <omp.h>
static long num+staps = 100000;        double step;
void main ()
{          int i;    double x, pi, sum = 0.0;
           step = 1.0/(double) num_steps;
           #pragma omp parallel
           {
                double x;
               #pragma omp for reduction(+:sum)
                   for (i=0;i<num_steps; i++){
                            x = (i+0.5)*step;
                            sum = sum + 4.0/(1.0+x*x);
                   }
           }
                pi = step * sum;
      }
```



Threads	1st SPMD	1st SPMD Padded	SPMD Critical	Pi Loop
1	1.86	1.86	1.87	1.91
2	1.03	1.01	1.01	1.02
3	1.08	0.69	0.68	0.80
4	0.97	0.53	0.53	0.68

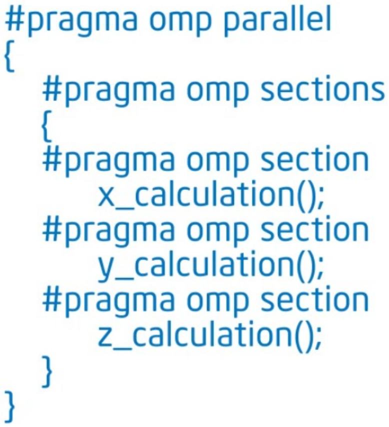
Barrier



```
#pragma omp parallel shared (A, B, C) private(id)
{
        id=omp_get_thread_num();
        A[id] = big_calc(id);
#pragma omp barrier
#pragma omp for
        for(i=0;i<N;i++){C[i]=big_calc3(i,A);}
#pragma omp for nowait
        for(i=0;i<N;i++){ B[i]=big_calc2(C, i); }
        A[id] = big_calc4(id);
}
```

Section

Нишките изпълняват код от различни `section`



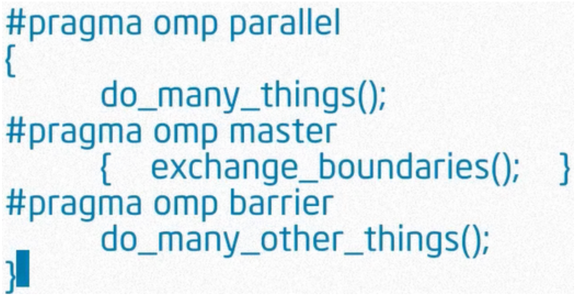
```
#pragma omp parallel
{
    #pragma omp sections
    {
    #pragma omp section
        x_calculation();
    #pragma omp section
        y_calculation();
    #pragma omp section
        z_calculation();
    }
}
```

Section

```c
int a = 6;
int b = 3;
omp_set_num_threads(4);
#pragma omp parallel
{
  #pragma omp sections
  {
    #pragma omp section
    {
      printf("Sum = %d on thread %d \n", a + b,
           omp_get_thread_num());
    }
    #pragma omp section
    {
      printf("Difference = %d on thread %d \n", a - b,
           omp_get_thread_num());
    }
  }
}
```

Master

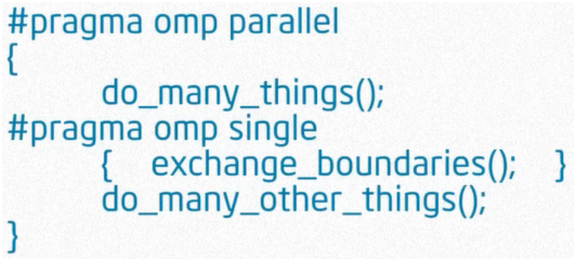
Когато искаме някоя част от кода да се изпълни само от една нишка използваме `master` или `single`. При `signle` нишката, която първа достигне до кода, го изпълнява, докото при `master` точно нишката с `id=0` изпълнява кода (, а останалите - не). Няма скрит `barrier` след `master`, но има скрит `barrier` след `single`.



```
#pragma omp parallel
{
        do_many_things();
#pragma omp master
        {    exchange_boundaries();    }
#pragma omp barrier
        do_many_other_things();
}
```

```
int i = 0, N = 8;
omp_set_num_threads(N);
int *a, *b, *c;
#pragma omp parallel
{
  #pragma omp master
  {
    a = malloc(N * sizeof(int));
    b = malloc(N * sizeof(int));
    c = malloc(N * sizeof(int));
    srand(time(NULL));
  }
  #pragma omp for
  for( i  = 0; i < N; i++) {
    a[i] = rand() % 10;
    b[i] = rand() % 10;
  }
  #pragma omp for
  for( i  = 0; i < N; i++) {
      c[i] = a[i] * b[i];
  }
  #pragma omp for
  for( i = 0; i < N ; i++) {
    printf("A[%d] * B[%d] = %d \n", i, i, c[i]);
  }
```

Single



```
#pragma omp parallel
{
        do_many_things();
#pragma omp single
        {   exchange_boundaries();   }
        do_many_other_things();
}
```

```c
int i = 0, N = 8;
omp_set_num_threads(N);
int *a, *b, *c;
#pragma omp parallel
{
  #pragma omp single
  {
    a = malloc(N * sizeof(int));
    b = malloc(N * sizeof(int));
    c = malloc(N * sizeof(int));
    srand(time(NULL));
  }
  #pragma omp for
  for( i  = 0; i < N; i++) {
    a[i] = rand() % 10;
    b[i] = rand() % 10;
  }
  #pragma omp for
  for( i  = 0; i < N; i++) {
      c[i] = a[i] * b[i];
  }
  #pragma omp for
  for( i = 0; i < N ; i++) {
    printf("A[%d] * B[%d] = %d \n", i, i, c[i]);
```

Домашна работа

`https://www.youtube.com/watch?list=PLLbPZJxtMs4ZHSamRRYCtvowRS0qIwC-I`
От “Introduction to OpenMP 08 Discussion 3 ”
до “Introduction to OpenMP 11 part 1 Module 6”.