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

Курс „Паралелно програмиране“

ИНСТИТУТ *за* СЪВРЕМЕННИ
ФИЗИЧЕСКИ ИЗСЛЕДВАНИЯ

Стоян Мишев

План

Loop
Section
Master
Single

Нишките изпълняват итерации с различен номер.

```
#pragma omp parallel

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

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():
  }
```

```
1   int i = 0;
2   omp_set_num_threads(4);
3
4   printf("Total number of threads allocated in the
        serial section %d \n", omp_get_num_threads() );
5   #pragma omp parallel
6   {
7     #pragma omp for
8     for( i = 0; i < omp_get_num_threads(); i++) {
9       printf("This is run by thread %d, Total threads
            in the parallel section %d\n",
            omp_get_thread_num(), omp_get_num_threads());
10    }
11  }
```

Schedule

schedule(static [,chunk])	Deal-out blocks of iterations of size "chunk" to each thread.
schedule(dynamic[,chunk])	Each thread grabs "chunk" iterations off a queue until all iterations have been handled.
schedule(guided[,chunk])	Threads dynamically grab blocks of iterations. The size of the block starts large and shrinks down to size "chunk" as the calculation proceeds.
schedule(runtime)	Schedule and chunk size taken from the OMP_SCHEDULE environment variable (or the runtime library).
schedule(auto)	Schedule is left up to the runtime to choose (does not have to be any of the above).

```
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;
```

```
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

```
1    int tnumber;
2    int i = 10, j = 10, k = 10;
3    printf("Before parallel region: i=%i, j=%i, k=%i\n",
         i, j, k);
4  #pragma omp parallel default(none) private(tnumber)
       reduction(+:i) reduction(*:j) reduction(^:k)
5    {
6      tnumber = omp_get_thread_num() + 1;
7      i = tnumber;
8      j = tnumber;
9      k = tnumber;
10     printf("Thread %i: i=%i, j=%i, k=%i\n", tnumber, i,
           j, k);
11   }
12   printf("After parallel region: i=%d, j=%d, k=%d\n", i
         , j, k);
```

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

```
#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;
     }
```

```
#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

```
#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);
}
```

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();
    }
}
```

```
1  int a = 6;
2  int b = 3;
3  omp_set_num_threads(4);
4  #pragma omp parallel
5  {
6    #pragma omp sections
7    {
8      #pragma omp section
9      {
10       printf("Sum = %d on thread %d \n", a + b,
             omp_get_thread_num());
11     }
12     #pragma omp section
13     {
14       printf("Difference = %d on thread %d \n", a - b,
             omp_get_thread_num());
15     }
16   }
17 }
```

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();
}
```

```
1   int i = 0, N = 8;
2   omp_set_num_threads(N);
3   int *a, *b, *c;
4   #pragma omp parallel
5   {
6     #pragma omp master
7     {
8       a = malloc(N * sizeof(int));
9       b = malloc(N * sizeof(int));
10      c = malloc(N * sizeof(int));
11      srand(time(NULL));
12    }
13    #pragma omp for
14    for( i  = 0; i < N; i++) {
15      a[i] = rand() % 10;
16      b[i] = rand() % 10;
17    }
18    #pragma omp for
19    for( i  = 0; i < N; i++) {
20        c[i] = a[i] * b[i];
21    }
22    #pragma omp for
23    for( i = 0; i < N ; i++) {
24      printf("A[%d] * B[%d] = %d \n", i, i, c[i]);
25    }
```

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

```
1   int i = 0, N = 8;
2   omp_set_num_threads(N);
3   int *a, *b, *c;
4   #pragma omp parallel
5   {
6     #pragma omp single
7     {
8       a = malloc(N * sizeof(int));
9       b = malloc(N * sizeof(int));
10      c = malloc(N * sizeof(int));
11      srand(time(NULL));
12    }
13    #pragma omp for
14    for( i  = 0; i < N; i++) {
15      a[i] = rand() % 10;
16      b[i] = rand() % 10;
17    }
18    #pragma omp for
19    for( i  = 0; i < N; i++) {
20        c[i] = a[i] * b[i];
21    }
22    #pragma omp for
23    for( i = 0; i < N ; i++) {
24      printf("A[%d] * B[%d] = %d \n", i, i, c[i]);
25    }
```

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”.