太原理工大学
TAIYUAN UNIVERSITY OF TECHNOLOGY

第5章
OpenMP实例—奇偶交换排序

岳俊宏
E-mail:yuejunhong@tyut.edu.cn; Tel:18234095983

目录

奇偶交换排序

冒泡排序

```
for (list_length = n; list_length >= 2; list_length--)
    for (i = 0; i < list_length-1; i++)
        if (a[i] > a[i+1]) {
            tmp = a[i];
            a[i] = a[i+1];
            a[i+1] = tmp;
        }
```

串行奇偶交换排序

奇偶交换排序是冒泡排序的一个变种，该算法更适合并行化。

Phase	Subscript in Array							
	0		1		2		3	
0	9	↔	7		8	↔	6	
	7		9		6		8	
1	7		9	↔	6		8	
	7		6		9		8	
2	7	↔	6		9	↔	8	
	6		7		8		9	
3	6		7	↔	8		9	
	6		7		8		9	

串行奇偶交换排序

```
void Odd_even_sort(
        int  a[]  /* in/out */,
        int  n    /* in     */) {
    int phase, i, temp;

    for (phase = 0; phase < n; phase++)
        if (phase % 2 == 0) { /* Even phase */
            for (i = 1; i < n; i += 2)
                if (a[i-1] > a[i]) {
                    temp = a[i];
                    a[i] = a[i-1];
                    a[i-1] = temp;
                }
        } else { /* Odd phase */
            for (i = 1; i < n-1; i += 2)
                if (a[i] > a[i+1]) {
                    temp = a[i];
                    a[i] = a[i+1];
                    a[i+1] = temp;
                }
        }
}  /* Odd_even_sort */
```

OpenMP-奇偶排序第1个版本

```
    for (phase = 0; phase < n; phase++) {
        if (phase % 2 == 0)
#           pragma omp parallel for num_threads(thread_count) \
               default(none) shared(a, n) private(i, tmp)
            for (i = 1; i < n; i += 2) {
               if (a[i-1] > a[i]) {
                  tmp = a[i-1];
                  a[i-1] = a[i];
                  a[i] = tmp;
               }
            }
        else
#           pragma omp parallel for num_threads(thread_count) \
               default(none) shared(a, n) private(i, tmp)
            for (i = 1; i < n-1; i += 2) {
               if (a[i] > a[i+1]) {
                  tmp = a[i+1];
                  a[i+1] = a[i];
                  a[i] = tmp;
               }
            }
    }
```

OpenMP-奇偶排序第2个版本

```
#   pragma omp parallel num_threads(thread_count) \
      default(none) shared(a, n) private(i, tmp, phase)
    for (phase = 0; phase < n; phase++) {
      if (phase % 2 == 0)
#        pragma omp for
        for (i = 1; i < n; i += 2) {
          if (a[i-1] > a[i]) {
            tmp = a[i-1];
            a[i-1] = a[i];
            a[i] = tmp;
          }
        }
      else
#        pragma omp for
        for (i = 1; i < n-1; i += 2) {
          if (a[i] > a[i+1]) {
            tmp = a[i+1];
            a[i+1] = a[i];
            a[i] = tmp;
          }
        }
    }
```

奇偶排序并行程序时间

double omp_get_wtime(void): 返回从某个特殊点所经过的时间，单位秒;(这个时间点在程序运行过程中必须是一致性)

```
int main(int argc, char* argv[]) {
   int  n;
   int* a;
   double start, finish;

   a = malloc(n*sizeof(int));
   Read_list(a, n);

   start = omp_get_wtime();
   Odd_even_sort(a, n);
   finish = omp_get_wtime();
   printf("Elapsed time = %e seconds\n", finish - start);
   free(a);
   return 0;
}  /* main */
```

奇偶排序并行程序时间

奇偶交换排序的时间：

- ✓ 方案1：用两条parallel for语句
- ✓ 方案2：用两条for语句运行

thread_count	1	2	3	4
Two parallel **for** directives	0.770	0.453	0.358	0.305
Two **for** directives	0.732	0.376	0.294	0.239

(Times are in seconds)

方案2比方案1快17%


循环调度

schedule子句——任务调度

OpenMP中，任务调度主要用于for循环的并行。当循环中每次迭代的计算量不等时，如果简单地给各个线程分配相同次数的迭代，则会造成各个线程计算负载不均衡，使得有些线程先执行完、有些后执行完，造成某些CPU核空闲，影响程序性能。

schedule子句——任务调度

如采用parallel for并行如下代码：

```
sum = 0.0;
for (i = 0; i <= n; i++)
    sum += f(i);

double f(int i) {
   int j, start = i*(i+1)/2, finish = start + i;
   double return_val = 0.0;

   for (j = start; j <= finish; j++) {
      return_val += sin(j);
   }
   return return_val;
} /* f */
```

由于f 函数调用所需时间与参数i的大小成正比，所以块划分不是最优的。

schedule子句——任务调度

- 更好的方案是轮流分配线程的工作：

Thread	Iterations
0	$0, n/t, 2n/t, \ldots$
1	$1, n/t+1, 2n/t+1, \ldots$
⋮	⋮
$t-1$	$t-1, n/t+t-1, 2n/t+t-1, \ldots$

循环划分任务.

OpenMP中，对for循环并行化的任务调度可使用schedule子句来实现。

schedule子句

schedule子句有如下形式：

schedule(<type[,<chunksize>]>)

type的类型：

static: 迭代能够在循环执行前分配给线程

dynamic或guided: 迭代在循环执行时被分配给线程

auto: 编译器和运行时系统决定调度方式

runtime: 调度在运行时决定

chunksize是一个正整数；在OpenMP中，迭代块是在顺序循环中连续执行的一块迭代语句，块中的迭代次数是chunksize.

schedule子句—— static

static：系统以轮转的方式将任务分配给线程；迭代能够在循环执行前分配给线程；

```
schedule(static,1)
```

Thread 0： 0,3,6,9

Thread 1： 1,4,7,10

Thread 2： 2,5,8,11

schedule子句—— static

```
schedule(static,2)
```

Thread 0 : 0, 1, 6, 7

Thread 1 : 2, 3, 8, 9

Thread 2 : 4, 5, 10, 11

schedule子句—— static

```
schedule(static,4)
```

Thread 0 : 0, 1, 2, 3

Thread 1 : 4, 5, 6, 7

Thread 2 : 8, 9, 10, 11

schedule子句—— dynamic

- dynamic或guided迭代在循环执行时被分配给线程，因此在一个线程完成了它的当前迭代集合后，它能从运行时系统中请求更多。
- 迭代被分成chunksize大小的连续迭代任务块
- 每个线程执行一块，执行完之后，将向系统请求另一块
- chunksize默认为1

schedule子句—— guided

- 每个线程执行一块，执行完之后，将向系统请求另一块
- 新块大小会变化，越来越小

Thread	Chunk	Size of Chunk	Remaining Iterations
0	1 – 5000	5000	4999
1	5001 – 7500	2500	2499
1	7501 – 8750	1250	1249
1	8751 – 9375	625	624
0	9376 – 9687	312	312
1	9688 – 9843	156	156
0	9844 – 9921	78	78
1	9922 – 9960	39	39
1	9961 – 9980	20	19
1	9981 – 9990	10	9
1	9991 – 9995	5	4
0	9996 – 9997	2	2
1	9998 – 9998	1	1
0	9999 – 9999	1	0

schedule子句—— runtime

- runtime调度在运行时决定；
- 当 schedule(runtime) 指定时，系统使用环境变量 OMP_SCHEDULE 在运行时来决定如何调度循环。OMP_SCHEDULE可能会呈现任何能被static、dynamic或guided调度所使用的值。
- export OMP_SCHEDULE=“static,1”

schedule子句—— 调度方式

- 这四种调度类型实际上只有static、dynamic、guided三种调度方式，runtime实际上是根据环境变量来选择前三种类型中的某一种。
- chunksize参数表示循环迭代次数，size参数必须是整数。static、dynamic、guided三种调度方式都可以使用size参数，也可以不使用。当type参数类型为runtime时， chunksize参数是非法的。

schedule子句——调度选择

- 调度开销：
- guided>dynamic>static
- 如果我们断定默认的调度方式性能低下，那么我们会做大量的试验来寻找最优的调度方式和迭代次数。


环境变量

OpenMP环境变量

- **环境变量**是能够被运行时系统所访问的命名值，即它们在程序的环境中是可得的。
  - ✓ PATH、HOME、SHELL
- OpenMP的环境变量主要有四个：
  - ✓ OMP_DYNAMIC、
  - ✓ OMP_NUM_THREADS、
  - ✓ OMP_NESTED、
  - ✓ OMP_SCHEDULE.

OMP_DYNAMIC

OMP_DYNAMIC环境变量:

FALSE：允许函数omp_set_num_thread()或者num_threads子句设置线程的数量；

TRUE：那么运行时会根据系统资源等因素进行调整，一般而言，生成与CPU数量相等的线程就是最好的利用资源了。

使用export命令设置环境变量值

export OMP_DYNAMIC=TRUE

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char* argv[]){
int j=0;
#pragma omp parallel num_threads(4)
{
#pragma omp for
        for(j=0;j<4;j++)

printf("j=%d,ThreadId=%d\n",j,omp_get_thread_num());
}
return 0;
}
```

```
[hdusr@Node1 ch5]$ export OMP_DYNAMIC=TRUE
[hdusr@Node1 ch5]$ ./omp_for
j=0,ThreadId=0
j=1,ThreadId=0
j=2,ThreadId=0
j=3,ThreadId=0
[hdusr@Node1 ch5]$ export OMP_DYNAMIC=false
[hdusr@Node1 ch5]$ ./omp_for
j=1,ThreadId=1
j=0,ThreadId=0
j=3,ThreadId=3
j=2,ThreadId=2
```

OMP_NUM_THREADS

OMP_NUM_THREADS环境变量主要用来设置parallel并行区域的默认线程数量。

注：必须OMP_DYNAMIC为FALSE时，OMP_NUM_THREADS环境变量才起作用。

使用export命令设置线程个数为4

export OMP_NUM_THREADS=4

OMP_NUM_THREADS

指定线程数目的方法

- ✓ 不指定，即默认为处理器的核数；
- ✓ 使用库函数omp_set_num_threads（int num）
- ✓ 在#parama omp parallel num_threads(num)
- ✓ 使用环境变量

```
$gcc –fopenmp  -o 1 1.c

$OMP_NUM_THREADS =4

$export OMP_NUM_THREADS//设置环境变量值
```

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char* argv[]){
int j=0;
#pragma omp parallel
{
#pragma omp for
        for(j=0;j<4;j++)

printf("j=%d,ThreadId=%d\n",j,omp_get_thread_num());
}
return 0;
}
```

```
[hdusr@Node1 ch5]$ gcc -g -Wall -fopenmp -o omp_for omp_for.c
[hdusr@Node1 ch5]$ export OMP_DYNAMIC=TRUE
[hdusr@Node1 ch5]$ export OMP_NUM_THREADS=4
[hdusr@Node1 ch5]$ ./omp_for
j=0,ThreadId=0
j=1,ThreadId=0
j=2,ThreadId=0
j=3,ThreadId=0
[hdusr@Node1 ch5]$ export OMP_DYNAMIC=FALSE
[hdusr@Node1 ch5]$ export OMP_NUM_THREADS=4
[hdusr@Node1 ch5]$ ./omp_for
j=1,ThreadId=1
j=0,ThreadId=0
j=3,ThreadId=3
j=2,ThreadId=2
```

OMP_NESTED

OMP_NESTED为TURE时，将启动嵌套并行。可以使用omp_set_nested()函数用参数0调用来停止嵌套并行。

OMP_SCHEDULE

- OMP_SCHEDULE主要是用来设置调度类型，只有在schedule子句的参数为runtime时才有效。

✓ 使用环境变量指定调度方式的方法

```
$gcc –fopenmp  -o 1 1.c

$OMP_SCHEDULE = “static,1”

$export OMP_SCHEDULE
```

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char* argv[]){
int j=0;

#pragma omp parallel for schedule(runtime)
        for(j=0;j<4;j++)
           printf("j=%d,ThreadId=%d\n",j,omp_get_thread_num());
return 0;
}
```

```
[hdusr@Node1 ch5]$ gcc -g -Wall -fopenmp -o omp_schedule omp_schedule.c
[hdusr@Node1 ch5]$ export OMP_NUM_THREADS=2
[hdusr@Node1 ch5]$ export OMP_SCHEDULE="static,1"
[hdusr@Node1 ch5]$ ./omp_schedule
j=1,ThreadId=1
j=3,ThreadId=1
j=0,ThreadId=0
j=2,ThreadId=0
[hdusr@Node1 ch5]$ export OMP_SCHEDULE="static,2"
[hdusr@Node1 ch5]$ ./omp_schedule
j=2,ThreadId=1
j=3,ThreadId=1
j=0,ThreadId=0
j=1,ThreadId=0
```

```c
#include <stdio.h>
#include <omp.h>
#include <unistd.h>

int main(int argc, char* argv[]){
int j=0;

#pragma omp parallel for schedule(runtime)
        for(j=0;j<8;j++){
          sleep(1);
          printf("j=%d,ThreadId=%d\n",j,omp_get_thread_num());}
return 0;
}
```

```
[hdusr@Node1 ch5]$ nano omp_schedule.c
[hdusr@Node1 ch5]$ gcc -g -Wall -fopenmp -o omp_schedule omp_schedule.c
[hdusr@Node1 ch5]$ export OMP_NUM_THREADS=3
[hdusr@Node1 ch5]$ export OMP_SCHEDULE="guided"
[hdusr@Node1 ch5]$ ./omp_schedule
j=3,ThreadId=0
j=5,ThreadId=2
j=0,ThreadId=1
j=4,ThreadId=0
j=1,ThreadId=1
j=6,ThreadId=2
j=7,ThreadId=0
j=2,ThreadId=1
```

```
[hdusr@Node1 ch5]$ export OMP_SCHEDULE="dynamic,2"
[hdusr@Node1 ch5]$ ./omp_schedule
j=2,ThreadId=0
j=4,ThreadId=1
j=0,ThreadId=2
j=3,ThreadId=0
j=5,ThreadId=1
j=1,ThreadId=2
j=6,ThreadId=0
j=7,ThreadId=0
```

内部控制变量

对于环境变量，在OpenMP的内部实现时有与之对应的内部控制变量。内部控制变量主要有5种：

nthreads-var：用于存储进入parallel区域的线程数量

dyn-var:用于控制是否允许动态调整使用的线程数量

nest-var：用于控制是否允许嵌套并行。

run-sched-var：用于存储当schedule子句带runtime参数时循环区域的调度信息。

def-sched-var:用于存储执行时的循环区域缺省调度信息。

内部控制变量

内部控制变量对程序员而言是不可见的，但是，了解这些内部控制变量可以帮助程序员更好地了解OpenMP的环境变量及对应的库函数的含义。

内部控制变量	环境变量	操作的库函数或子句	初始值
nthreads-var	OMP_NUM_THREADS	omp_set_num_threads() omp_get_max_threads() num_threads 子句	由实现定义
dyn-var	OMP_DYNAMIC	omp_set_dynamic() omp_get_dynamic()	由实现定义
nest-var	OMP_NESTED	omp_set_nested() omp_get_nested()	false
run-sched-var	OMP_SCHEDULE	无	由实现定义
def-sched-var	无	无	由实现定义

小结

- 指令：
  - ✓ for: 使for循环被多个线程并行执行;
  - ✓ parallel for: 使for循环的代码被多个线程并行执行。
- 子句：
  - ✓ schedule：调度任务实现分配给线程任务的不同划分方式
- 函数：
  - ✓ omp_get_wtime()：计算OpenMP并行程序花费时间
  - ✓ omp_set_num_thread()： 设置线程的数量

作业

1. 用OpenMP编程实现并行奇偶交换排序
2. 熟练掌握循环调度的方式

结束!

```
[hdusr@Node1 mpi-work]$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/hdusr/mpi-install/bin:/home/h
dusr/.local/bin:/home/hdusr/bin
```

```
[hdusr@Node1 ~]$ pwd Home
/home/hdusr
```

```
[hdusr@Node1 ch5]$ export OMP_DYNAMIC=false
[hdusr@Node1 ch5]$ ./omp_for
j=1,ThreadId=1
j=0,ThreadId=0
j=3,ThreadId=3
j=2,ThreadId=2
[hdusr@Node1 ch5]$ nano omp_for.c
[hdusr@Node1 ch5]$ gcc -g -Wall -fopenmp -o omp_for omp_for.c
[hdusr@Node1 ch5]$ ./omp_for
j=0,ThreadId=0
j=1,ThreadId=0
j=2,ThreadId=0
j=3,ThreadId=0
[hdusr@Node1 ch5]$ export OMP_NUM_THREADS=4
[hdusr@Node1 ch5]$ ./omp_for
j=3,ThreadId=3
j=0,ThreadId=0
j=1,ThreadId=1
j=2,ThreadId=2
[hdusr@Node1 ch5]$ export OMP_DYNAMIC=false
[hdusr@Node1 ch5]$ export OMP_DYNAMIC=TRUE
[hdusr@Node1 ch5]$ ./omp_for
j=0,ThreadId=0
j=1,ThreadId=0
j=2,ThreadId=0
j=3,ThreadId=0
```