并行程序设计与算法第四次作业

April 14, 2024

1 简答题

习题 1 中译版本的题目有歧义，原题参考英文版教材p263 习题5.5

假定在 Bleeblon 计算机上，浮点型变量能够存储小数点后 3 位数字，它的浮点寄存器可以存储小数点后 4 位，并且在任意的浮点操作后，结果在存储前被四舍五入为小数点后 3 位。现在假设一个 C 程序声明了一个数组:

float a[]=4.0.3.0.3.0，1000.0;

考虑如下代码:

```
int i;
float sum=0.0;
# pragma omp parallel for num_threads(2) \
        reduction(+:sum)
    for(i=0:i< 4:i++)
        sum += a[i];
        printf("sum=%4.1f\n".sum);
```

假设系统讲迭代 i=0, 1 分配给线程 0，将迭代 i=2,3 分配给线程 1，哪里在 Bleeblon 计算机上，该程序的输出是什么?

答案

使用 **parallel for** 来进行加法时，运行时系统将为每个线程创建一个私有变量，该私有变量将用于存储该线程的部分和。设我们在线程 0 上调用这些私有变量 local_sum0，在线程 1 上调用 local_sum1。在线程 0 完成其迭代之后，local_sum0 = 4.00e+00。在线程 1 完成其迭代之后，local_sum1 = 1.00e+03，因为寄存器和 1.004e+03 将被四舍五入。现在，当两个私有变量被添加时，存储在寄存器中的值将是 1.004e+03，当它存储在主存中时，我们的总和将是 = 1.00e+03。

因此，代码的输出将是和 = 1000.0。

习题 2

考虑循环

```
a[o] = 0;
for (i=1;i < n; i++)
    a[i] = a[i-1] + 1;
```

在该程序中存在循环依赖。
(1) 分析该程序中存在的循环依赖，并设计改写程序消除此依赖
(2) 加入 openmp 指令，对改写后的程序并行化

答案

(1) 分析可得，$a[i] = \sum_{j=0}^{i} j = i * (i+1)/2$
故消除依赖后的程序为：

```
for (i=0;i < n; i++)
    a[i] = i*(i+1)/2;
```

(2) 使用 parallel for 进行并行化：

```
# pragma omp parallel for num_threads(thread_count) \
    default(none) private(i) shared(a,n)
    for (i=0;i < n; i++)
        a[i] = i*(i+1)/2;
```

习题 3

我们考察 8000x8000 作为之前的矩阵-向量乘法程序的输人时该程序的性能。如果一个缓存行包含 64 字节或者 8 个双精度数，将输入向量表示为 y，那么:

1. 假定线程 0 和线程 2 被分配给了不同的处理器，在线程 0 和线程 2 之间的伪共享（false-sharing）可不可能在向量 y 上发生? 为什么?

2. 如果线程 0 和线程 3 被分配给了不同的处理器，那么伪共享（false-sharing）可不可能发生在向量 y 的任何地方?

答案

需要发生伪共享（false-sharing）的前提条件是需要有 y 中的元素是属于同一个 cache line ，但被花费给了不同的线程。在题设条件下，数据在四个线程内的划分为：

Thread 0: y[0], y[1], ..., y[1999]

Thread 1: y[2000], y[2001], ..., y[3999]

Thread 2: y[4000], y[4001], ..., y[5999]

Thread 3: y[6000], y[6001], ..., y[7999]

因此 thread 0 和 thread 2 中最近的向量元素为 y[1999] 和 y[4000]。因为一个缓存行仅能包含 8 个浮点数，故其不可能出现在用一个 cache line 中，故 thread 0 和 thread 2 不会发生伪共享。

thread 0 和 thread 3 同理，也不会发生伪共享。

习题 4

使用一维数组和 openmp 指令来实现并行的矩阵-向量乘法，其中矩阵为 float A[m*n]，输入向量为 float x[n]，结果向量为 float y[m]（手写代码给出关键的循环部分即可）

答案

```
# pragma omp parallel for num_thread(thread_count) \
    default(none) private(i,j) shared (A,x,y,m,n)
    for (i = 0;i<m;i++)
    y[i] = 0.0;
    for (j = 0;j<n;j++)
        y[i] += A[i*n+j] + x[j];
```