

軟體分析與最佳化 HW6

612410017 林靖紳

Q1: 下面為OpenMP程式碼片段，OpenMP parallel region將產生數個thread，請說明每個thread的執行內容為何？

```
1 void v_add(double* x, double* y, double* z) {  
2     #pragma omp parallel  
3     {  
4         for(int i=0; i<ARRAY_SIZE; i++)  
5             z[i] = x[i] + y[i];  
6     }  
7 }
```

- 每個執行緒將進入平行區域 (`#pragma omp parallel`)
- 每個執行緒都會獨立執行整個 for 迴圈，不需要等待其他執行緒完成
- 在 for 迴圈中，每個執行緒將執行： $z[i] = x[i] + y[i]$
 - 將陣列 x 和 y 對應位置的元素相加，並把結果存在陣列 z 中
- OpenMP會根據可用的執行緒數目自動分配工作，確保每個執行緒處理的 i 的範圍是公平的
- 總的來說，每個執行緒都專注處理 for 迴圈中的一部分，這樣就能夠同時、並行地完成陣列元素相加的工作，提高效能

Q2: 下面為OpenMP程式碼片段，OpenMP parallel region將產生數個thread，請說明每個thread的執行內容為何？

```
1 void v_add(double* x, double* y, double* z) {  
2     #pragma omp parallel for  
3     for(int i=0; i<ARRAY_SIZE; i++)  
4         z[i] = x[i] + y[i];  
5 }
```

- `#pragma omp parallel for` 告訴編譯器將 for 迴圈平行化
 - 將 for 分割成多個子任務，每個執行緒處理其中一部分。
- 每個執行緒將獨立處理 for 迴圈的一部分，其 i 的範圍會根據OpenMP的自動劃分而定，確保工作平均分配。
- for 迴圈中的操作是 $z[i] = x[i] + y[i]$
 - 將陣列 x 和 y 對應位置的元素相加，並把結果存在陣列 z 中
- 總的來說，這個設計讓多個執行緒同時處理陣列元素的相加操作，每個執行緒處理 for 迴圈的一部分，從而提高整體程式的效能

Q1 與 Q2 的區別

- 平行化的區域：
 - Q1: `#pragma omp parallel` 放在整個函式內，表示整個函式內的程式碼都是平行執行的
 - Q2: `#pragma omp parallel for` 放在 for 迴圈前面，僅對該迴圈進行平行化
- 平行化的精度：
 - Q1: 每個執行緒執行整個 for 迴圈，並使用同一份的 x、y 和 z。
 - Q2: for 迴圈的每次迭代被不同的執行緒處理，每個執行緒負責處理 for 迴圈的一部分，並使用相同的 x、y 和獨立的部分 z。

Q3: 請利用OpenMP改寫下面的程式碼片段，平行化for-loop的執行，同時以滿足下列的要求:

- OpenMP parallel region會產生4個 work threads
- 4個thread執行的運算內容如下

<code>z[0] = x[0] + y[0]</code> <code>z[1] = x[1] + y[1]</code> <code>z[2] = x[2] + y[2]</code> <code>z[3] = x[3] + y[3]</code> ...	<code>z[25] = x[25] + y[25]</code> <code>z[26] = x[26] + y[26]</code> <code>z[27] = x[27] + y[27]</code> <code>z[28] = x[28] + y[28]</code> ...	<code>z[50] = x[50] + y[50]</code> <code>z[51] = x[51] + y[51]</code> <code>z[52] = x[52] + y[52]</code> <code>z[53] = x[53] + y[53]</code> ...	<code>z[75] = x[75] + y[75]</code> <code>z[76] = x[76] + y[76]</code> <code>z[77] = x[77] + y[77]</code> <code>z[78] = x[78] + y[78]</code> ...
thread a	thread b	thread c	thread d

```
void v_add(double* x, double* y, double* z) {  
    for(int i=0; i<100; i++)  
        z[i] = x[i] + y[i];  
}
```

```
1 void v_add(double* x, double* y, double* z) {  
2     #pragma omp parallel num_threads(4)  
3     {  
4         int thread_id = omp_get_thread_num();  
5         int chunk_size = 100 / 4;  
6         int start_index = thread_id * chunk_size;  
7         int end_index = (thread_id + 1) * chunk_size;  
8  
9         for (int i = start_index; i < end_index; i++) {  
10            z[i] = x[i] + y[i];  
11        }  
12    }  
13 }  
14
```

Q4: 請利用OpenMP改寫下面的程式碼片段，平行化for-loop的執行，同時以滿足下列的要求:

- OpenMP parallel region會產生4個 work threads
- 4個thread執行的運算內容如下:

```
z[0] = x[0] + y[0]
z[4] = x[4] + y[4]
z[8] = x[8] + y[8]
z[12] = x[12] + y[12]
...
```

thread a

```
z[1] = x[1] + y[1]
z[5] = x[5] + y[5]
z[9] = x[9] + y[9]
z[13] = x[13] + y[13]
...
```

thread b

```
z[2] = x[2] + y[2]
z[6] = x[6] + y[6]
z[10] = x[10] + y[10]
z[14] = x[14] + y[14]
...
```

thread c

```
z[3] = x[3] + y[3]
z[7] = x[7] + y[7]
z[11] = x[11] + y[11]
z[15] = x[15] + y[15]
...
```

thread d

```
void v_add(double* x, double* y, double* z) {
    for(int i=0; i<100; i++)
        z[i] = x[i] + y[i];
}
```

```
1 void v_add(double* x, double* y, double* z) {
2     #pragma omp parallel num_threads(4)
3     {
4         int thread_id = omp_get_thread_num();
5         int chunk_size = 100 / 4;
6
7         for (int i = 0; i < chunk_size; i++) {
8             int index = i * 4 + thread_id;
9             if (index <= 100) {
10                z[index] = x[index] + y[index];
11            }
12        }
13    }
14 }
15
```

Q5: 請利用OpenMP改寫下面的程式碼片段，平行化for-loop的執行，同時以滿足下列的要求:

- OpenMP parallel region會產生4個 work threads
- 4個thread執行的運算內容如下:

```
z[0] = x[0] + y[0]
z[1] = x[1] + y[1]
z[8] = x[8] + y[8]
z[9] = x[9] + y[9]
...
```

thread a

```
z[2] = x[2] + y[2]
z[3] = x[3] + y[3]
z[10] = x[10] + y[10]
z[11] = x[11] + y[11]
...
```

thread b

```
z[4] = x[4] + y[4]
z[5] = x[5] + y[5]
z[12] = x[12] + y[12]
z[13] = x[13] + y[13]
...
```

thread c

```
z[6] = x[6] + y[6]
z[7] = x[7] + y[7]
z[14] = x[14] + y[14]
z[15] = x[15] + y[15]
...
```

thread d

```
void v_add(double* x, double* y, double* z) {
    for(int i=0; i<100; i++)
        z[i] = x[i] + y[i];
}
```

```
1 void v_add(double* x, double* y, double* z) {
2     #pragma omp parallel num_threads(4)
3     {
4         int thread_id = omp_get_thread_num();
5         for (int i = thread_id * 2; i < 100; i += 16) {
6             z[i] = x[i] + y[i];
7             z[i + 1] = x[i + 1] + y[i + 1];
8             z[i + 8] = x[i + 8] + y[i + 8];
9             z[i + 9] = x[i + 9] + y[i + 9];
10        }
11    }
12 }
13
```