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?

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

- 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?

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

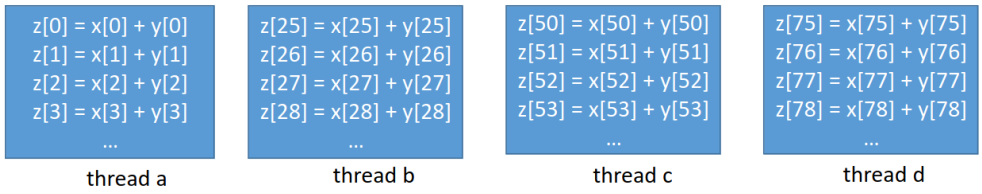
- `#pragma omp parallel for` 告訴編譯器將 for 迴圈平行化
  - 將 for 分割成多個子任務，每個執行緒處理其中一部分。
- 每個執行緒將獨立處理 for 迴圈的一部分，其 i 的範圍會根據OpenMP的自動劃分而定，確保工作平均分配。
- for 迴圈中的操作是 z[i] = x[i] + y[i]
  - 將陣列 x 和 y 對應位置的元素相加，並把結果存在陣列 z 中
- 總的來說，這個設計讓多個執行緒同時處理陣列元素的相加操作，每個執行緒處理 for 迴圈的一部分，從而提高整體程式的效能

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




```
void v_add(double* x, double* y, double* z) {

  for(int i=0; i<100; i++)
      z[i] = x[i] + y[i];
}
```

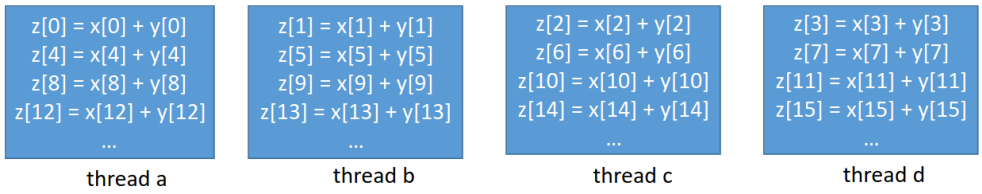
```
void v_add(double* x, double* y, double* z) {
    #pragma omp parallel num_threads(4)
    {
        int thread_id = omp_get_thread_num();
        int chunk_size = 100 / 4;
        int start_index = thread_id * chunk_size;
        int end_index = (thread_id + 1) * chunk_size;

        for (int i = start_index; i < end_index; i++) {
            z[i] = x[i] + y[i];
        }
    }
}

```

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:




```
void v_add(double* x, double* y, double* z) {

  for(int i=0; i<100; i++)
      z[i] = x[i] + y[i];
}
```

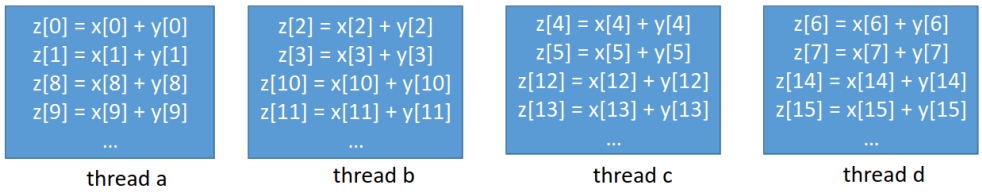
```
void v_add(double* x, double* y, double* z) {
    #pragma omp parallel num_threads(4)
    {
        int thread_id = omp_get_thread_num();
        int chunk_size = 100 / 4;

        for (int i = 0; i < chunk_size; i++) {
            int index = i * 4 + thread_id;
            if (index <= 100) {
                z[index] = x[index] + y[index];
            }
        }
    }
}

```

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:




```
void v_add(double* x, double* y, double* z) {

  for(int i=0; i<100; i++)
      z[i] = x[i] + y[i];
}
```

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

```