Activity 1

1.2.

```
[Running] cd "f:\Deakin Year3 Tasks and Assigments\SIT315\M2.S3P\M2.S3P-resources\" && g++ -std=c++17 ParallelVectorAddition.cpp -o
ParallelVectorAddition && "f:\Deakin Year3 Tasks and Assigments\SIT315\M2.S3P\M2.S3P-resources\"ParallelVectorAddition
Chunk size: 1000, Time taken by parallel function: 55352615 microseconds
Chunk size: 10000, Time taken by parallel function: 1429002 microseconds
Chunk size: 100000, Time taken by parallel function: 578434 microseconds
Chunk size: 1000000, Time taken by parallel function: 502744 microseconds

[Done] exited with code=0 in 59.163 seconds

[Running] cd "f:\Deakin Year3 Tasks and Assigments\SIT315\M2.S3P\M2.S3P-resources\" && g++ -std=c++17 OpenMPVectorAdd.cpp -o OpenMPVectorAdd &&
"f:\Deakin Year3 Tasks and Assigments\SIT315\M2.S3P\M2.S3P-resources\"OpenMPVectorAdd
Time taken by function with OpenMP: 2528551 microseconds

[Done] exited with code=0 in 3.196 seconds
```

In this test, the vector addition implemented using std::thread outperformed the OpenMP implementation when the chunk size was set to a larger value (1,000,000), with the former taking approximately 502,744 microseconds compared to OpenMP's 2,528,511 microseconds. This indicates that while OpenMP is generally easier to use and typically does not require manual tuning, a carefully optimized std::thread implementation can be more efficient in specific scenarios.

Activity 2

1.

Adding the default(none) clause in the OpenMP directive forces explicit declaration of how each variable is shared among threads. To prevent compilation errors, variables like the vector pointers (v1, v2, v3) should be declared as shared, as all threads need to access the same memory locations, while the loop index (i) should be private to ensure each thread operates on different elements. Incorrectly marking vector pointers as private would lead to each thread working on an uninitialized or separate copy of the vector, causing the program to produce incorrect results due to threads not updating the intended shared data.

2.

```
27      for (int i = 0; i < size; i++) {
28          v3[i] = v1[i] + v2[i];
29      }
30
31      // Initialize the total sum variable
32      long long total = 0;
33
34      // Compute the total sum using an atomic update
35      #pragma omp parallel for shared(total)
36      for (int i = 0; i < size; i++) {
37          #pragma omp atomic
38          total += v3[i];
39      }
40
41      cout << "Total sum of elements in v3: " << total << endl;
42
43      delete[] v1;
44      delete[] v2;
45      delete[] v3;
```

端口 SQL CONSOLE 问题 输出 调试控制台 终端 Code

```
[Running] cd "f:\Deakin Year3 Tasks and Assigments\SIT315\M2.S3P\M2.S3P-resources\" && g++ -std=c++17 OpenMPVectorAdd.cpp -o OpenMPVectorAdd &&
"f:\Deakin Year3 Tasks and Assigments\SIT315\M2.S3P\M2.S3P-resources\"OpenMPVectorAdd
Total sum of elements in v3: 9893034453

[Done] exited with code=0 in 3.485 seconds
```

3.

```
    randomVector(v1, size);
    randomVector(v2, size);

    #pragma omp parallel for
    for (int i = 0; i < size; i++) {
        v3[i] = v1[i] + v2[i];
    }

    // Compute the total sum using the reduction clause
    long long total = 0;

    #pragma omp parallel for reduction(+:total)
    for (int i = 0; i < size; i++) {
        total += v3[i];
    }

    cout << "Total sum of elements in v3 using reduction: " << total << endl;
```

端口 SQL CONSOLE 问题 输出 调试控制台 终端 Code

```
[Running] cd "f:\Deakin Year3 Tasks and Assigments\SIT315\M2.S3P\M2.S3P-resources\" && g++ -std=c++17 OpenMPVectorAdd.cpp -o OpenMPVectorAdd &&
"f:\Deakin Year3 Tasks and Assigments\SIT315\M2.S3P\M2.S3P-resources\"OpenMPVectorAdd
Total sum of elements in v3 using reduction: 9893006248

[Done] exited with code=0 in 3.366 seconds
```

4.

```
    // Reduction method
    long long total_reduction = 0;
    #pragma omp parallel for reduction(+:total_reduction)
    for (int i = 0; i < size; i++) {
        total_reduction += v3[i];
    }

    // Critical section method
    long long total_critical = 0;
    #pragma omp parallel
    {
        long long partial_sum = 0;

        #pragma omp for
        for (int i = 0; i < size; i++) {
            partial_sum += v3[i];
        }

        #pragma omp critical
        {
            total_critical += partial_sum;
        }
    }

    // Output both results for comparison
    cout << "Total sum of elements in v3 using reduction: " << total_reduction << endl;
    cout << "Total sum of elements in v3 using critical section: " << total_critical << endl;
```

端口 SQL CONSOLE 问题 输出 调试控制台 终端 Code

```
[Running] cd "f:\Deakin Year3 Tasks and Assigments\SIT315\M2.S3P\M2.S3P-resources\" && g++ -std=c++17 OpenMPVectorAdd.cpp -o OpenMPVectorAdd &&
"f:\Deakin Year3 Tasks and Assigments\SIT315\M2.S3P\M2.S3P-resources\"OpenMPVectorAdd
Total sum of elements in v3 using reduction: 9893236557
Total sum of elements in v3 using critical section: 9893236557

[Done] exited with code=0 in 3.616 seconds
```

5.

```
[Running] cd "f:\Deakin Year3 Tasks and Assigments\SIT315\M2.S3P\M2.S3P-resources\" && g++ -std=c++17 OpenMPVectorAdd.cpp -o OpenMPVectorAdd &&
"f:\Deakin Year3 Tasks and Assigments\SIT315\M2.S3P\M2.S3P-resources\"OpenMPVectorAdd
Scheduling: static, Chunk size: 1000, Time: 137 ms, Total: 9893163184
Scheduling: static, Chunk size: 10000, Time: 135 ms, Total: 9893163184
Scheduling: static, Chunk size: 100000, Time: 135 ms, Total: 9893163184
Scheduling: dynamic, Chunk size: 1000, Time: 136 ms, Total: 9893163184
Scheduling: dynamic, Chunk size: 10000, Time: 134 ms, Total: 9893163184
Scheduling: dynamic, Chunk size: 100000, Time: 133 ms, Total: 9893163184
Scheduling: guided, Chunk size: 1000, Time: 132 ms, Total: 9893163184
Scheduling: guided, Chunk size: 10000, Time: 132 ms, Total: 9893163184
Scheduling: guided, Chunk size: 100000, Time: 130 ms, Total: 9893163184

[Done] exited with code=0 in 4.295 seconds
```

Static scheduling produced nearly identical execution times across all chunk sizes, indicating that the workload was evenly distributed. Dynamic scheduling performed similarly, with only slight variations, suggesting minimal overhead from work reassignment. Guided scheduling showed a slight performance improvement, particularly with larger chunk sizes, due to its balanced approach of starting with larger chunks and gradually decreasing the size, thereby optimizing both load balancing and overhead.