Seminar 5- MultiThreading- OMP

Activity 1

Evaluated the performance of the three implementations using three different sizes. Thread Number was kept at 16 threads for both OpenMP and PTThread. The results were-

For size of 1000:

- Sequential- 22ms
- OpenMP- 21,234ms
- PTThread- 1196ms

Sequential performing best when size is 1000.

For size of 100,000:

- Sequential- 1868ms
- OpenMP- 29,830ms
- PT Thread- 2751ms

Sequential is yet again performing best when size is 100,000.

For size of 1,000,000,000:

- Sequential- 22,990,641ms
- OpenMP- 36,301,493ms
- Pt Thread- 12,052,122ms

PT Thread is performing best for such a large size of 1,000,000,000.

From the results above, it can be concluded that Sequential works best with smaller sizes. But as size grows larger, PT thread becomes a better choice. Meanwhile, OpenMP is performing the worst in all the size categories.

Activity 2

1. Default(none) does not allow any variables to be shared among the threads. That is why we are getting an error.
   Using shared(size): Gives compilation error. This is because we did not share total, v1,v2 and v3.
   Using shared(v1): Gives compilation error. This is because we did not share total, size,v2 and v3.
   Using shared(size,total,v1,v2,v3): This works as all the variables in the parallel program are shared now.

```
#pragma omp parallel default(none) shared(size, total, v1, v2, v3)
{
    int id = omp_get_thread_num();
    printf("\n Thread id is %d\n*************************\n", id);
 /*    #pragma omp single
      {
        cout << "Total thread are: " << omp_get_num_threads() << endl;
      } */

    #pragma omp for schedule(static, 10)
    for (size_t i = 0; i < size; i++)
    {
        //cout<<"\nindex i is: "<< i <<endl;
        v3[i] = v1[i] + v2[i];
        total += v3[i];
    }
}
```

Using private(total): Now the total variable from shared to private. This prevents the total value from getting incremented and I get 0 as the final output.

Conclusion: Only data shared between threads will be executed in the parallel program. Or else, we are bound to get different values.

2.

```
#pragma omp parallel default(none) shared(size, total, v1, v2, v3)
{
    int id = omp_get_thread_num();
    printf("\n Thread id is %d\n*************************\n", id);
 /*    #pragma omp single
      {
        cout << "Total thread are: " << omp_get_num_threads() << endl;
      } */

    #pragma omp for
    for (size_t i = 0; i < size; i++)
    {
        //cout<<"\nindex i is: "<< i <<endl;
        v3[i] = v1[i] + v2[i];
        #pragma omp atomic
        total += v3[i];
    }
}

auto stop = high_resolution_clock::now();

//ToDo: Add Comment
auto duration = duration_cast<microseconds>(stop - start);

cout << "Total value of addition: " << total << endl;
cout << "Time taken by function: "
     << duration.count() << " microseconds" << endl;

return 0;
```

```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

*************************

 Thread id is 1
*************************
Total value of addition: 99967
Time taken by function: 28467 microseconds
PS C:\Trimester 2 2021\M2.S3P-resources>
```

USING ATOMIC DIRECTIVE

3.

```
    #pragma omp parallel default(none) shared(size, v1, v2, v3) reduction(+:total)
    {
        int id = omp_get_thread_num();
        printf("\n Thread id is %d\n*************************\n", id);
     /*    #pragma omp single
        {
            cout << "Total thread are: " << omp_get_num_threads() << endl;
        } */

        #pragma omp for
        for (size_t i = 0; i < size; i++)
        {
            //cout<<"\nindex i is: "<< i <<endl;
            v3[i] = v1[i] + v2[i];

            total += v3[i];
        }
    }

    auto stop = high_resolution_clock::now();

    //ToDo: Add Comment
    auto duration = duration_cast<microseconds>(stop - start);

    cout << "Total value of addition: " << total << endl;
    cout << "Time taken by function: "
         << duration.count() << " microseconds" << endl;

    return 0;
}
```

PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

```
*************************

 Thread id is 7
*************************
Total value of addition: 100044
Time taken by function: 26269 microseconds
PS C:\Trimester 2 2021\M2.S3P-resources>
```

Úsing Reduction

4.

```cpp
    int privTotal;

     #pragma omp parallel default(none) shared(size, total, v1, v2, v3) firstprivate(privTotal)
    {
        int id = omp_get_thread_num();
        printf("\n Thread id is %d\n**************************\n", id);
      /*    #pragma omp single
           {
            cout << "Total thread are: " << omp_get_num_threads() << endl;
           } */

        #pragma omp for
        for (size_t i = 0; i < size; i++)
        {
            //cout<<"\nindex i is: "<< i <<endl;
            v3[i] = v1[i] + v2[i];

            privTotal += v3[i];
        }

        #pragma omp critical
        {
            total += privTotal;
        }
    }

    auto stop = high_resolution_clock::now();

    //ToDo: Add Comment
    auto duration = duration_cast<microseconds>(stop - start);

    cout << "Total value of addition: " << total << endl;
    cout << "Time taken by function: "
         << duration.count() << " microseconds" << endl;

    return 0;
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
**************************

 Thread id is 12
**************************
Total value of addition: 97984
Time taken by function: 26200 microseconds
PS C:\Trimester 2 2021\M2.S3P-resources>
```

Using Critical region

I get similar results which is close to exact for all the cases.

5. Only changing the scheduling technique to dynamic affected the execution time. For higher chunks, dynamic's performance was similar to static or guided. But for lower chunks, the execution time significantly increased. Below is the example when I used size 2 chunk and size 100 chunk for dynamic. For size 2 chunk execution time=876160. For size 100 chunk execution time=216559.

VectorAdd.cpp ×

VectorAdd.cpp > main()

```cpp

    #pragma omp parallel default(none) shared(size, total, v1, v2, v3) firstprivate(privTotal)
    {
        int id = omp_get_thread_num();
        printf("\n Thread id is %d\n*************************\n", id);
     /*    #pragma omp single
        {
            cout << "Total thread are: " << omp_get_num_threads() << endl;
        } */

        #pragma omp for schedule(dynamic, 2)
        for (size_t i = 0; i < size; i++)
        {
            //cout<<"\nindex i is: "<< i <<endl;
            v3[i] = v1[i] + v2[i];

            privTotal += v3[i];
        }

        #pragma omp critical
        {
            total += privTotal;
        }
    }

    auto stop = high_resolution_clock::now();

    //ToDo: Add Comment
    auto duration = duration_cast<microseconds>(stop - start);

    cout << "Total value of addition: " << total << endl;
    cout << "Time taken by function: "
         << duration.count() << " microseconds" << endl;

    return 0;
}
```

PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

```
*************************

 Thread id is 1
*************************
Total value of addition: 989341920
Time taken by function: 876160 microseconds
PS C:\Trimester 2 2021\M2.S3P-resources>
```

VectorAdd.cpp ×

VectorAdd.cpp > main()

```cpp

    #pragma omp parallel default(none) shared(size, total, v1, v2, v3) firstprivate(privTotal)
    {
        int id = omp_get_thread_num();
        printf("\n Thread id is %d\n*************************\n", id);
     /*    #pragma omp single
        {
            cout << "Total thread are: " << omp_get_num_threads() << endl;
        } */

        #pragma omp for schedule(dynamic, 100)
        for (size_t i = 0; i < size; i++)
        {
            //cout<<"\nindex i is: "<< i <<endl;
            v3[i] = v1[i] + v2[i];

            privTotal += v3[i];
        }

        #pragma omp critical
        {
            total += privTotal;
        }
    }

    auto stop = high_resolution_clock::now();

    //ToDo: Add Comment
    auto duration = duration_cast<microseconds>(stop - start);

    cout << "Total value of addition: " << total << endl;
    cout << "Time taken by function: "
         << duration.count() << " microseconds" << endl;

    return 0;
}
```

PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

```
*************************

 Thread id is 1
*************************
Total value of addition: 989329545
Time taken by function: 216559 microseconds
PS C:\Trimester 2 2021\M2.S3P-resources>
```