HPC OPENMP TUTORIAL 4 REPORT

CS22B2015 – HARSHITH B

The parallel code for reduction and critical sections is provided in the main.c file. The key section of the code performs the following steps:

1. Threads are allocated in powers of 2 (1, 2, 4, 8, 16, 32, 64, 128) within a loop, and the sum is calculated using the OpenMP pragma reduction.
2. Two datasets consisting of approximately 15 million double-precision floating-point numbers is generated using Python and stored in the output1.txt and output2.txt files.
3. The threads versus time data is recorded into a text file. This data is then processed, and the respective results are plotted and analyzed using results.py.

1. Parallel Code Segment for Multiplication and Addition

```c
printf("Vector Dot Product\n");
for(int t = 0; t < num_options; t++){
    int num_threads = thread_counts[t];
    omp_set_num_threads(num_threads);

    double dot_product = 0.0;
    start = omp_get_wtime();

    #pragma omp parallel
    {
        double local_sum = 0.0;
        #pragma omp for
        for(int i = 0; i < count; i++){
            local_sum += arr1[i] * arr2[i];
        }

        #pragma omp critical
        dot_product += local_sum;
    }

    end = omp_get_wtime();
    printf("%d\t%.6f\t%.12f\n", num_threads, end - start, dot_product);
    fprintf(f_dot, "%d %lf\n", num_threads, end - start);
}
```

2. Terminal Output

```
(venv) harshith@harshithb:~/Projects /SEM 6/HPC/tutorial-4$ ./main
Vector Dot Product
1       0.070798        3758873747356410368.000000000000
2       0.040675        3758873747355359232.000000000000
4       0.025236        3758873747355091968.000000000000
6       0.021679        3758873747355244544.000000000000
8       0.021532        3758873747355185664.000000000000
10      0.038120        3758873747355194368.000000000000
12      0.030346        3758873747355226624.000000000000
16      0.023518        3758873747355228672.000000000000
20      0.023104        3758873747355247104.000000000000
32      0.016972        3758873747355226624.000000000000
64      0.023649        3758873747355222016.000000000000
```

3. Thread vs Time Plots and Observations

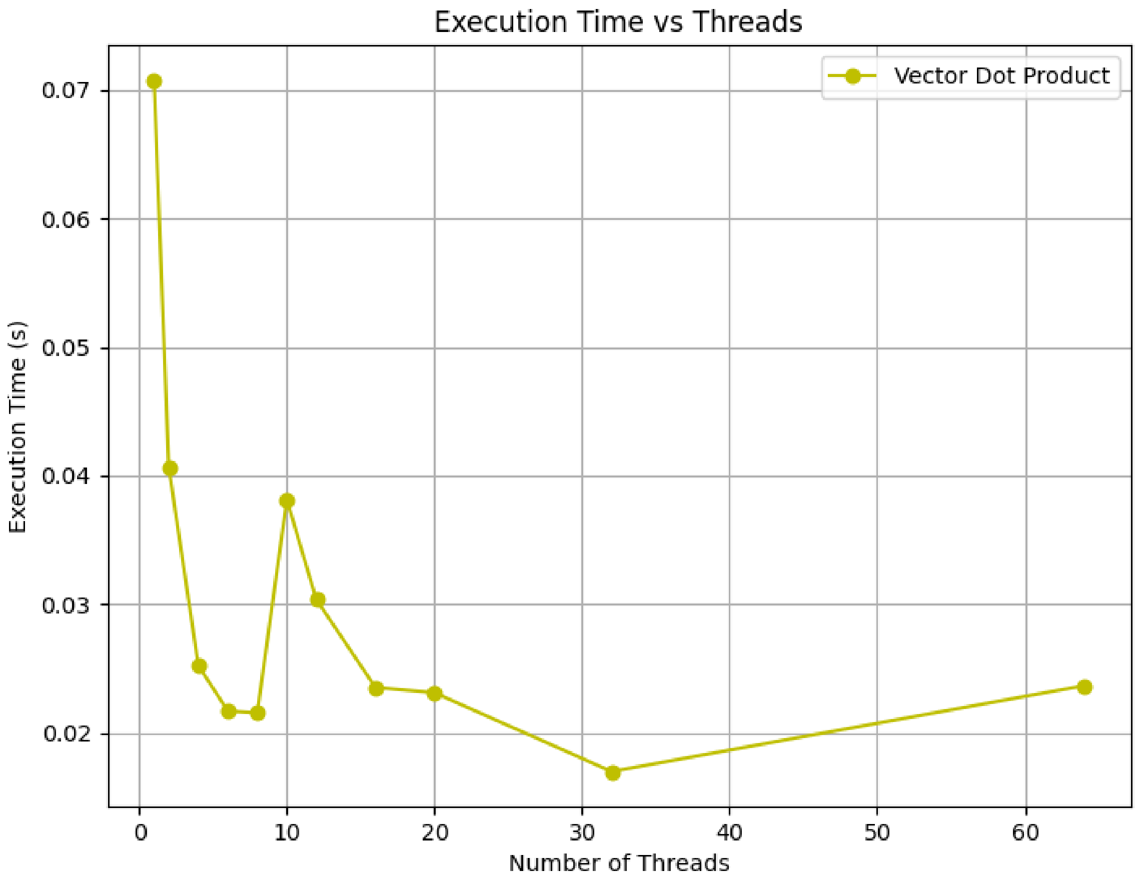
Vector Dot Product:

The execution time for the vector dot product decreases sharply as the number of threads increases from 1 to around 8 threads, indicating effective parallelization in this range.

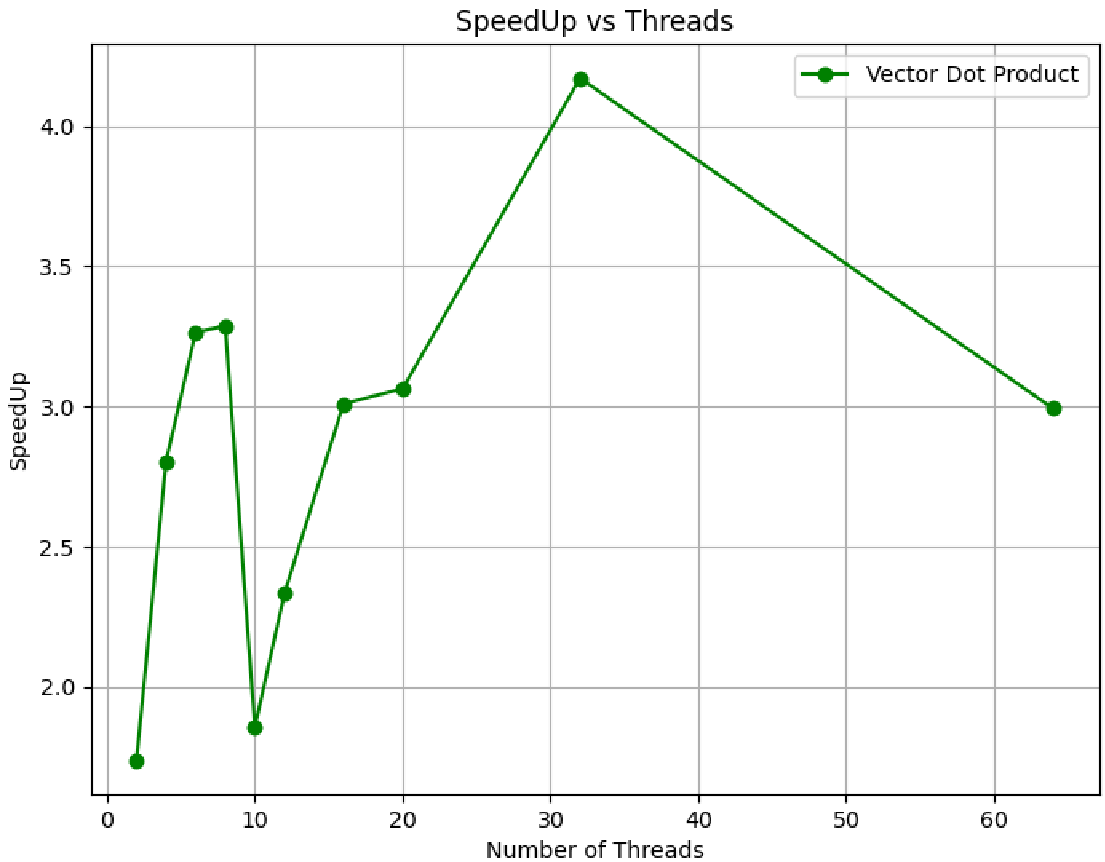
The best performance appears to occur around 32 threads, where the execution time is at its lowest.

Beyond 32 threads, the execution time increases slightly, likely due to overheads such as thread contention, synchronization, and communication costs among threads.

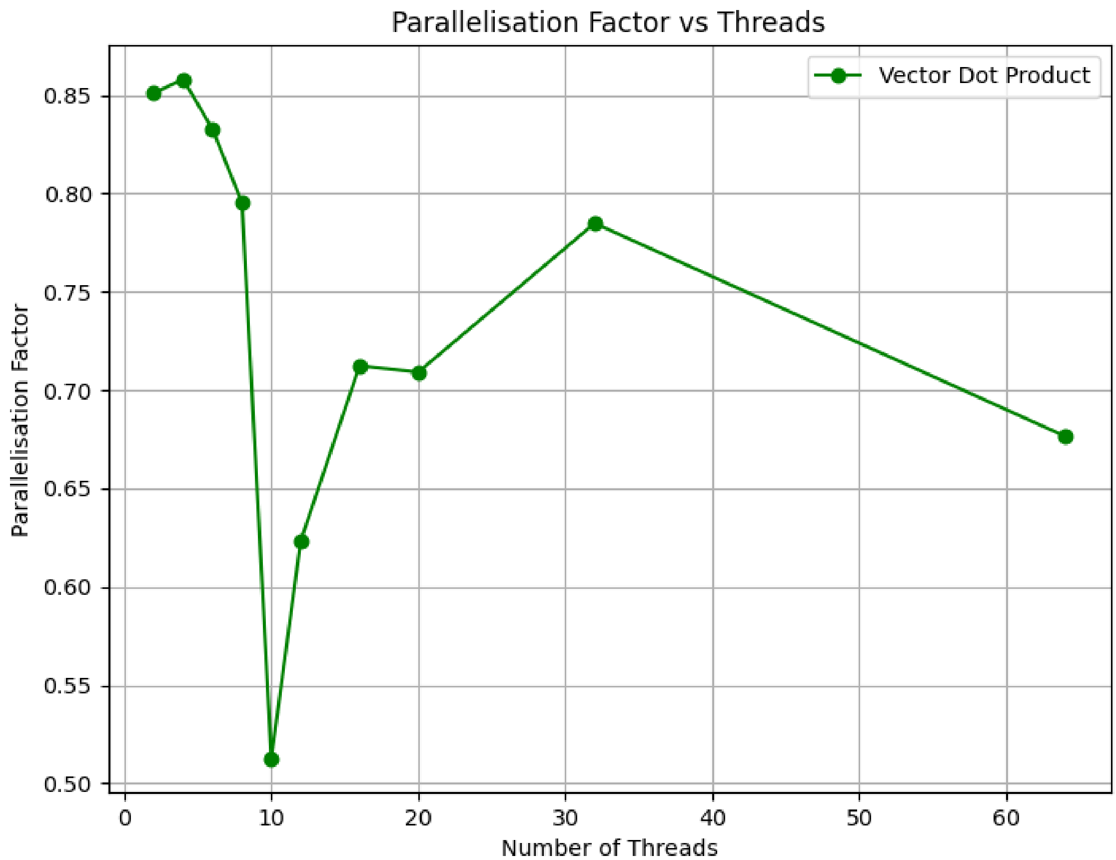
This suggests that the vector dot product scales well with increasing threads up to a point, after which diminishing returns and overhead costs start to dominate the performance.



4. SpeedUp vs Processors (SpeedUp == T(1) / T(n))



5. Parallelization Fraction and Inference



- ***Parallel Vector Dot Product:***

- The parallelization factor peaks initially but declines as the number of threads increases, stabilizing around **0.855**.
- This indicates diminishing returns for parallelization beyond a certain threshold (approximately 4 **threads**).
- Overhead due to thread management likely impacts performance at higher thread counts.