HPC Tutorial 6 Report
CS22B2012
K Aditya Sai

The code for Matrix Multiplication is implemented in C++ with OpenMP.

1. **Thread Allocation**: The code is executed with thread counts ranging from 1 to 64

2. **Matrix Generation**: A C++ code generates matrices of dimension 1000 x 1000 with double precision floating point elements ranging from 1-100

3. **Performance Analysis**: Execution times for the multiplication are recorded and speedup/parallelization fractions are calculated.

4. **Visualization**: Python scripts generate plots for execution time and speedup.

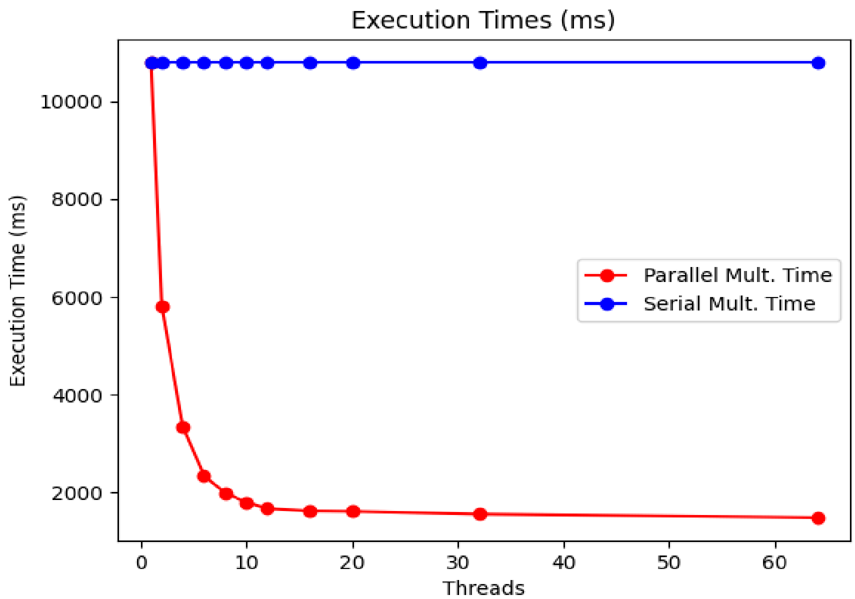
Parallel Code for Matrix Multiplication:

```
void multiplyMatricesSerial(const vector<vector<double>> &matrix1, const vector<vector<double>> &matrix2, vector<vector<double>> &result)
{
    for (int i = 0; i < SIZE; ++i)
    {
        for (int j = 0; j < SIZE; ++j)
        {
            result[i][j] = 0.0;
            for (int k = 0; k < SIZE; ++k)
            {
                result[i][j] += matrix1[i][k] * matrix2[k][j];
            }
        }
    }
}
```

Serial Code for Matrix Multiplication:

```
void multiplyMatricesParallel(const vector<vector<double>> &matrix1, const vector<vector<double>> &matrix2, vector<vector<double>> &result)
{
    #pragma omp parallel for collapse(2)
    for (int i = 0; i < SIZE; ++i)
    {
        for (int j = 0; j < SIZE; ++j)
        {
            result[i][j] = 0.0;
            for (int k = 0; k < SIZE; ++k)
            {
                result[i][j] += matrix1[i][k] * matrix2[k][j];
            }
        }
    }
}
```

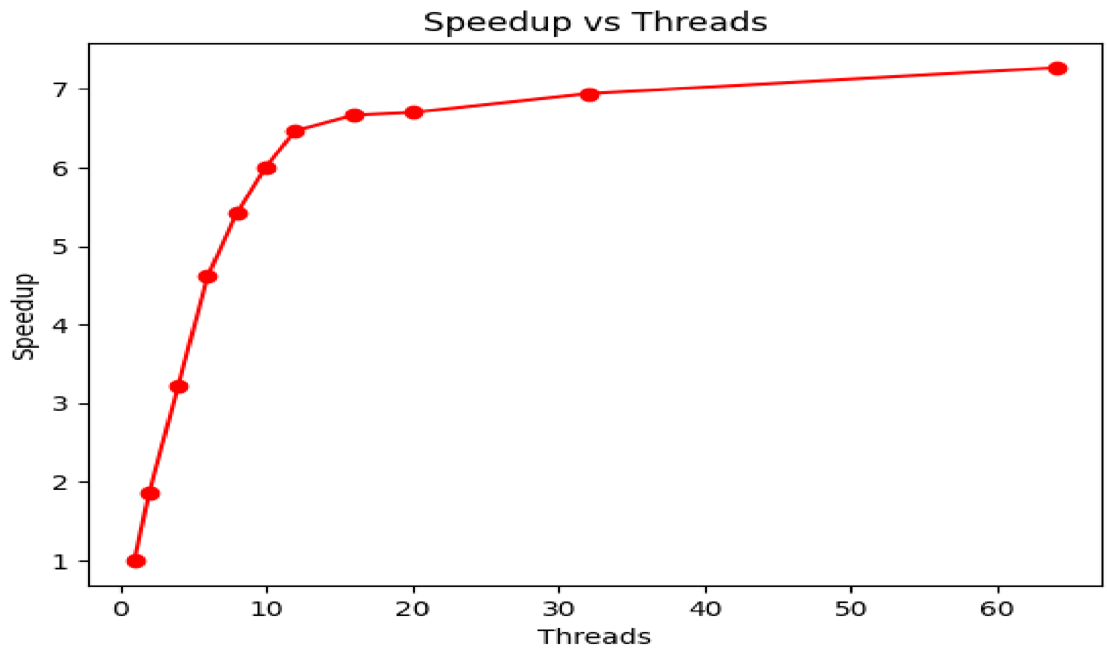
Plot Threads v/s Time :



Observations :

1. When increasing thread count from 2 to 12, execution time drops until it reaches 12 threads.
2. The multiplication and the thread management overhead would result in an increase in the execution time.
3. However, we see that due to the large size of the problem, the execution time remains similar for threads ranging from 20 to 64 but seems to be on a decline.
4. This suggests that the thread context switch is not as dominant compared to the computational cost of matrix multiplication.

Plot Threads v/s Speedup:



Observations :

1. The sudden spike in the overall speedup from 1 thread to 12 threads shows that for the system on which this was performed, 12 threads is the most optimal in case of matrix multiplication.
2. The speedup does not appear to have a steady ascent as the number of threads increases.
3. The parallel construct seems to efficiently parallelize and reduce the execution time up-to 12 threads after which the CPU and thread management overheads are expected to negatively affect the speedup.

Inferences :

1. Since speedup is calculated by Amdahl's law, the speedup is inversely proportional to parallelized execution time.
2. If we look at the speedup and execution time graphs side by side we would notice that the slope of the descent in **speedup** looks similar to the magnitude of slope of ascent in **execution time**.
3. Both slopes seem to have had a steady but tiny improvements in the performance with increase in number of threads.

Estimated Parallelization Fraction :

```
=== Parallelization Fraction Table ===
   Threads                P. Fraction (Parallel Mult.)
-------------------------------------------------------
         1                                    0.000000
         2                                    0.922108
         4                                    0.919848
         6                                    0.939820
         8                                    0.932055
        10                                    0.926185
        12                                    0.922292
        16                                    0.906638
        20                                    0.895541
        32                                    0.883499
        64                                    0.876134
```

Observations :

1. The parallelization fraction reaches its peak at 8 threads.
2. The parallelization fraction does drop slightly but not at an alarming rate.
3. Up to 16 threads, the program maintains a parallelization fraction (≥ 0.9).

Conclusion:

- The optimal number of threads for this workload appears to be around **6-8**.
- Beyond this point, increasing threads doesn't seem to affect the execution time or the parallelization factor much negatively.
- The magnitude of the reduction in the speedup of the program by increasing threads, is very less, similar to addition of matrices.
- One thing to notice would be that the speedup and execution time graphs are smooth curves unlike addition
- We could imply that multiplication is very positively affected by parallelization and a large number of threads doesn't affect it much negatively as it's a very computationally expensive operation even when the dimensions of the matrix are smaller than that of the matrices used in addition.