**Class:** Final Year (Computer Science and Engineering)

**Year:** 2024-25 **Semester:** 1

**Course:** High Performance Computing Lab

**Practical No. 5**

**Exam Seat No:22510023**

**Name : Ganesh Chavhan**

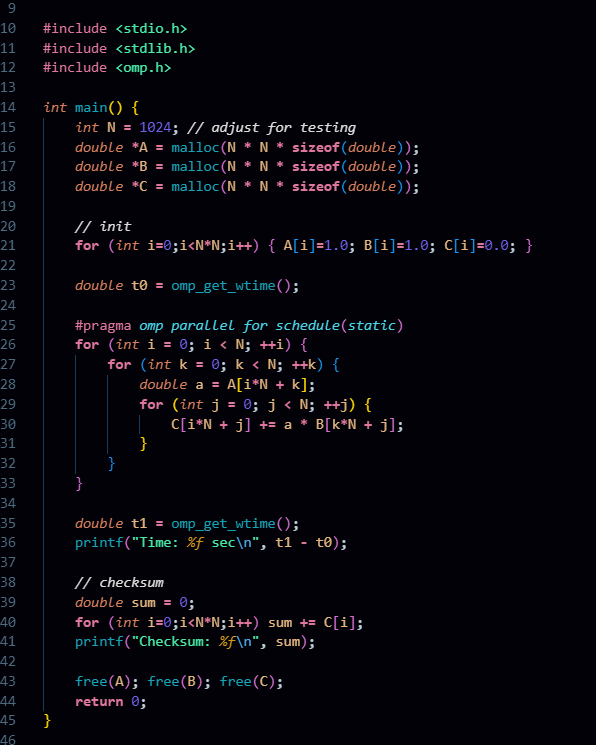
**Title of practical: Implementation of OpenMP programs.**

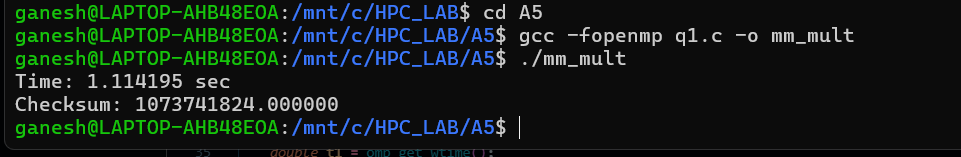
Implement following Programs using OpenMP with C:

1. Implementation of Matrix-Matrix Multiplication.
2. Implementation of Matrix-scalar Multiplication.
3. Implementation of Matrix-Vector Multiplication.
4. Implementation of Prefix sum.

**Problem Statement 1:** Implementation of Matrix-Matrix Multiplication.

**Screenshots:**





**Information:**

Uses #pragma omp parallel for with static scheduling.

Matrices initialized with 1.0 values.

Output validated using checksum.

Measures execution time using omp\_get\_wtime()

**Analysis:**

Computation complexity: **O(N³)**.

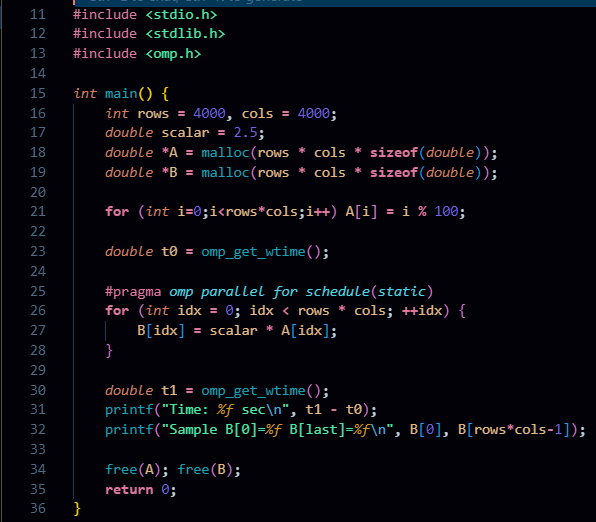
Each row’s computation is independent, hence parallelizable.

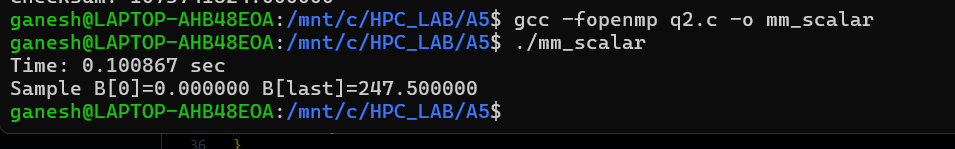
Parallel execution reduces runtime significantly on large input sizes.

Speedup is limited by memory bandwidth when N is very large.

**Problem Statement 2:** Implementation of Matrix-scalar Multiplication.

**Screenshots:**





**Information:**

Scalar chosen = 2.5.

Matrix initialized with values from 0 to 99.

Uses OpenMP parallel for loop.

Output validated by printing first and last elements.

**Analysis:**

Complexity: **O(N²)**.

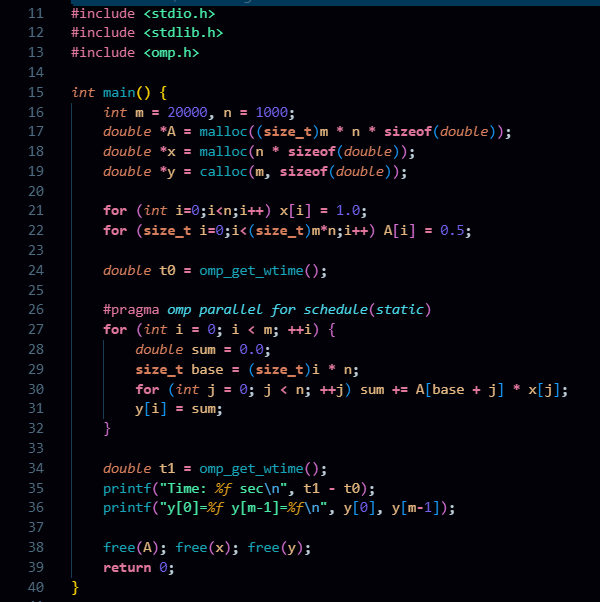
Each element multiplication is independent (embarrassingly parallel).

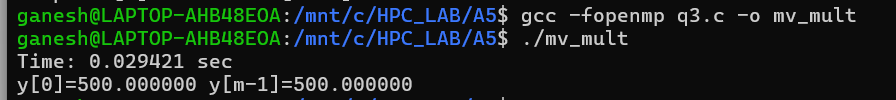
Good scalability and near-linear speedup.

Minimal synchronization required.

**Problem Statement 3:** Implementation of Matrix-Vector Multiplication

**Screenshots:**





**Information:**

Matrix initialized with 0.5.

Vector initialized with 1.0.

Each row’s dot product computed in parallel.

Results stored in output vector y.

**Analysis:**

Time complexity: **O(m·n)**.

Parallelism applied at row level; good load balance.

Memory locality improves performance.

For large matrices, parallelization reduces time significantly compared to sequential.

**Problem Statement 4:** Implementation of Prefix sum

**Screenshots:**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

*int* main() {

*int* n **=** 20000000**;**

*double* **\***a **=** malloc(n **\*** **sizeof**(*double*))**;**

*double* **\***s **=** malloc(n **\*** **sizeof**(*double*))**;**

**for** (*int* i**=**0**;**i**<**n**;**i**++**) a[i] **=** 1.0**;** *// or random*

*int* T**;**

    #pragma *omp* *parallel*

    { **if** (omp\_get\_thread\_num() **==** 0) T **=** omp\_get\_num\_threads()**;** }

*double* **\***block\_sum **=** malloc(T **\*** **sizeof**(*double*))**;**

*double* t0 **=** omp\_get\_wtime()**;**

*// Phase 1: local scan and block sums*

    #pragma *omp* *parallel*

    {

*int* tid **=** omp\_get\_thread\_num()**;**

*int* start **=** (*long*)tid **\*** n **/** T**;**

*int* end **=** (*long*)(tid **+** 1) **\*** n **/** T**;**

*double* local **=** 0.0**;**

**for** (*int* i **=** start**;** i **<** end**;** **++**i) {

            local **+=** a[i]**;**

            s[i] **=** local**;** *// local inclusive scan*

        }

        block\_sum[tid] **=** local**;**

    }

*// Phase 2: prefix of block sums (sequential, T small)*

*double* acc **=** 0.0**;**

**for** (*int* t **=** 0**;** t **<** T**;** **++**t) {

*double* tmp **=** block\_sum[t]**;**

        block\_sum[t] **=** acc**;** *// block\_sum[t] will hold offset for block t*

        acc **+=** tmp**;**

    }

*// Phase 3: add offsets to each block (except first)*

    #pragma *omp* *parallel*

    {

*int* tid **=** omp\_get\_thread\_num()**;**

*int* start **=** (*long*)tid **\*** n **/** T**;**

*int* end **=** (*long*)(tid **+** 1) **\*** n **/** T**;**

*double* offset **=** block\_sum[tid]**;**

**if** (offset **!=** 0.0) {

**for** (*int* i **=** start**;** i **<** end**;** **++**i) s[i] **+=** offset**;**

        }

    }

*double* t1 **=** omp\_get\_wtime()**;**

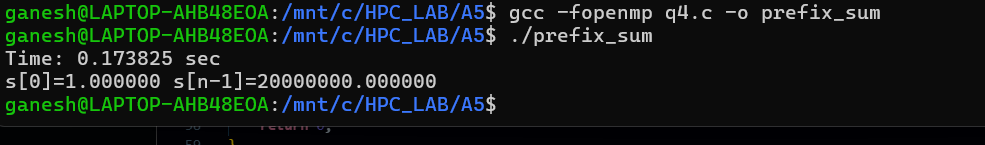
    printf("Time: *%f* sec\n"**,** t1 **-** t0)**;**

    printf("s[0]=*%f* s[n-1]=*%f*\n"**,** s[0]**,** s[n**-**1])**;**

    free(a)**;** free(s)**;** free(block\_sum)**;**

**return** 0**;**

}



**Information:**

· Input array initialized with 1.0.

· Performed in **3 phases**:

Local prefix sum by each thread.

Sequential prefix of block sums.

Offset addition for each block.

· Uses thread IDs and block partitions.

**Analysis:**

Complexity: **O(n)** with improved parallel throughput.

Achieves concurrency by dividing array into blocks.

Needs synchronization barriers between phases.

Demonstrates parallel algorithm design for dependency-heavy problems.

**Github Link: <https://github.com/Ganesh-Chavhan/HPC_LAB/tree/main/A5>**