**Course: High Performance Computing Lab**

**Practical No 5**

**PRN : 23520006**

**Name: Vivek katkar**

**Batch : B8**

**Title -** Implementation of OpenMP programs

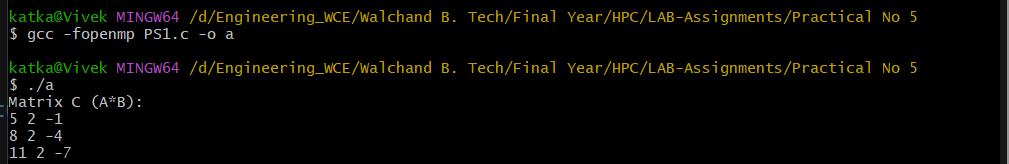
**Problem Statement 1:**

Implementation of Matrix-Matrix Multiplication

**Information:**  
Matrix–Matrix multiplication is a compute-intensive operation where each element of the result matrix is the dot product of a row from the first matrix and a column from the second.  
In OpenMP, independent element computations can be parallelized using #pragma omp parallel for, and collapse(2) can combine loop iterations for better load balancing.

**Analysis:**

* Time complexity: O(n^3)
* Highly parallelizable because each C[i][j] independently.
* Synchronization is not required since each thread writes to a unique element of the result matrix.
* **#include <stdio.h>**
* **#include <omp.h>**
* **#define N 3**
* **int main() {**
* **int A[N][N], B[N][N], C[N][N] = {0};**
* **// Initialize matrices**
* **for (int i = 0; i < N; i++)**
* **for (int j = 0; j < N; j++) {**
* **A[i][j] = i + j;**
* **B[i][j] = i - j;**
* **}**
* **#pragma omp parallel for collapse(2)**
* **for (int i = 0; i < N; i++) {**
* **for (int j = 0; j < N; j++) {**
* **for (int k = 0; k < N; k++) {**
* **C[i][j] += A[i][k] \* B[k][j];**
* **}**
* **}**
* **}**
* **printf("Matrix C (A\*B):\n");**
* **for (int i = 0; i < N; i++) {**
* **for (int j = 0; j < N; j++)**
* **printf("%d ", C[i][j]);**
* **printf("\n");**
* **}**
* **return 0;**
* **}**

****

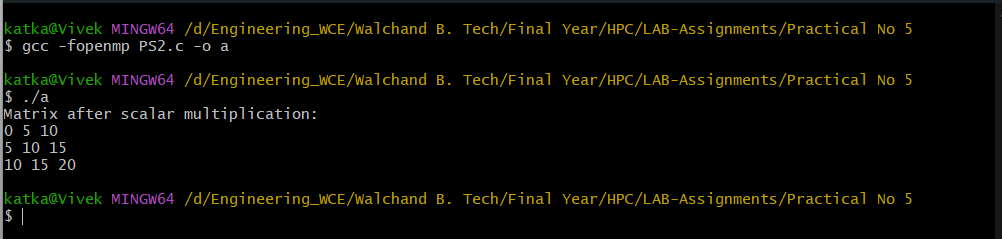
**Problem Statement 2** :

Matrix–Scalar Multiplication using OpenMP

**Information:**  
Matrix–Scalar multiplication involves multiplying every element of a matrix by a constant scalar. In OpenMP, this is highly parallelizable since all elements are updated independently.

**Analysis:**

* Time complexity: O(n^2)
* No synchronization needed because threads work on different elements.
* collapse(2) improves parallel loop efficiency.
* **#include <stdio.h>**
* **#include <omp.h>**
* **#define N 3**
* **int main() {**
* **int A[N][N], scalar = 5;**
* **for (int i = 0; i < N; i++)**
* **for (int j = 0; j < N; j++)**
* **A[i][j] = i + j;**
* **#pragma omp parallel for collapse(2)**
* **for (int i = 0; i < N; i++)**
* **for (int j = 0; j < N; j++)**
* **A[i][j] \*= scalar;**
* **printf("Matrix after scalar multiplication:\n");**
* **for (int i = 0; i < N; i++) {**
* **for (int j = 0; j < N; j++)**
* **printf("%d ", A[i][j]);**
* **printf("\n");**
* **}**
* **return 0;**
* **}**



**Problem Statement 3** :

Matrix–Vector Multiplication using OpenMP

**Information:**  
Matrix–Vector multiplication computes the dot product of each row of a matrix with a given vector. Parallelization is straightforward since each result element is computed independently.

**Analysis:**

* Time complexity: O(n^2)
* Easy to parallelize with #pragma omp parallel for.
* No synchronization needed as each thread writes to separate memory locations.

**#include <stdio.h>**

**#include <omp.h>**

**#define N 3**

**int main() {**

**int A[N][N], V[N], R[N] = {0};**

**for (int i = 0; i < N; i++) {**

**V[i] = i + 1;**

**for (int j = 0; j < N; j++)**

**A[i][j] = i + j;**

**}**

**#pragma omp parallel for**

**for (int i = 0; i < N; i++)**

**for (int j = 0; j < N; j++)**

**R[i] += A[i][j] \* V[j];**

**printf("Result Vector:\n");**

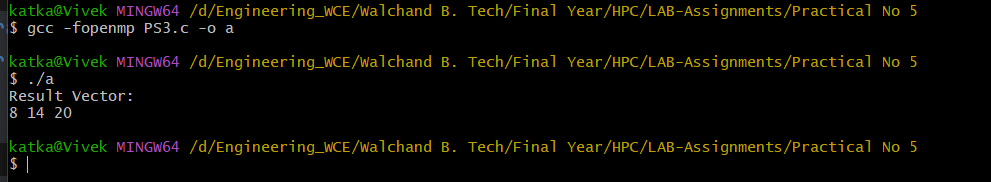
**for (int i = 0; i < N; i++)**

**printf("%d ", R[i]);**

**printf("\n");**

**return 0;**

**}**



**Problem Statement 4** :

Prefix Sum using OpenMP

**Information:**  
The prefix sum problem computes the cumulative sum of an array. Unlike multiplication problems, prefix sum has a dependency chain, making it harder to parallelize efficiently.

**Analysis:**

* Sequential prefix sum is O(n), parallel version can use tree-based scan.
* In naive OpenMP, a true parallel prefix sum needs synchronization or a two-phase algorithm.
* For demonstration, we show a sequential dependency in parallel regions.

**#include <stdio.h>**

**#include <omp.h>**

**#define N 8**

**int main() {**

**int arr[N], prefix[N];**

**for (int i = 0; i < N; i++) arr[i] = i + 1;**

**prefix[0] = arr[0];**

**for (int i = 1; i < N; i++)  // Sequential due to dependency**

**prefix[i] = prefix[i - 1] + arr[i];**

**printf("Prefix Sum Array:\n");**

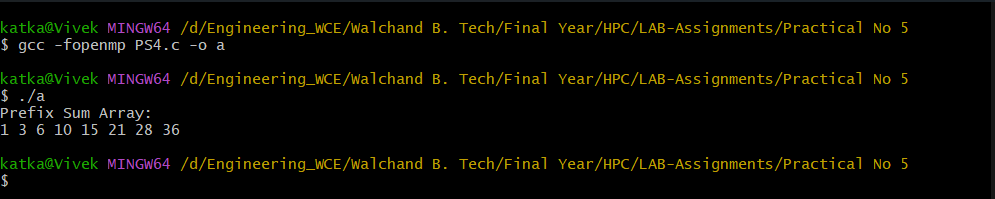
**for (int i = 0; i < N; i++)**

**printf("%d ", prefix[i]);**

**printf("\n");**

**return 0;**

**}**



**GitHub Link:** <https://github.com/vivekkatkar/hpcl>

**Conclusion**

This assignment implemented matrix and array operations using OpenMP, showing that multiplication tasks parallelize efficiently due to independent computations. The collapse and parallel for clauses improved performance by balancing workloads. Prefix sum revealed challenges from data dependencies, requiring careful synchronization or specialized algorithms. Overall, the work highlighted the trade-off between correctness and performance in parallel programming.