Nagar Yuwak Shikshan Sanstha's

Yeshwantrao Chavan College of Engineering

(An Autonomous Institution affiliated to Rashtrasant Tukadoji Maharaj Nagpur University)
Hingna Road, Wanadongri, Nagpur - 441 110
NAAC A++
Ph.: 07104-237919, 234623, 329249, 329250 Fax: 07104-232376, Website: www.ycce.edu

Department of Artificial Intelligence & Data Science

Vision of the Department

To be a well-known centre for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.

Mission of the Department

To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary area by developing Problem-solving skills through emerging technologies.

Session 2025-2026

Vision: Dream of where you want.	**Mission:** Means to achieve Vision

Program Educational Objectives of the program (PEO): (broad statements that describe the professional and career accomplishments)

PEO1	**Preparation**	**P: Preparation**	**Pep-CL abbreviation pronounce as Pep-si-lL easy to recall**
PEO2	**Core Competence**	**E: Environment (Learning Environment)**	
PEO3	**Breadth**	**P: Professionalism**	
PEO4	**Professionalism**	**C: Core Competence**	
PEO5	**Learning Environment**	**L: Breadth (Learning in diverse areas)**	

Program Outcomes (PO):

1. Understand and Apply Parallel Programming Concepts
2. Analyse and Improve Program Performance.
3. Demonstrate Practical Skills in HPC Tools and Environments.

Keywords of POs:
Engineering knowledge, Problem analysis, Design/development of solutions, Conduct Investigations of Complex Problems, Engineering Tool Usage, The Engineer and The World, Ethics, Individual and Collaborative Team work, Communication, Project Management and Finance, Life-Long Learning

PSO Keywords: Cutting edge technologies, Research

"I am an engineer, and I know how to apply engineering knowledge to investigate, analyse and design solutions to complex problems using tools for entire world following all ethics in a collaborative way with proper management skills throughout my life." *to contribute to the development of cutting-edge technologies and Research*.

Integrity: I will adhere to the Laboratory Code of Conduct and ethics in its entirety.

Name and Signature of Student and Date
Sakshi Gokhale
12/08/25

Session	2025-26 (ODD)	**Course Name**	HPC Lab
Semester	7 AIDS	**Course Code**	22ADS706
Roll No	16	**Name of Student**	Sakshi Gokhale

Practical Number	2
Course Outcome	1. Understand and Apply Parallel Programming Concepts 2. Analyse and Improve Program Performance
Aim	Measuring Program Performance
Problem Definition	Measuring Program Performance
Theory (100 words)	In High Performance Computing (HPC) and general programming, measuring program performance is essential to evaluate efficiency and resource usage. Performance is usually assessed in terms of execution time, memory consumption, and scalability. By measuring performance, developers can identify slow sections of code, optimize algorithms, and compare different implementations of the same problem. This process also plays a major role in benchmarking HPC applications on various hardware platforms. One of the simplest methods in Linux is the time command, which reports three values: real time (wall clock time), user time (time spent in program execution), and system time (time spent in kernel operations). It is effective for measuring overall runtime. For finer analysis, built-in timing functions are used within programs. In OpenMP, omp_get_wtime() gives wall clock time for parallel code regions, while MPI programs use MPI_Wtime() to measure computation and communication time separately. These approaches help identify bottlenecks in specific program segments. For advanced analysis, profiling tools such as gprof or perf are applied. Profiling provides detailed statistics about function calls, memory usage, and system behavior, offering deeper insight into program execution. Overall,

	performance measurement ensures efficient, optimized, and scalable applications.

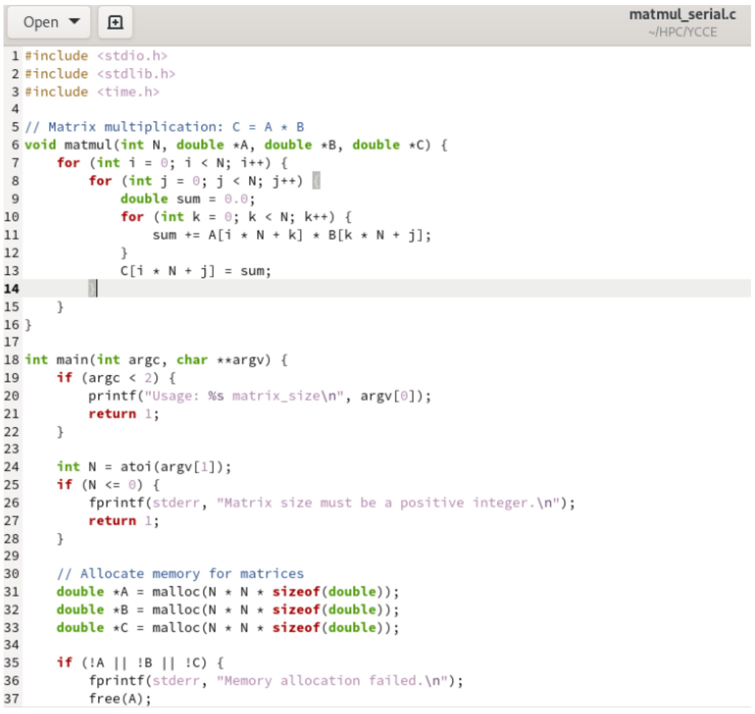
Procedure and Execution
 	
(100 Words)
	Code:
	
	Serial Matrix Multiplication
	
	`#include <stdio.h>`
	`#include <stdlib.h>`
	`#include <time.h>`
	
	`// Serial matrix multiplication`
	`void matmul(int N, double *A, double *B, double *C) {`
	`    for (int i = 0; i < N; i++) {`
	`        for (int j = 0; j < N; j++) {`
	`            double sum = 0.0;`
	`            for (int k = 0; k < N; k++) {`
	`                sum += A[i * N + k] * B[k * N + j];`
	`            }`
	`            C[i * N + j] = sum;`
	`        }`
	`    }`
	`}`
	
	`int main(int argc, char **argv) {`
	`    if (argc < 2) {`
	`        printf("Usage: %s matrix_size\n", argv[0]);`
	`        return 1;`
	`    }`
	
	`    int N = atoi(argv[1]);`
	`    double *A = malloc(N * N * sizeof(double));`
	`    double *B = malloc(N * N * sizeof(double));`
	`    double *C = malloc(N * N * sizeof(double));`
	
	`    // Initialize matrices`
	`    for (int i = 0; i < N * N; i++) {`
	`        A[i] = 1.0;`
	`        B[i] = 2.0;`
	`    }`
	
	`    clock_t start = clock();`
	`    matmul(N, A, B, C);`

```
    clock_t end = clock();

    double time_spent = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Serial MatMul elapsed time: %f seconds\n", time_spent);

    free(A);
    free(B);
    free(C);

    return 0;
}
```

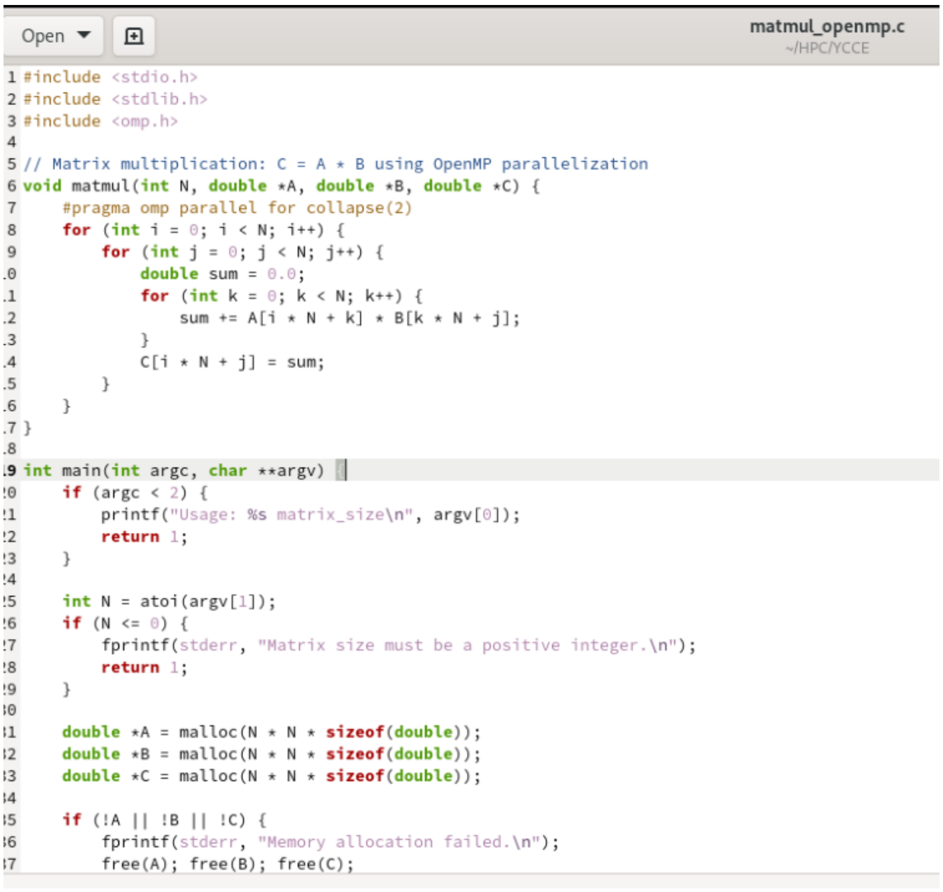
Parallel Matrix Multiplication with OpenMP

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

// Parallel matrix multiplication using OpenMP
void matmul(int N, double *A, double *B, double *C) {
    #pragma omp parallel for collapse(2)
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            double sum = 0.0;
            for (int k = 0; k < N; k++) {
                sum += A[i * N + k] * B[k * N + j];
            }
            C[i * N + j] = sum;
        }
    }
}

int main(int argc, char **argv) {
    if (argc < 2) {
        printf("Usage: %s matrix_size\n", argv[0]);
        return 1;
    }

    int N = atoi(argv[1]);
    double *A = malloc(N * N * sizeof(double));
    double *B = malloc(N * N * sizeof(double));
    double *C = malloc(N * N * sizeof(double));
    // Initialize matrices
    for (int i = 0; i < N * N; i++) {
        A[i] = 1.0;
        B[i] = 2.0;
    }

    double start = omp_get_wtime();
    matmul(N, A, B, C);
    double end = omp_get_wtime();

    printf("OpenMP MatMul elapsed time: %f seconds\n", end - start);

    free(A);
    free(B);
    free(C);

    return 0;
}
```

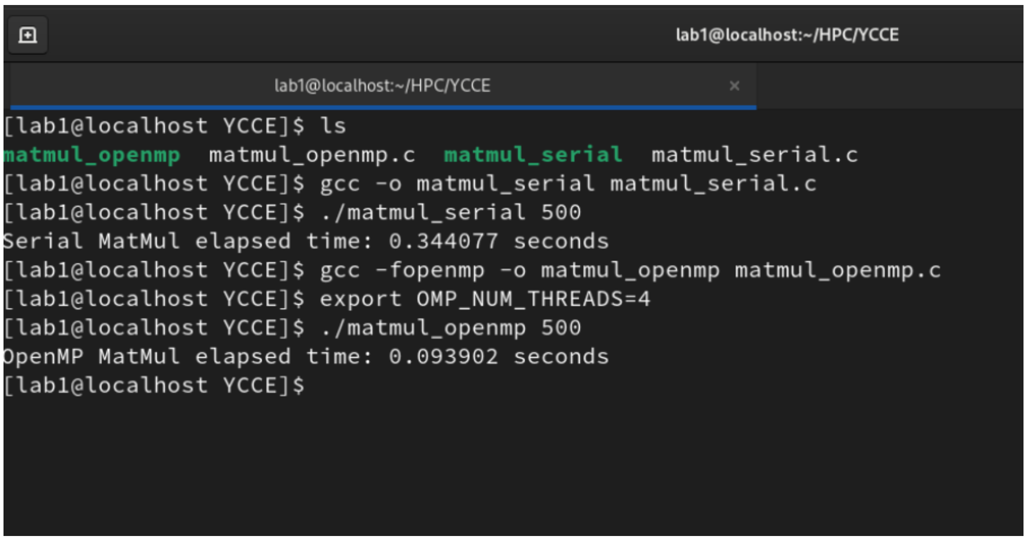
Output:



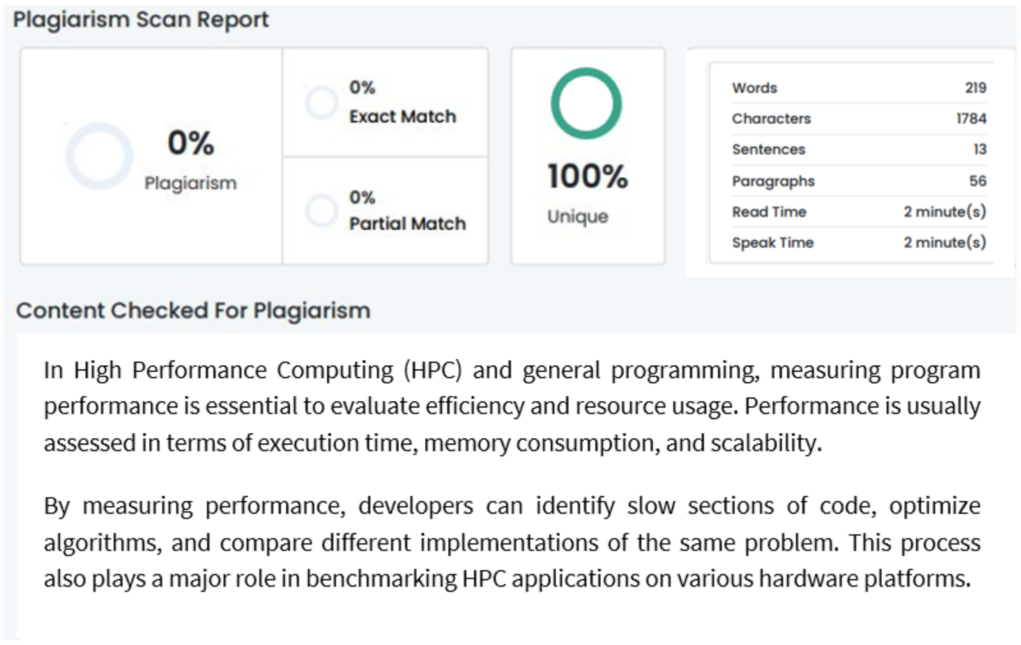
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Matrix multiplication: C = A * B
void matmul(int N, double *A, double *B, double *C) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            double sum = 0.0;
            for (int k = 0; k < N; k++) {
                sum += A[i * N + k] * B[k * N + j];
            }
            C[i * N + j] = sum;

    }
}

int main(int argc, char **argv) {
    if (argc < 2) {
        printf("Usage: %s matrix_size\n", argv[0]);
        return 1;
    }

    int N = atoi(argv[1]);
    if (N <= 0) {
        fprintf(stderr, "Matrix size must be a positive integer.\n");
        return 1;
    }

    // Allocate memory for matrices
    double *A = malloc(N * N * sizeof(double));
    double *B = malloc(N * N * sizeof(double));
    double *C = malloc(N * N * sizeof(double));

    if (!A || !B || !C) {
        fprintf(stderr, "Memory allocation failed.\n");
        free(A);
```



```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

// Matrix multiplication: C = A * B using OpenMP parallelization
void matmul(int N, double *A, double *B, double *C) {
    #pragma omp parallel for collapse(2)
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            double sum = 0.0;
            for (int k = 0; k < N; k++) {
                sum += A[i * N + k] * B[k * N + j];
            }
            C[i * N + j] = sum;
        }
    }
}

int main(int argc, char **argv) {
    if (argc < 2) {
        printf("Usage: %s matrix_size\n", argv[0]);
        return 1;
    }

    int N = atoi(argv[1]);
    if (N <= 0) {
        fprintf(stderr, "Matrix size must be a positive integer.\n");
        return 1;
    }

    double *A = malloc(N * N * sizeof(double));
    double *B = malloc(N * N * sizeof(double));
    double *C = malloc(N * N * sizeof(double));

    if (!A || !B || !C) {
        fprintf(stderr, "Memory allocation failed.\n");
        free(A); free(B); free(C);
```



```
[lab1@localhost YCCE]$ ls
matmul_openmp  matmul_openmp.c  matmul_serial  matmul_serial.c
[lab1@localhost YCCE]$ gcc -o matmul_serial matmul_serial.c
[lab1@localhost YCCE]$ ./matmul_serial 500
Serial MatMul elapsed time: 0.344077 seconds
[lab1@localhost YCCE]$ gcc -fopenmp -o matmul_openmp matmul_openmp.c
[lab1@localhost YCCE]$ export OMP_NUM_THREADS=4
[lab1@localhost YCCE]$ ./matmul_openmp 500
OpenMP MatMul elapsed time: 0.093902 seconds
[lab1@localhost YCCE]$
```

Output Analysis	The experiment clearly demonstrates the performance improvement achieved by parallelizing matrix multiplication using OpenMP. While the serial version required about 0.34 seconds, the OpenMP version reduced the execution time to about 0.09 seconds, achieving nearly a 3.7× speedup with 4 threads. This shows that parallel programming with OpenMP can significantly enhance computation efficiency for large-scale matrix operations, making it well-suited for High-Performance Computing (HPC) applications.
Link of student Github profile where lab assignment has been uploaded	https://github.com/sakshi-gokhale/Lab-HPC
Conclusion	The results confirm that OpenMP parallelization significantly improves the performance of matrix multiplication. The parallel implementation using 4 threads reduced the computation time by almost 3.7 times compared to the serial version. This proves the effectiveness of parallel programming in High Performance Computing (HPC), especially for computationally intensive tasks like matrix operations.
Plag Report (Similarity index < 12%)	 Plagiarism Scan Report 0% Plagiarism 0% Exact Match 0% Partial Match 100% Unique Words 219 Characters 1784 Sentences 13 Paragraphs 56 Read Time 2 minute(s) Speak Time 2 minute(s) Content Checked For Plagiarism In High Performance Computing (HPC) and general programming, measuring program performance is essential to evaluate efficiency and resource usage. Performance is usually assessed in terms of execution time, memory consumption, and scalability. By measuring performance, developers can identify slow sections of code, optimize algorithms, and compare different implementations of the same problem. This process also plays a major role in benchmarking HPC applications on various hardware platforms.

Vision of the Department

To be a well-known centre for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.

Mission of the Department

To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary area by developing Problem-solving skills through emerging technologies.

Date	12/08/25