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

**Year:** 2023-24 **Semester:** 1

**Course:** High Performance Computing Lab

**Practical No. 5**

**Exam Seat No: 2020BTECS00037**

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

Implement following Programs using OpenMP with C:

1. Implementation of sum of two lower triangular matrices.
2. Implementation of Matrix-Matrix Multiplication.

**Problem Statement 1:**

**matrix\_sum\_serial**

#include <stdio.h>

#include <time.h>

// Size of the matrices

#define N 4

int main() {

    // Lower triangular matrices

    int A[N][N];

    int B[N][N];

    // Resultant matrix

    int C[N][N];

    clock\_t start\_time, end\_time;

    // Initialize matrices A and B

    for (int i = 0; i < N; i++) {

        for (int j = 0; j <= i; j++) {

            A[i][j] = i + j + 1;

            B[i][j] = i - j + 1;

        }

    }

    start\_time = clock();

    for (int i = 0; i < N; i++) {

        for (int j = 0; j <= i; j++) {

            C[i][j] = A[i][j] + B[i][j];

        }

    }

    end\_time = clock();

    printf("Resultant Matrix C:\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j <= i; j++) {

            printf("%d ", C[i][j]);

        }

        printf("\n");

    }

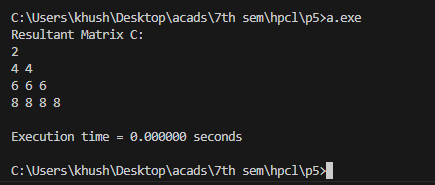
    double exec\_time = (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC;

    printf("\nExecution time = %.6f seconds\n", exec\_time);

    return 0;

}

**Screenshots:**

****

1. **Implementation of Matrix-Matrix Multiplication.**

**Problem Statement 2:**

**matrix\_mul\_serial**

#include <stdio.h>

#include <stdlib.h>

#include <sys/time.h>

#define ROWS\_A 8

#define COLS\_A 8

#define ROWS\_B 8

#define COLS\_B 8

double get\_seconds(struct timeval start, struct timeval end) {

    return (end.tv\_sec - start.tv\_sec) + (end.tv\_usec - start.tv\_usec) / 1e6;

}

int main() {

    srand(time(NULL));

    struct timeval start\_time, end\_time;

    int A[ROWS\_A][COLS\_A];

    int B[ROWS\_B][COLS\_B];

    int C[ROWS\_A][COLS\_B];

    // Initialize matrices A and B with random values

    for (int i = 0; i < ROWS\_A; i++) {

        for (int j = 0; j < COLS\_A; j++) {

            // Random values between 0 and 99

            A[i][j] = rand() % 100;

        }

    }

    for (int i = 0; i < ROWS\_B; i++) {

        for (int j = 0; j < COLS\_B; j++) {

            // Random values between 0 and 99

            B[i][j] = rand() % 100;

        }

    }

    gettimeofday(&start\_time, NULL);

    // Perform matrix multiplication

    for (int i = 0; i < ROWS\_A; i++) {

        for (int j = 0; j < COLS\_B; j++) {

            C[i][j] = 0;

            for (int k = 0; k < COLS\_A; k++) {

                C[i][j] += A[i][k] \* B[k][j];

            }

        }

    }

    gettimeofday(&end\_time, NULL);

    // Print the result matrix C

    printf("Matrix A:\n");

    for (int i = 0; i < ROWS\_A; i++) {

        for (int j = 0; j < COLS\_A; j++) {

            printf("%d ", A[i][j]);

        }

        printf("\n");

    }

    printf("\nMatrix B:\n");

    for (int i = 0; i < ROWS\_B; i++) {

        for (int j = 0; j < COLS\_B; j++) {

            printf("%d ", B[i][j]);

        }

        printf("\n");

    }

    printf("\nResultant Matrix C:\n");

    for (int i = 0; i < ROWS\_A; i++) {

        for (int j = 0; j < COLS\_B; j++) {

            printf("%d ", C[i][j]);

        }

        printf("\n");

    }

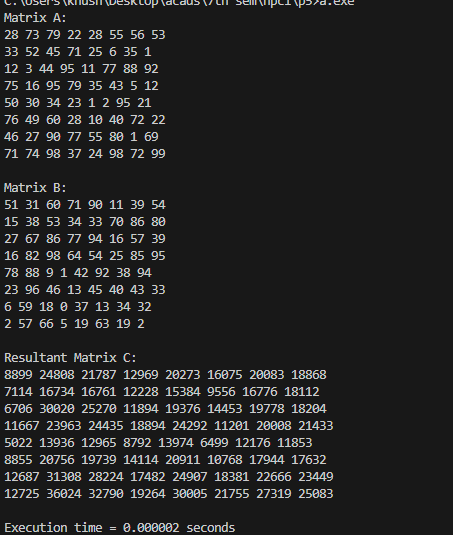
    double exec\_time = get\_seconds(start\_time, end\_time);

    printf("\nExecution time = %.6f seconds\n", exec\_time);

    return 0;

}

**Screenshots:**

****

**Parallel**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <omp.h>

#define ROWS\_A 8

#define COLS\_A 8

#define ROWS\_B 8

#define COLS\_B 8

int main() {

    srand(time(NULL));

    double start\_time, end\_time;

    int A[ROWS\_A][COLS\_A];

    int B[ROWS\_B][COLS\_B];

    int C[ROWS\_A][COLS\_B];

    // Initialize matrices A and B with random values

    for (int i = 0; i < ROWS\_A; i++) {

        for (int j = 0; j < COLS\_A; j++) {

            // Random values between 0 and 99

            A[i][j] = rand() % 100;

        }

    }

    for (int i = 0; i < ROWS\_B; i++) {

        for (int j = 0; j < COLS\_B; j++) {

            // Random values between 0 and 99

            B[i][j] = rand() % 100;

        }

    }

    start\_time = omp\_get\_wtime();

    omp\_set\_num\_threads(8);

    // Perform matrix multiplication

    for (int i = 0; i < ROWS\_A; i++) {

        for (int j = 0; j < COLS\_B; j++) {

            C[i][j] = 0;

            for (int k = 0; k < COLS\_A; k++) {

                C[i][j] += A[i][k] \* B[k][j];

            }

        }

    }

    end\_time = omp\_get\_wtime();

    // Print the result matrix C

    printf("Matrix A:\n");

    for (int i = 0; i < ROWS\_A; i++) {

        for (int j = 0; j < COLS\_A; j++) {

            printf("%d ", A[i][j]);

        }

        printf("\n");

    }

    printf("\nMatrix B:\n");

    for (int i = 0; i < ROWS\_B; i++) {

        for (int j = 0; j < COLS\_B; j++) {

            printf("%d ", B[i][j]);

        }

        printf("\n");

    }

    printf("\nResultant Matrix C:\n");

    for (int i = 0; i < ROWS\_A; i++) {

        for (int j = 0; j < COLS\_B; j++) {

            printf("%d ", C[i][j]);

        }

        printf("\n");

    }

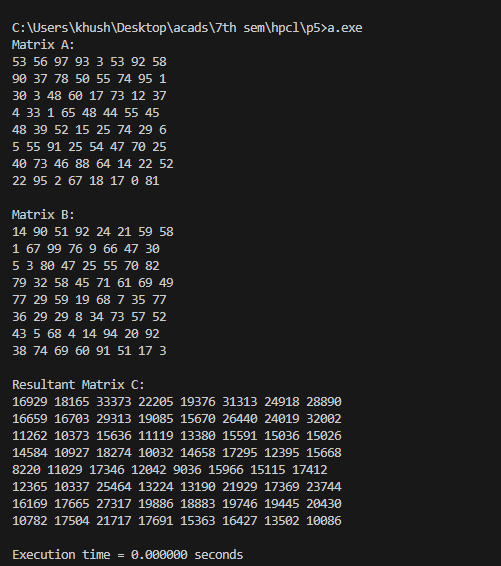
    double exec\_time = end\_time - start\_time;

    printf("\nExecution time = %lf seconds\n", exec\_time);

    return 0;

}

**Screenshots:**

****