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

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

**Course:** High Performance Computing Lab

**Practical No. 5**

**Exam Seat No: 2020BTECS00055**

**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:**

**Sequential code:**

#include <stdio.h>

#include <omp.h>

#define N 5  // Size of matrices

void sumLowerTriangularMatrices(int A[][N], int B[][N], int result[][N]) {

    for (int i = 0; i < N; i++) {

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

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

        }

    }

}

int main() {

    // int matrixA[N][N] = {

    //     {1, 0, 0, 0},

    //     {2, 3, 0, 0},

    //     {4, 5, 6, 0},

    //     {7, 8, 9, 10}

    // };

    // int matrixB[N][N] = {

    //     {5, 0, 0, 0},

    //     {4, 6, 0, 0},

    //     {3, 2, 1, 0},

    //     {8, 7, 6, 5}

    // };

    int matrixA[N][N] = {

        {1, 0, 0, 0, 0},

        {2, 3, 0, 0, 0},

        {4, 5, 6, 0, 0},

        {7, 8, 9, 10, 0},

        {11, 12, 13, 14, 15}

    };

    int matrixB[N][N] = {

        {5, 0, 0, 0, 0},

        {4, 6, 0, 0, 0},

        {3, 2, 1, 0, 0},

        {8, 7, 6, 5, 0},

        {9, 10, 11, 12, 13}

    };

    int result[N][N];

    double start\_time = omp\_get\_wtime();

    sumLowerTriangularMatrices(matrixA, matrixB, result);

    double end\_time = omp\_get\_wtime();

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

    printf("Matrix A:\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            printf("%d ", matrixA[i][j]);

        }

        printf("\n");

    }

    printf("\nMatrix B:\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            printf("%d ", matrixB[i][j]);

        }

        printf("\n");

    }

    printf("\nResult Matrix:\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            if (j <= i) {

                printf("%d ", result[i][j]);

            } else {

                printf("0 ");

            }

        }

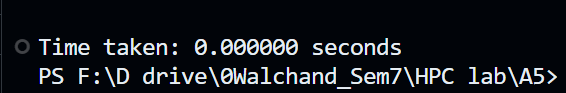
        printf("\n");

    }

    printf("\nTime taken: %f seconds\n", elapsed\_time);

    return 0;

}

****

**Parallel Code:**

#include <stdio.h>

#include <omp.h>

#define N 5  // Size of matrices

void sumLowerTriangularMatrices(int A[][N], int B[][N], int result[][N]) {

    #pragma omp parallel for

    for (int i = 0; i < N; i++) {

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

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

        }

    }

}

int main() {

    // int matrixA[N][N] = {

    //     {1, 0, 0, 0},

    //     {2, 3, 0, 0},

    //     {4, 5, 6, 0},

    //     {7, 8, 9, 10}

    // };

    // int matrixB[N][N] = {

    //     {5, 0, 0, 0},

    //     {4, 6, 0, 0},

    //     {3, 2, 1, 0},

    //     {8, 7, 6, 5}

    // };

    int matrixA[N][N] = {

        {1, 0, 0, 0, 0},

        {2, 3, 0, 0, 0},

        {4, 5, 6, 0, 0},

        {7, 8, 9, 10, 0},

        {11, 12, 13, 14, 15}

    };

    int matrixB[N][N] = {

        {5, 0, 0, 0, 0},

        {4, 6, 0, 0, 0},

        {3, 2, 1, 0, 0},

        {8, 7, 6, 5, 0},

        {9, 10, 11, 12, 13}

    };

    int result[N][N];

    double start\_time = omp\_get\_wtime();

    sumLowerTriangularMatrices(matrixA, matrixB, result);

    double end\_time = omp\_get\_wtime();

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

    printf("Matrix A:\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            printf("%d ", matrixA[i][j]);

        }

        printf("\n");

    }

    printf("\nMatrix B:\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            printf("%d ", matrixB[i][j]);

        }

        printf("\n");

    }

    printf("\nResult Matrix:\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            if (j <= i) {

                printf("%d ", result[i][j]);

            } else {

                printf("0 ");

            }

        }

        printf("\n");

    }

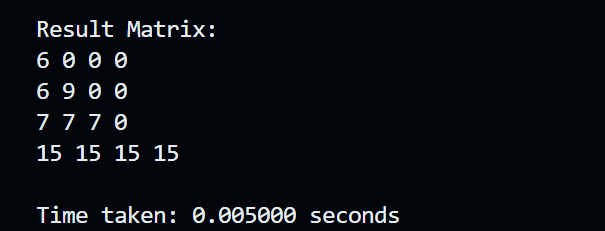
    printf("\nTime taken: %f seconds\n", elapsed\_time);

    return 0;

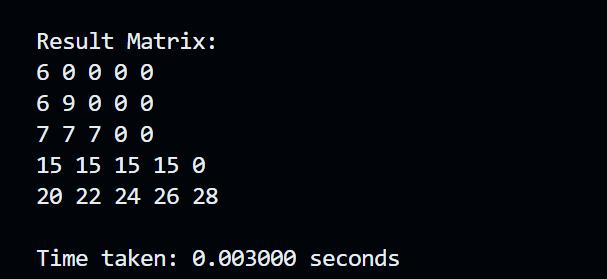
}

**Screenshots:**

**4 x 4 Matrix addition**

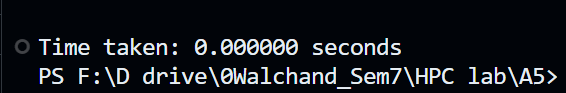
****

**5x5 Matrix addition**

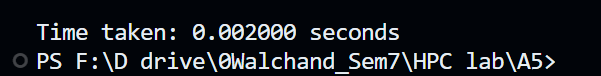
****

**Information:**

**Time taken by Sequential code is**

****

**Time Taken by Parallel code is**

****

**Analysis:**

With 1 Thread:

- The code will run both the sequential and parallel parts sequentially.

- The parallel part introduces overhead due to thread creation and synchronization.

- The parallel execution time is likely to be higher than the sequential execution time.

- For simple computations, using 1 thread is sufficient, and adding more threads may result in overhead surpassing potential gains.

**Problem Statement 2:**

**Sequential Code:**

#include <stdio.h>

#include <omp.h>

#define N 4  // Size

void matrixMultiplication(int A[][N], int B[][N], int result[][N]) {

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            result[i][j] = 0;

            for (int k = 0; k < N; k++) {

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

            }

        }

    }

}

int main() {

    int matrixA[N][N] = {

        {1, 0, 0, 0},

        {2, 3, 0, 0},

        {4, 5, 6, 0},

        {7, 8, 9, 10}

    };

    int matrixB[N][N] = {

        {5, 0, 0, 0},

        {4, 6, 0, 0},

        {3, 2, 1, 0},

        {8, 7, 6, 5}

    };

    // int matrixA[N][N] = {

    //     {1, 2, 3},

    //     {4, 5, 6},

    //     {7, 8, 9}

    // };

    // int matrixB[N][N] = {

    //     {9, 8, 7},

    //     {6, 5, 4},

    //     {3, 2, 1}

    // };

    int result[N][N];

    double start\_time = omp\_get\_wtime();

    matrixMultiplication(matrixA, matrixB, result);

    double end\_time = omp\_get\_wtime();

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

    printf("Matrix A:\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            printf("%d ", matrixA[i][j]);

        }

        printf("\n");

    }

    printf("\nMatrix B:\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            printf("%d ", matrixB[i][j]);

        }

        printf("\n");

    }

    printf("\nResult Matrix:\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            printf("%d ", result[i][j]);

        }

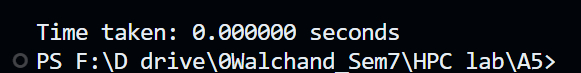
        printf("\n");

    }

    printf("\nTime taken: %f seconds\n", elapsed\_time);

    return 0;

}

****

**Parallel Code:**

#include <stdio.h>

#include <omp.h>

#define N 4  // Size of matrices

void matrixMultiplication(int A[][N], int B[][N], int result[][N]) {

    #pragma omp parallel for

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            result[i][j] = 0;

            for (int k = 0; k < N; k++) {

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

            }

        }

    }

}

int main() {

    int matrixA[N][N] = {

        {1, 0, 0, 0},

        {2, 3, 0, 0},

        {4, 5, 6, 0},

        {7, 8, 9, 10}

    };

    int matrixB[N][N] = {

        {5, 0, 0, 0},

        {4, 6, 0, 0},

        {3, 2, 1, 0},

        {8, 7, 6, 5}

    };

    // int matrixA[N][N] = {

    //     {1, 2, 3},

    //     {4, 5, 6},

    //     {7, 8, 9}

    // };

    // int matrixB[N][N] = {

    //     {9, 8, 7},

    //     {6, 5, 4},

    //     {3, 2, 1}

    // };

    int result[N][N];

    double start\_time = omp\_get\_wtime();

    matrixMultiplication(matrixA, matrixB, result);

    double end\_time = omp\_get\_wtime();

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

    printf("Matrix A:\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            printf("%d ", matrixA[i][j]);

        }

        printf("\n");

    }

    printf("\nMatrix B:\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            printf("%d ", matrixB[i][j]);

        }

        printf("\n");

    }

    printf("\nResult Matrix:\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            printf("%d ", result[i][j]);

        }

        printf("\n");

    }

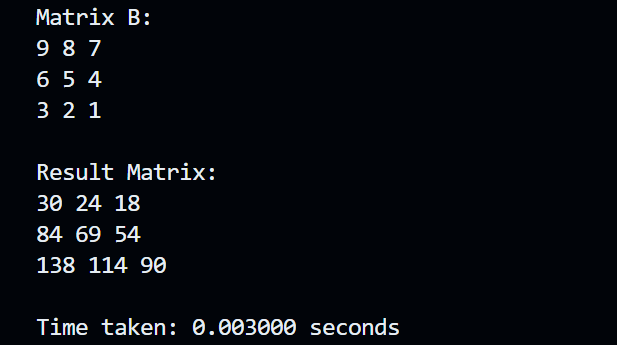
    printf("\nTime taken: %f seconds\n", elapsed\_time);

    return 0;

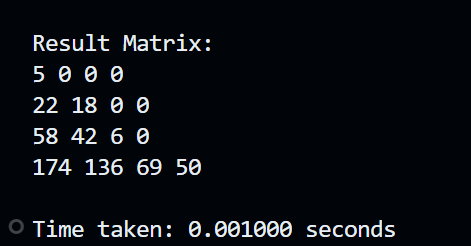
}

**Screenshots:**

**3x3 matrix**

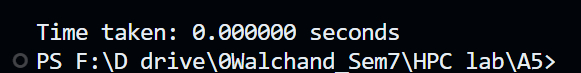
****

**4x4 matrix**

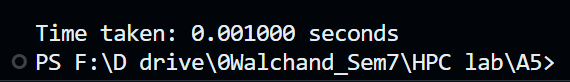
****

**Information:**

Time Taken by sequential Code:

****

Time Taken by Parallel code:



**Analysis:**

- Sequential Execution:

The result matrix is computed sequentially using nested loops for matrix multiplication.

- Parallel Execution:

OpenMP is employed to parallelize the matrix multiplication loop. The `omp\_set\_num\_threads(100)` sets the desired number of threads to 100, although the actual number may vary depending on available resources. The `#pragma omp parallel for shared(A, B, result) collapse(2)` directive starts a parallel loop, distributing the work among the specified number of threads.

Using 100 threads in this scenario may not necessarily yield a significant speedup because matrix multiplication is inherently parallelizable. The efficiency of parallelization depends on factors like the hardware, available resources, and the size of the matrices being multiplied.

**Github Link:**