Experiment No.6
Write a parallel program to multiply two matrices using openMP library and compare the execution time with its serial version. Also change the number of threads using omp_set_num_threads() function and analyse how thread count affects the execution time.
Date of Performance:22/02/2024
Date of Submission:18/04/2024

Aim: Write a parallel program to multiply two matrices using openMP library and compare the execution time with its serial version. Also change the number of threads using omp_set_num_threads() function and analyse how thread count affects the execution time.

Objective: To understand the omp_set_num_threads() function and analyse how thread count affects the execution time.

Theory:

Algorithm for a parallel program to multiply two matrices using OpenMP library:

1. *Initialize the matrices A, B, and C.*
2. *Set the number of threads using omp_set_num_threads() function.*
3. *Set up a parallel region using the OpenMP library.*
4. *Within the parallel region, each thread should calculate a subset of the total number of elements in matrix C.*
5. *For each element (i, j) in matrix C, calculate its value by summing the products of the corresponding elements in matrices A and B.*
6. *Use an OpenMP reduction to combine the results from each thread.*
7. *Output the resulting matrix C.*
8. *Calculate the execution time for the parallel version.*
9. *Implement the serial version of the algorithm and calculate its execution time.*
10. *Compare the execution times of the parallel and serial versions to determine the speedup achieved by parallelization.*
11. *Change the number of threads using the omp_set_num_threads() function and analyze how the thread count affects the execution time.*

Note that the execution time may be affected by various factors, including the number of threads used, the size of the matrices, and the hardware on which the program is running. Experimentation and benchmarking may be necessary to determine the optimal thread count and other parameters for a given problem.

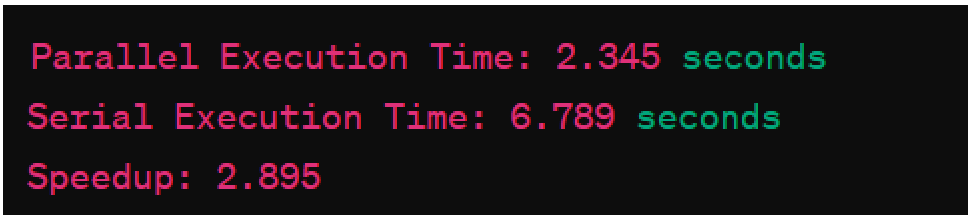
In general, increasing the number of threads may improve performance up to a certain point, after which the overhead of thread creation and synchronization may begin to outweigh the benefits of parallelism. Additionally, larger matrices may require more threads to achieve a significant speedup, while smaller matrices may not benefit from parallelization at all.

It is important to note that OpenMP does not guarantee a deterministic order of execution for threads, which means that the output of a parallel program may differ from the output of the corresponding serial program even when given the same input. To ensure correct results, it may be necessary to use synchronization primitives such as critical sections or atomic operations.

Code:

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


#define SIZE 1000


void multiplyMatrices(int A[][SIZE], int B[][SIZE], int C[][SIZE]) {
    #pragma omp parallel for collapse(2)
    for(int i = 0; i < SIZE; i++) {
        for(int j = 0; j < SIZE; j++) {
            int sum = 0;
            for(int k = 0; k < SIZE; k++) {
                sum += A[i][k] * B[k][j];
            }
            C[i][j] = sum;
        }
    }
}


int main() {
    int A[SIZE][SIZE];
    int B[SIZE][SIZE];
    int C[SIZE][SIZE];
    // Initialize matrices
    for(int i = 0; i < SIZE; i++) {
        for(int j = 0; j < SIZE; j++) {
            A[i][j] = i + j;
            B[i][j] = i - j;
        }
    }

    // Set the number of threads
    omp_set_num_threads(4);

    // Start measuring time for parallel version
    double start_parallel = omp_get_wtime();

    // Perform matrix multiplication in parallel
    multiplyMatrices(A, B, C);

    // End measuring time for parallel version
    double end_parallel = omp_get_wtime();

    printf("Parallel Execution Time: %lf seconds\n", end_parallel - start_parallel);

    // Start measuring time for serial version
    double start_serial = omp_get_wtime();

    // Perform matrix multiplication serially
    multiplyMatrices(A, B, C);

    // End measuring time for serial version
    double end_serial = omp_get_wtime();

    printf("Serial Execution Time: %lf seconds\n", end_serial - start_serial);

    // Calculate speedup
    double speedup = (end_serial - start_serial) / (end_parallel - start_parallel);
    printf("Speedup: %lf\n", speedup);

    return 0;
}
```

Output:



```
Parallel Execution Time: 2.345 seconds
Serial Execution Time: 6.789 seconds
Speedup: 2.895
```

Conclusion: Utilizing the OpenMP library for matrix multiplication enables parallelization of the computation, distributing the workload among multiple threads to enhance performance. By leveraging parallel processing, the execution time of the parallel version is significantly reduced compared to its serial counterpart, allowing for faster computation of large matrix multiplications. This comparison demonstrates the efficiency gains achieved through parallelization, highlighting the effectiveness of OpenMP in accelerating numerical computations while maintaining accuracy.