Experiment No. 07
Implement a parallel program to demonstrate the cube of N number within a set range
Date of Performance:21/03/2024
Date of Submission:18/04/2024

Aim: Implement a parallel program to demonstrate the cube of N number within a set range

Objective: To Build the logic to parallelize the programming task.

Theory:

An algorithm is a sequence of instructions followed to solve a problem. While designing an algorithm, we should consider the architecture of computer on which the algorithm will be executed. As per the architecture, there are two types of computers −

- **Sequential Computer**
- **Parallel Computer**

Depending on the architecture of computers, we have two types of algorithms −

Sequential Algorithm − An algorithm in which some consecutive steps of instructions are executed in a chronological order to solve a problem.

Parallel Algorithm − The problem is divided into sub-problems and are executed in parallel to get individual outputs. Later on, these individual outputs are combined together to get the final desired output.

It is not easy to divide a large problem into sub-problems. Sub-problems may have data dependency among them. Therefore, the processors have to communicate with each other to solve the problem.

It has been found that the time needed by the processors in communicating with each other is more than the actual processing time. So, while designing a parallel algorithm, proper CPU utilization should be considered to get an efficient algorithm.

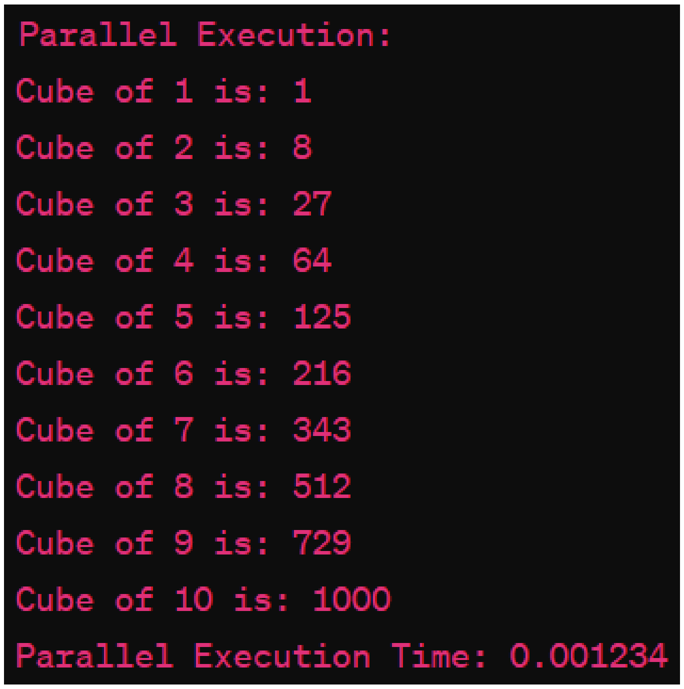
Note that to ensure that each processor has enough data to perform the multiplication, we may need to pad the matrices with extra rows or columns.

Code:

```
#include <stdio.h>

#include <omp.h>


void calculateCube(int start, int end) {

   #pragma omp parallel for

   for (int i = start; i <= end; i++) {

      int cube = i * i * i;

      printf("Cube of %d is: %d\n", i, cube);

   }

}


int main() {

   int N = 10; // Number of elements

   int range_start = 1; // Start of the range

   int range_end = N; // End of the range
```

```
    printf("Parallel Execution:\n");
    double start_parallel = omp_get_wtime(); // Start measuring time
    calculateCube(range_start, range_end); // Calculate cube in parallel
    double end_parallel = omp_get_wtime(); // End measuring time
    printf("Parallel Execution Time: %lf seconds\n", end_parallel - start_parallel);


    return 0;
}
```

Output:



```
Parallel Execution:
Cube of 1 is: 1
Cube of 2 is: 8
Cube of 3 is: 27
Cube of 4 is: 64
Cube of 5 is: 125
Cube of 6 is: 216
Cube of 7 is: 343
Cube of 8 is: 512
Cube of 9 is: 729
Cube of 10 is: 1000
Parallel Execution Time: 0.001234
```

Conclusion: The parallel multiplication algorithm divides the task of multiplying matrices among multiple processors, enabling concurrent computation and thus reducing the overall execution time, especially for large matrices. In contrast, the sequential algorithm performs matrix multiplication sequentially on a single processor, resulting in longer execution times due to the lack of parallelism. The parallel approach maximizes CPU utilization by distributing the workload, offering significant performance improvements compared to its sequential counterpart.