GROUP NO: 4

Project Report

# MULTIPROCESSOR PROCESSOR TASK SCHEDULING PROBLEM USING BAT ALGORITHM

**Department of Computer Science & Engineering, Indian Institute of Technology, Roorkee**

**Submitted to:**

Prof. Neetesh Kumar

**Presented by:**

Jimmy Aghera (23535031)

Manan Shah (23535010)

Swaroop singh (23535023)

Piyush Tayal (23535021)

## Introduction

The project focuses on solving the Multiprocessor Processor Task Scheduling Problem using the Bat Algorithm. This problem involves assigning a set of tasks represented as a Directed Acyclic Graph (DAG) to a set of homogeneous processors in a way that minimizes the makespan, i.e., the total time taken to complete all tasks.

The key components of the project include:

**Task Representation:** Each task is represented by a node in the DAG, with each node containing information such as task number and processing time.
**Edge Representation:** Directed edges between nodes represent the precedence relations between tasks. An edge from task ni to task nj indicates that nj cannot start before ni completes.
**Communication Costs:** Communication costs between tasks are represented by weights on the edges. If tasks ni and nj are assigned to different processors, the communication cost must be added to the completion time of nj.
**Bat Algorithm:** The Bat Algorithm is used to optimize the task assignment. Bats represent potential solutions, and their positions are updated iteratively to find the best assignment that minimizes the makespan.
**Evaluation Function:** An evaluation function calculates the makespan for a given assignment of tasks to processors. This function considers task processing times, precedence relations, and communication costs.

The goal of the project is to demonstrate the effectiveness of the Bat Algorithm in solving the Multiprocessor Processor Task Scheduling Problem and to provide insights into its application in real-world scheduling scenarios.

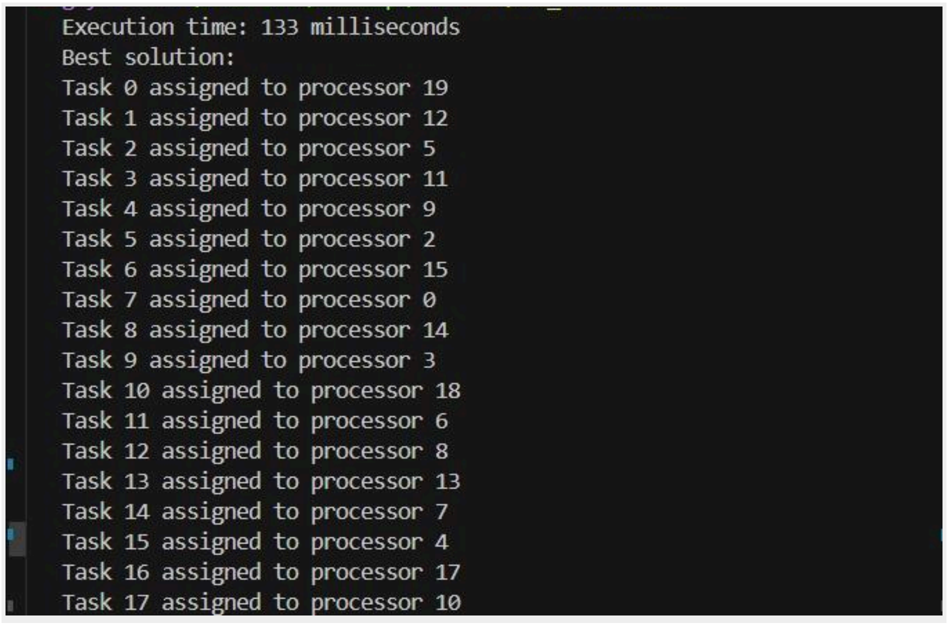
## Dataset

For Dataset as our project doesn't require data set from any source as specific we are making graphs on the random bases, you can see the data for the graph generation below.

```
TaskGraph initializeTaskGraph()
{
   TaskGraph graph;
   const int num_tasks = 100;
   const int num_edges = 200;
   srand(time(0));
   for (int i = 0; i < num_tasks; ++i){
      graph.tasks.push_back({i, rand() % 10 + 1});
   }
   for (int i = 0; i < num_edges; ++i)
   {
      int source_task = rand() % num_tasks;
      int destination_task = rand() % num_tasks;
      int communication_cost = rand() % 10 + 1;
graph.edges.push_back({source_task, destination_task, communication_cost});
   }
   return graph;
}
```

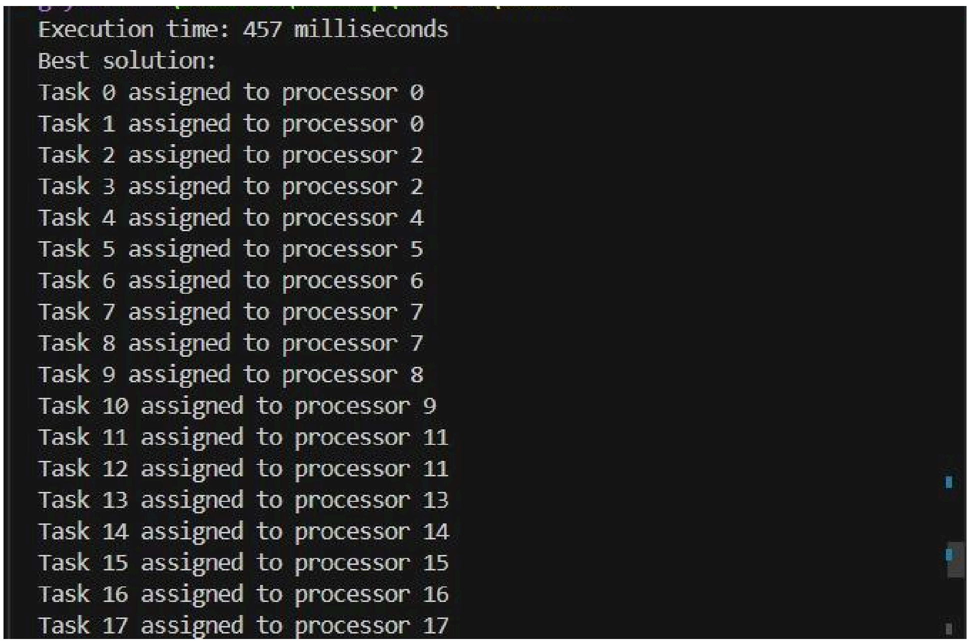
## Output

Sequential Output:



```
Execution time: 133 milliseconds
Best solution:
Task 0 assigned to processor 19
Task 1 assigned to processor 12
Task 2 assigned to processor 5
Task 3 assigned to processor 11
Task 4 assigned to processor 9
Task 5 assigned to processor 2
Task 6 assigned to processor 15
Task 7 assigned to processor 0
Task 8 assigned to processor 14
Task 9 assigned to processor 3
Task 10 assigned to processor 18
Task 11 assigned to processor 6
Task 12 assigned to processor 8
Task 13 assigned to processor 13
Task 14 assigned to processor 7
Task 15 assigned to processor 4
Task 16 assigned to processor 17
Task 17 assigned to processor 10
```

Parallel Output:

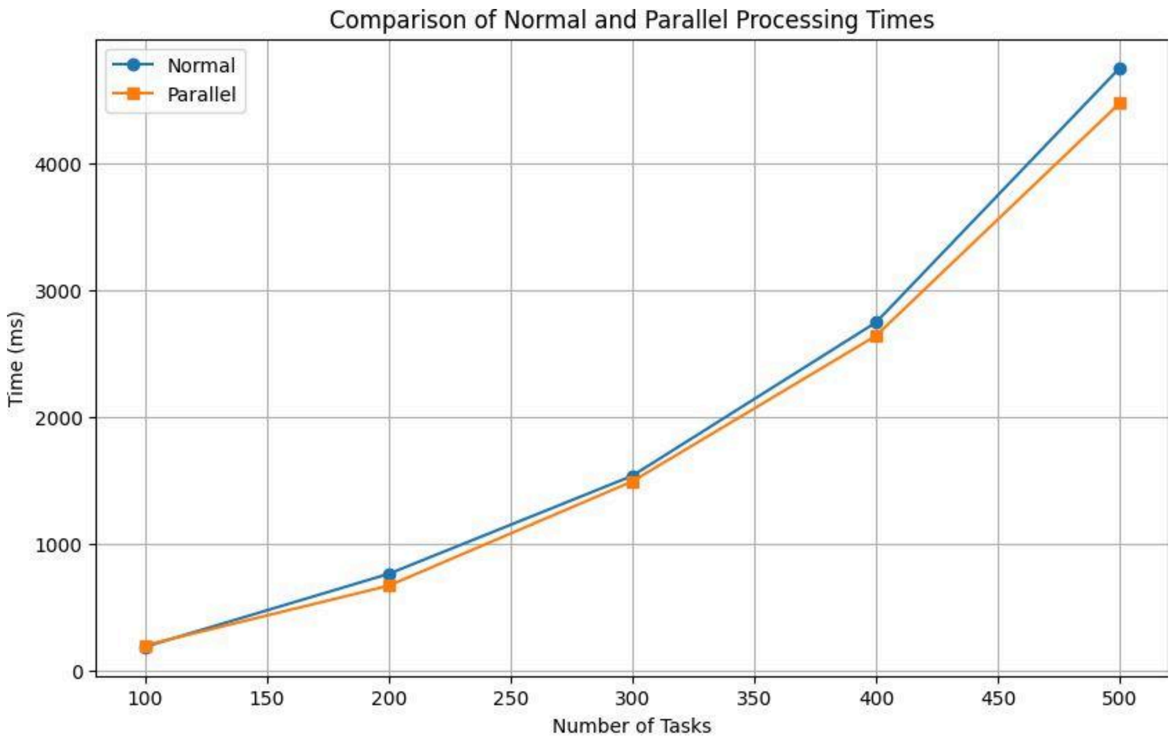


```
Execution time: 457 milliseconds
Best solution:
Task 0 assigned to processor 0
Task 1 assigned to processor 0
Task 2 assigned to processor 2
Task 3 assigned to processor 2
Task 4 assigned to processor 4
Task 5 assigned to processor 5
Task 6 assigned to processor 6
Task 7 assigned to processor 7
Task 8 assigned to processor 7
Task 9 assigned to processor 8
Task 10 assigned to processor 9
Task 11 assigned to processor 11
Task 12 assigned to processor 11
Task 13 assigned to processor 13
Task 14 assigned to processor 14
Task 15 assigned to processor 15
Task 16 assigned to processor 16
Task 17 assigned to processor 17
```

## Analysis/Result

| Tasks(n) | Serial execution(ms) | Parallel Execution(ms) |
|---|---|---|
| 100 | **186** | 198 |
| 200 | 761 | **668** |
| 300 | 1534 | **1490** |
| 400 | 2743 | **2638** |
| 500 | 4748 | **4471** |

The performance comparison was conducted based on **50 iterations**, using **20 processors** and **20 bats**. For each graph, the number of **edges was set to twice the number of tasks**. The results indicate that as the number of tasks increases, parallel execution with OpenMP becomes more efficient in reducing processing time for task scheduling problems.



Overall, the results demonstrate the effectiveness of parallel execution in reducing processing time for task scheduling problems, especially as the number of tasks increases. As the task size grows from 100 to 500, the parallel execution times consistently outperform the serial execution times. This trend underscores the scalability of parallelization, showcasing its ability to efficiently distribute and process tasks across multiple processors. Additionally, the relatively stable parallel execution times across the different task sizes suggest that parallelization can offer consistent performance gains even with varying task complexities. This highlights the potential of parallel execution to significantly enhance the efficiency of task scheduling algorithms, particularly in scenarios with large and computationally intensive task sets.

## Conclusion

The project implemented a task scheduling algorithm using the bat algorithm and compared its performance between serial and parallel execution. The algorithm aimed to assign a set of tasks to a group of processors, considering task dependencies and communication costs, with the goal of minimizing the makespan.

The results showed that parallel execution with OpenMP significantly reduced processing time compared to serial execution, especially as the number of tasks increased. This highlights the effectiveness of parallel computing in handling large-scale task scheduling problems. Additionally, the bat algorithm demonstrated its capability to efficiently explore the solution space and find near-optimal solutions.

In conclusion, the project demonstrated the potential of parallel computing and metaheuristic algorithms like the bat algorithm in solving complex task scheduling problems. The findings can be valuable for optimizing task scheduling in multiprocessor systems, leading to improved efficiency and resource utilization.