ANT COLONY OPTIMIZATION
(USING OPENMP)

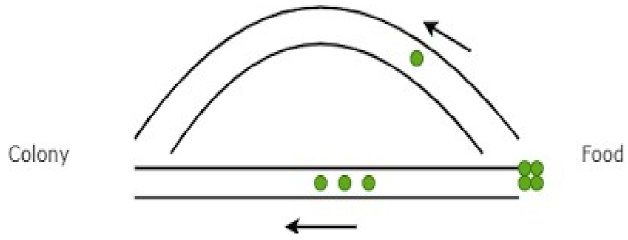
Team members:

-> Meghana M(21011101075)

-> Satvika M (21011101113)

Description:

The problem at hand involves simulating an ant colony optimization algorithm for pathfinding in a 3D environment represented by a map. The ants are tasked with finding the shortest path between their home and a food source while depositing pheromones along the way to communicate and determine the optimal path.



Approach:

- Map Representation: The 3D environment is represented by a map, with dimensions specified by constants such as MAP_X, MAP_Y, and MAP_Z. The home and food source locations are also defined within this map.

- Ant Initialization: A total of TOTAL_ANTS ants are initialized with random starting positions within the map. Each ant is assigned a maximum amount of pheromone it can carry (MAX_HORM_LEFT), and its initial state is set to explore (-1).

- Simulation Loop:

  Update Hormone Map: In each iteration of the simulation loop, ants deposit pheromones based on their current positions and states.

  Ant Movement: Ants move within the map based on a set of rules influenced by the pheromone concentrations in their vicinity. The movement of ants is parallelized using OpenMP directives to exploit multi-threading capabilities.

  Pheromone Decline: Pheromone concentrations gradually decline over time to simulate the evaporation of pheromones.

  Print Information: Periodically, information such as the number of ants in each state is printed to monitor the simulation progress.

- Performance Measurement:

  Baseline Execution Time: The execution time of the simulation is measured for a single processing element (thread) to establish a baseline.

  Speedup Calculation: The speedup is calculated as the ratio of the baseline execution time to the execution time with multiple processing elements. This metric indicates the performance improvement achieved through parallelization.

  Parallel Efficiency Calculation: The parallel efficiency represents the ratio of the speedup achieved using multiple processing elements to the number of processing elements. It quantifies the effectiveness of utilizing additional processing elements in improving performance.

- Graphical Analysis:

  Speedup vs. Number of Threads: Graph 1 illustrates the speedup achieved with varying numbers of processing elements for different problem sizes (N).

  Speedup vs. Problem Size: Graph 2 shows how the speedup varies with different problem sizes for fixed numbers of processing elements.

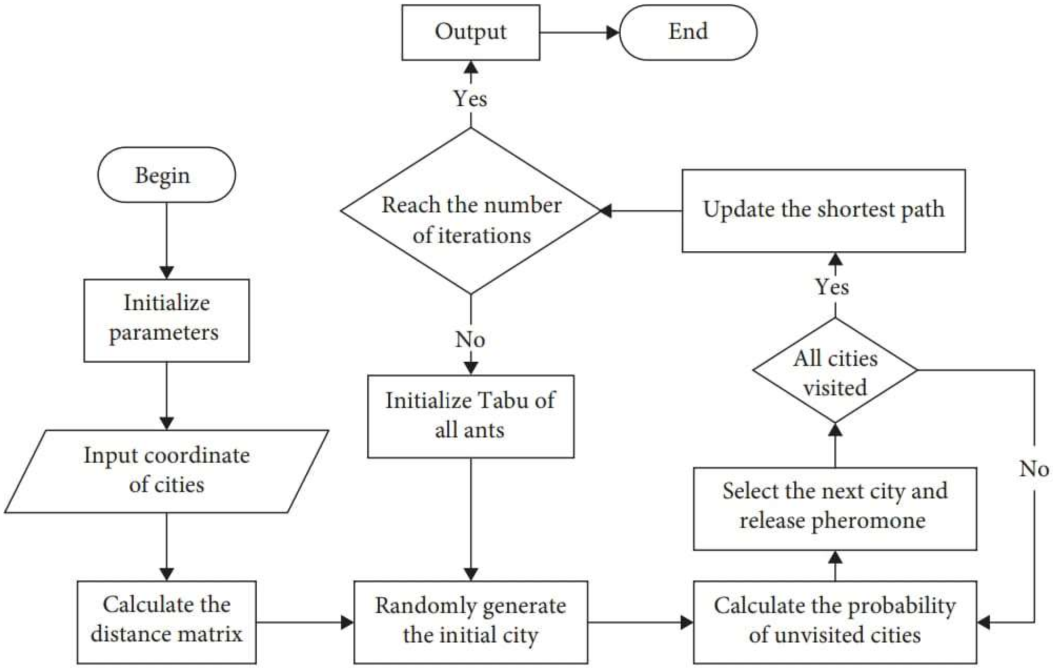
Advantages of High-Performance Computing (HPC):

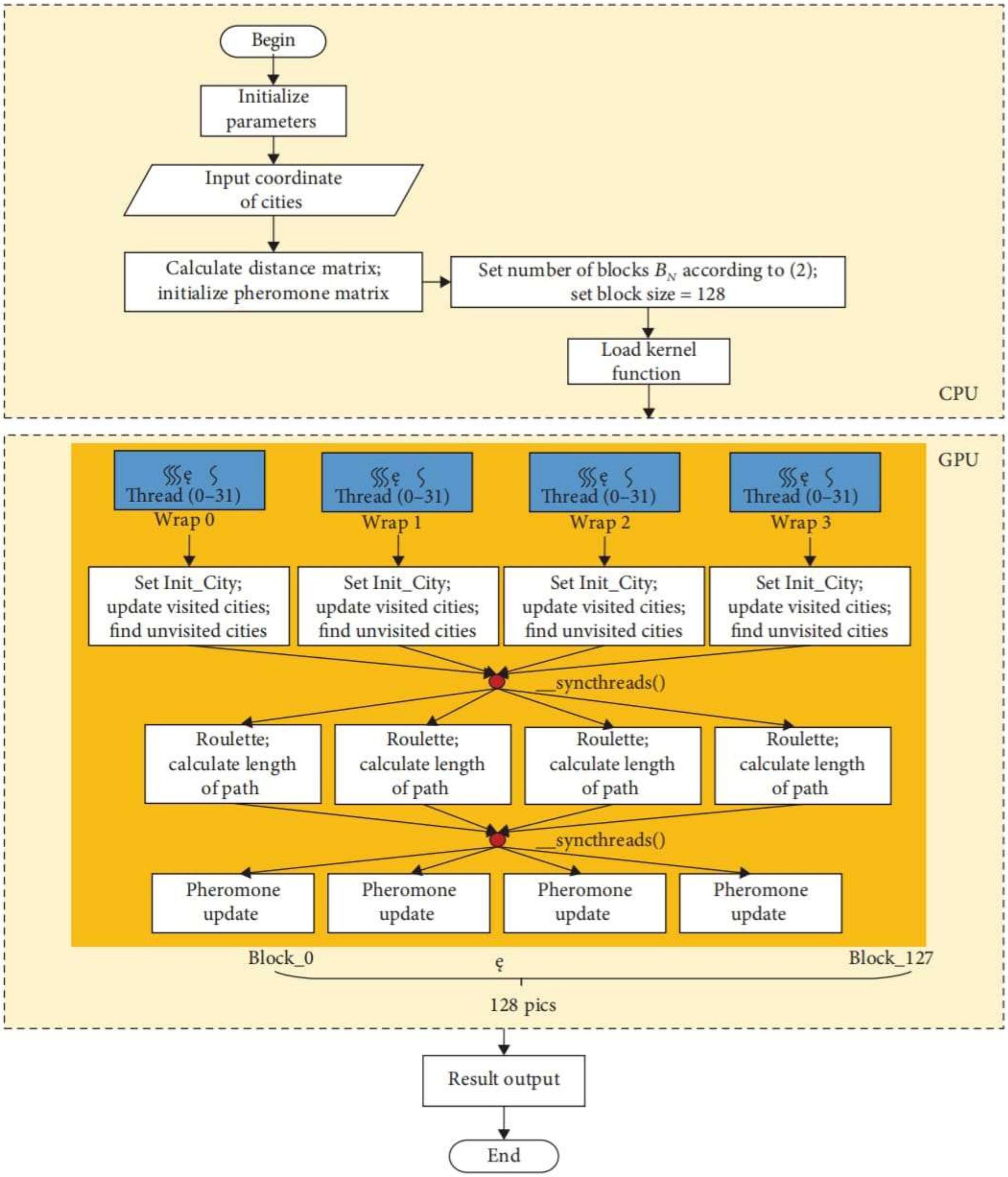
- Performance Improvement: HPC techniques such as parallelization significantly reduce execution times for computationally intensive tasks.
- Scalability: The ability to scale the number of processing elements allows efficient utilization of computational resources, enabling larger problem sizes to be tackled within reasonable time frames.
- Exploration of Solution Space: Parallel execution facilitates the exploration of a larger solution space, enhancing the quality of solutions obtained by optimization algorithms.

Applications of ACO:

- Traveling Salesman Problem
- Vehicle Routing Problem:
  routes of a fleet of vehicles to deliver goods to a set of customers.
- Job Scheduling:
  minimizes makespan or total processing time
- Network Routing:
  efficient paths for data transmission in communication network

Block Diagram:


Output
End
Yes
Begin
Reach the number of iterations
Update the shortest path
Initialize parameters
Yes
No
All cities visited
Initialize Tabu of all ants
Input coordinate of cities
No
Select the next city and release pheromone
Calculate the distance matrix
Randomly generate the initial city
Calculate the probability of unvisited cities

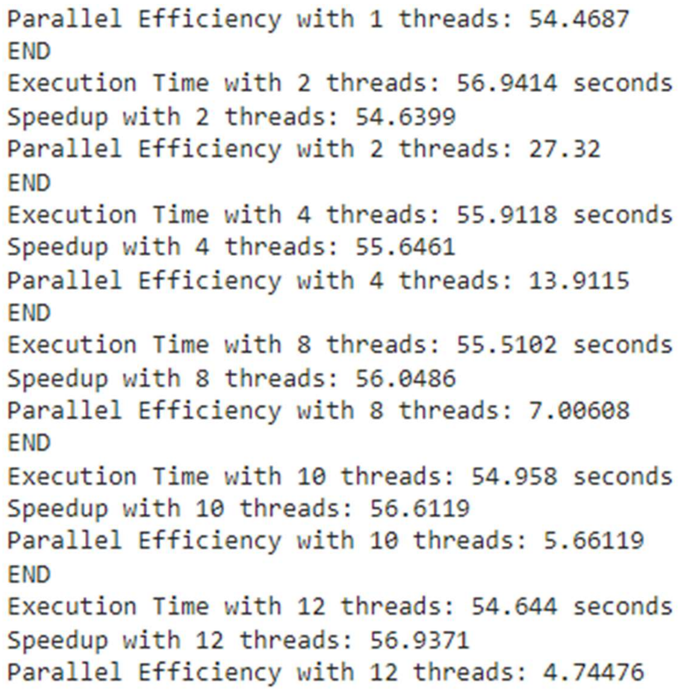

Begin
Initialize parameters
Input coordinate of cities
Calculate distance matrix; initialize pheromone matrix
Set number of blocks B_N according to (2); set block size = 128
Load kernel function
CPU
GPU
Thread (0–31)
Wrap 0
Thread (0–31)
Wrap 1
Thread (0–31)
Wrap 2
Thread (0–31)
Wrap 3
Set Init_City; update visited cities; find unvisited cities
Set Init_City; update visited cities; find unvisited cities
Set Init_City; update visited cities; find unvisited cities
Set Init_City; update visited cities; find unvisited cities
__syncthreads()
Roulette; calculate length of path
Roulette; calculate length of path
Roulette; calculate length of path
Roulette; calculate length of path
__syncthreads()
Pheromone update
Pheromone update
Pheromone update
Pheromone update
Block_0
Block_127
128 pics
Result output
End

Parallel Implementation of ACO:

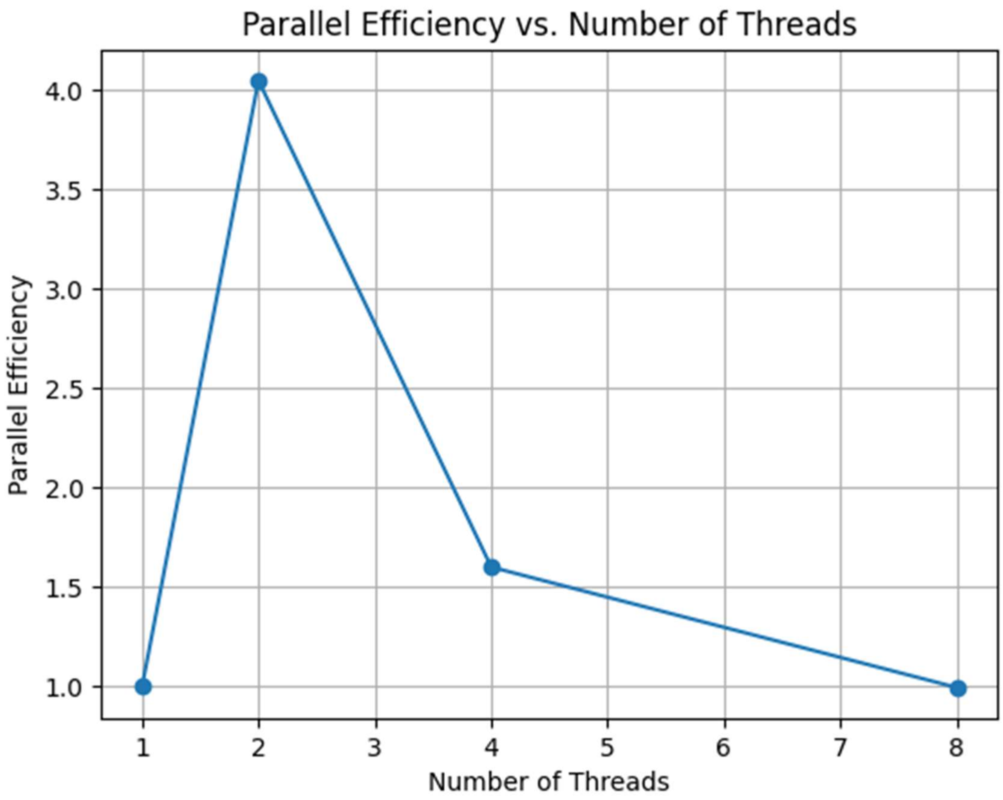
- horm_map_decline():
  parallelizes a loop over map points to update the pheromone levels. The loop iterations are divided among multiple threads to accelerate the computation.
- ant_move():

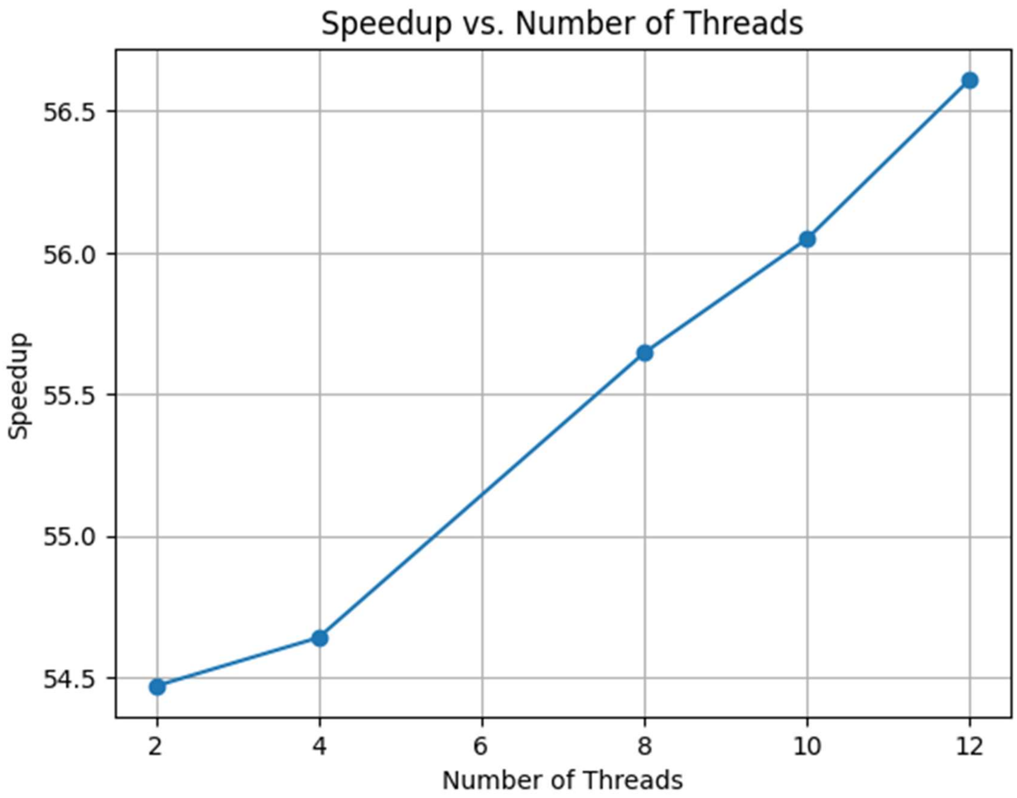
the #pragma omp parallel for directive parallelizes the loop over the ants. The loop is divided into chunks, and each chunk is executed by a separate thread. The num_threads(4) directive specifies that four threads should be used for parallel execution, and schedule(dynamic, 512) specifies the scheduling policy for distributing loop iterations among threads.

Graphs:



```
Parallel Efficiency with 1 threads: 54.4687
END
Execution Time with 2 threads: 56.9414 seconds
Speedup with 2 threads: 54.6399
Parallel Efficiency with 2 threads: 27.32
END
Execution Time with 4 threads: 55.9118 seconds
Speedup with 4 threads: 55.6461
Parallel Efficiency with 4 threads: 13.9115
END
Execution Time with 8 threads: 55.5102 seconds
Speedup with 8 threads: 56.0486
Parallel Efficiency with 8 threads: 7.00608
END
Execution Time with 10 threads: 54.958 seconds
Speedup with 10 threads: 56.6119
Parallel Efficiency with 10 threads: 5.66119
END
Execution Time with 12 threads: 54.644 seconds
Speedup with 12 threads: 56.9371
Parallel Efficiency with 12 threads: 4.74476
```


Parallel Efficiency vs. Number of Threads
Parallel Efficiency
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1
2
3
4
5
6
7
8
Number of Threads


Speedup vs. Number of Threads
Speedup
56.5
56.0
55.5
55.0
54.5
2
4
6
8
10
12
Number of Threads

References:

- https://downloads.hindawi.com/journals/ietcdt/2023/9915769.pdf?_gl=1*1mb0ssr*_ga*MTE3MDUzNTI2MS4xNzA2MTE4NjM1*_ga_NF5QFMJT5V*MTcwNjE0Mjc4OS4zLjEuMTcwNjE0MzAwOC40MS4wLjA.&_ga=2.170033952.1045504058.1706118636-1170535261.1706118635
- https://github.com/icecat2012/parallel-ant-colony-optimization/tree/master/CUDA
- https://tommliu.github.io/Parallel-Ant-Colony-System/

Github Links:

- https://github.com/Megha2306/Parallelization-of-Ant-Colony-Optimization-using-openmp
- https://github.com/satvika03/Ant-Colony-Optimization