ACCELERATING EVOLUTION:

PARALLELISATION OF EVOLUTIONARY ALGORITHM ON GPU

ASHIRBAD SARANGI

SC23M002

CONTENT

OBJECTIVE

To try and implement evolutionary algorithm - ACO and compare the run time in CPU and GPU

GPU PRE-REQUISITES

GPU PRE-REQUISITES

- GPU – Graphics Processing Unit
- Fast amount of dense matrix multiplication due to parallelism

CUDA Basic Linear Algebra Subprograms

cuBLAS

WHAT IS CUBLAS GOOD FOR ?

- Anything that uses heavy linear algebra computations (on dense matrices) can likely benefit from GPU acceleration
  - Graphics
  - Machine learning
  - Computer vision
  - Physical simulations
  - Finance
  - etc…..

- cuBLAS excels in situations where the performance is needed to be maximized by batching multiple kernels using streams.
  - Like making many small matrix-matrix multiplications on dense matrices

- cuBLAS selected column-first indexing

FEATURES

- All of the functions defined in cuBLAS have four versions which correspond to the four types of numbers in CUDA C
  - S, s : single precision (32 bit) real float
  - D, d : double precision (64 bit) real float
  - C, c : single precision (32 bit) complex float (implemented as a float2)
  - Z, z : double precision (64 bit) complex float
  - H, h : half precision (16 bit) real float

- Functions used for Matrix / Vector Multiplication :
  - cublasSgemm ⟶ cublas S gemm
  - cublasHgemm
  - cublasDgemv ⟶ cublas D gemv

ARRAY INDEXING

The arrays are linearized into one dimension, so we will use an <u>indexing macro</u>.

```
#define IDX2C(i,j,ld) (((j)*(ld))+(i))
```

Where “i” is the row, “j” is the column, and “ld” is the leading dimension.

In column major storage “ld” is the number of rows.

NUMPY VS CUBLAS

Numpy	math	cuBLAS (<T> is one of S, D, C, Z, H)
numpy.matmul(**α**, **χ**)	$(\lambda\mathbf{A})_{ij} = \lambda(\mathbf{A})_{ij}$	cublas<T>gemm(**α**, **χ**)
numpy.dot(**χ**, **γ**) (Multiply arguments element-wise)	$(A \circ B)_{i,j} = (A)_{i,j}(B)_{i,j}.$	cublas<T>gemm(**χ**, **γ**)
numpy.matmul(**A**,**χ**)	**A**χ = C	cublas<T>gemm(**χ**, **A**)
numpy.matmul(**A**, **B**)	$\boldsymbol{C} \leftarrow \alpha\boldsymbol{A}\boldsymbol{B} + \beta\boldsymbol{C}$	cublas<T>gemm(**A**, **B**)

EVOLUTIONARY ALGORITHMS

EVOLUTIONARY ALGORITHMS

- Evolutionary Algorithms (EAs) are a family of optimization algorithms inspired by the process of natural selection. They are used to find approximate solutions to optimization and search problems.

- Key Concepts :
  - Natural Selection:
    - Mimics the process of natural selection where individuals with favorable traits are more likely to survive and reproduce.
  - Population:
    - Solutions are represented as individuals in a population. Multiple solutions coexist and evolve over generations.
  - Crossover and Mutation:
    - Individuals undergo genetic operations like crossover (recombination) and mutation to create new offspring.
  - Fitness Function:
    - Measures the quality of an individual. Individuals with higher fitness values are more likely to contribute to the next generation.

ANT COLONY OPTIMISATION

```
Initialize pheromone levels

Repeat for a fixed number of iterations or until a convergence
criterion is met:

    Place ants at the starting point

    For each ant:

        Construct a solution by probabilistically selecting components

    Update pheromone levels based on the constructed solutions

    Evaporate pheromones
```

PROJECT

AIM

To try to find out a path that touches important cities of India using ACO and compare the run time in CPU and GPU

PHASES OF PROJECT

Phase I

Process map of India and extract the pixel values of important cities.

Phase II

Code ACO in CPU and plot a solution path in the map based on the extracted points

Phase III

Repeat the same thing in GPU

PHASES OF PROJECT

Phase I

Process map of India and extract the pixel values of important cities.

Phase II

Code ACO in CPU and plot a solution path in the map based on the extracted points

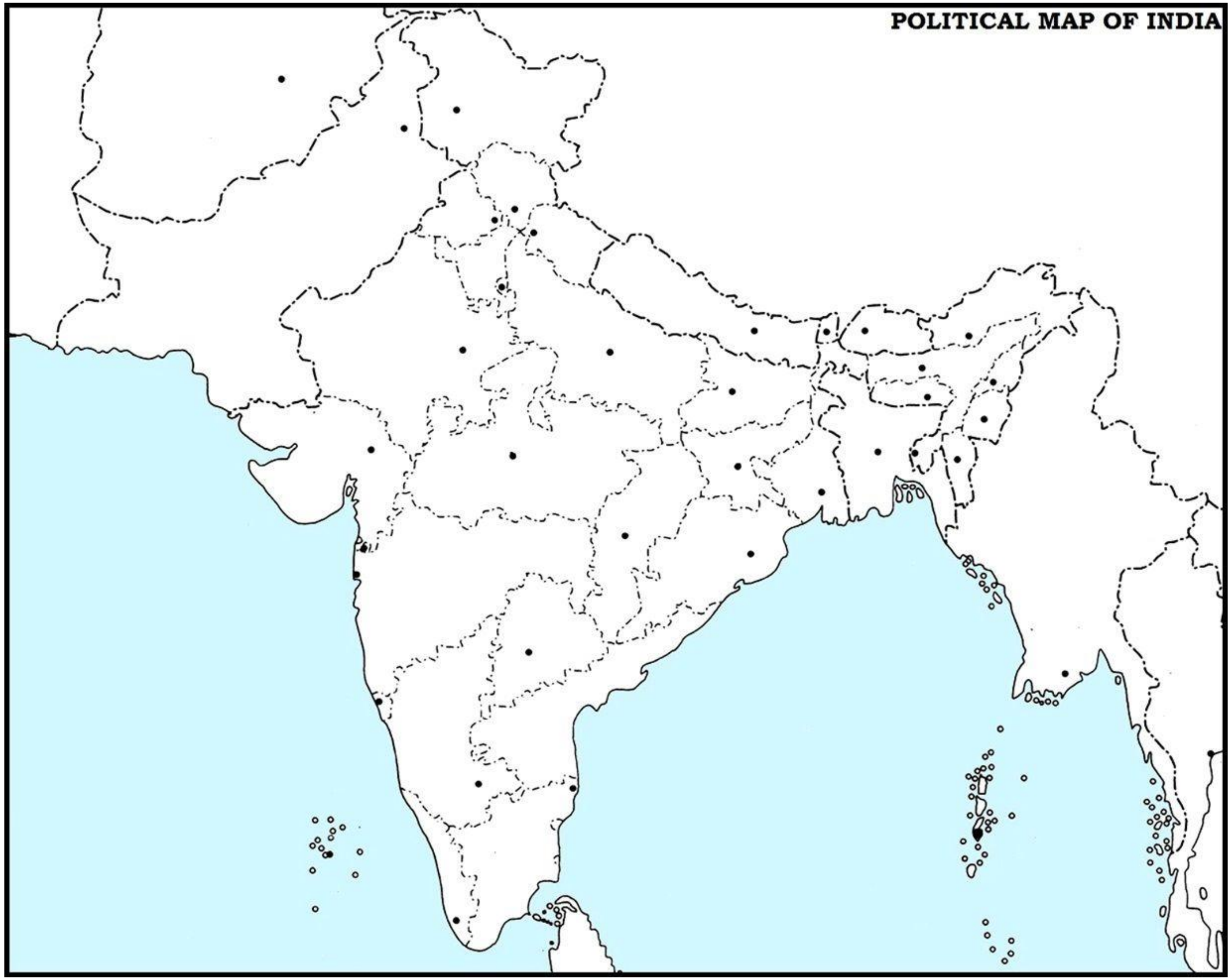
Phase III

Repeat the same thing in GPU

AIM

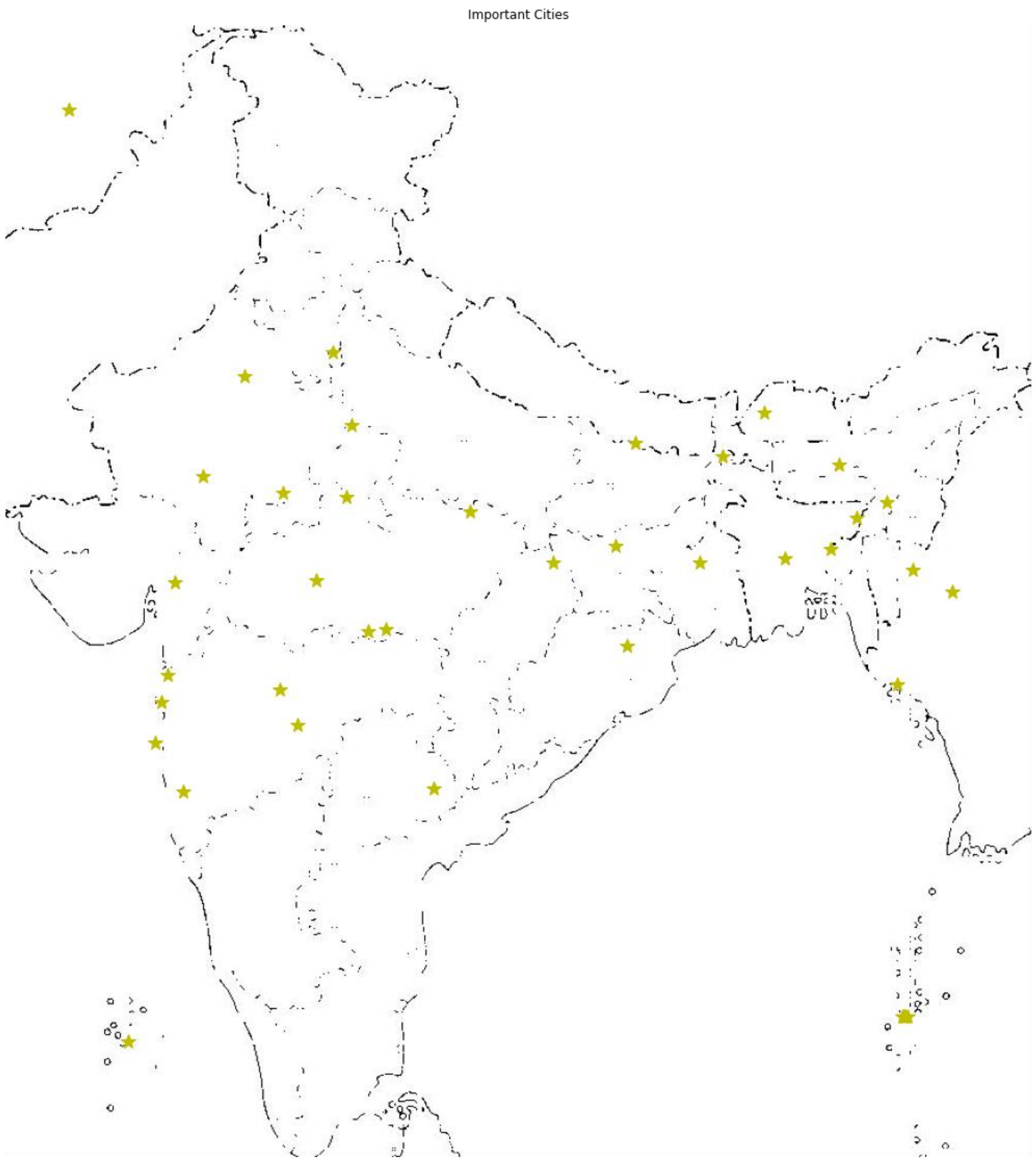
To try to find out a path that touches important cities of India using ACO and compare the run time in CPU and GPU

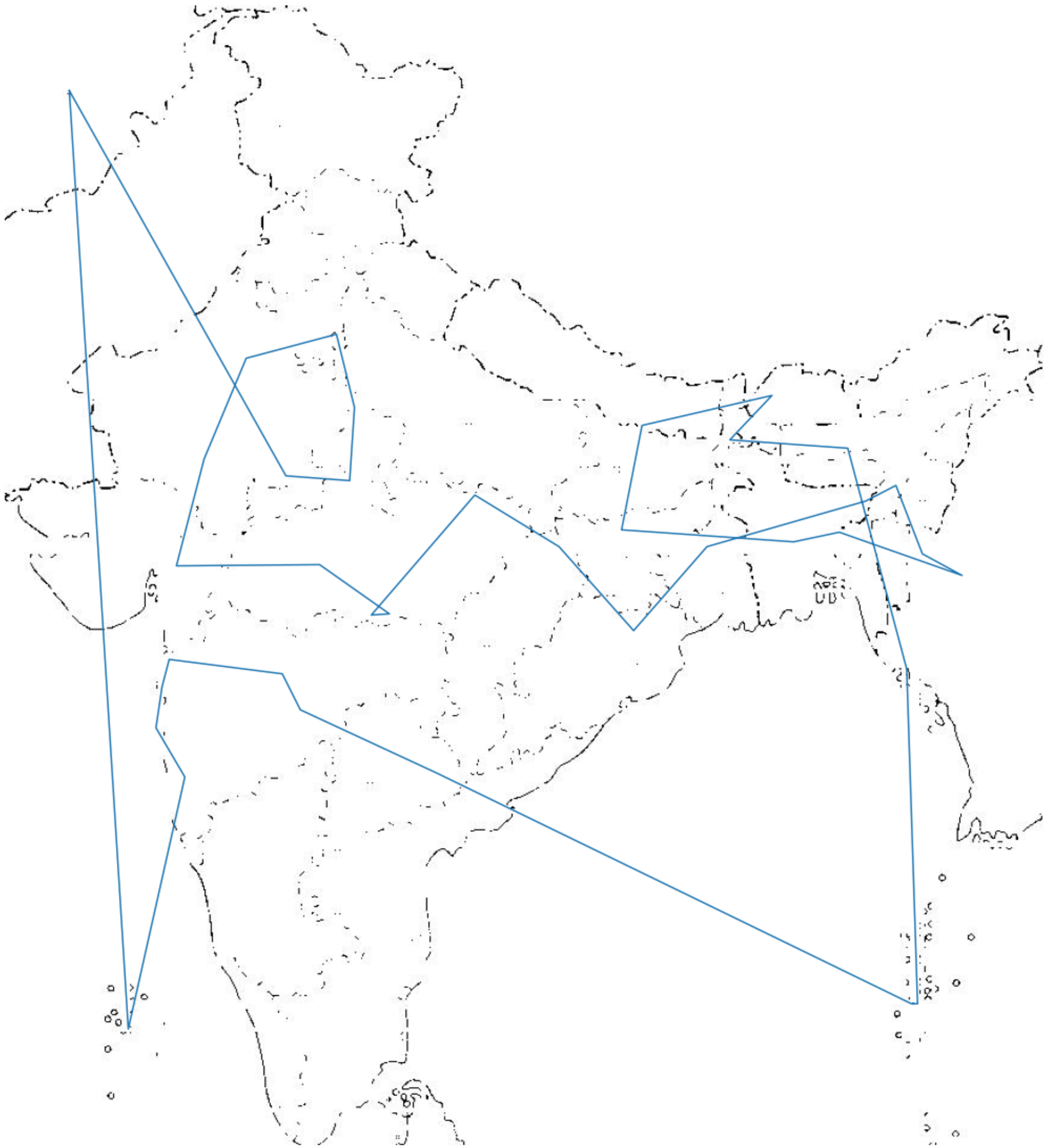
INITIAL MAP



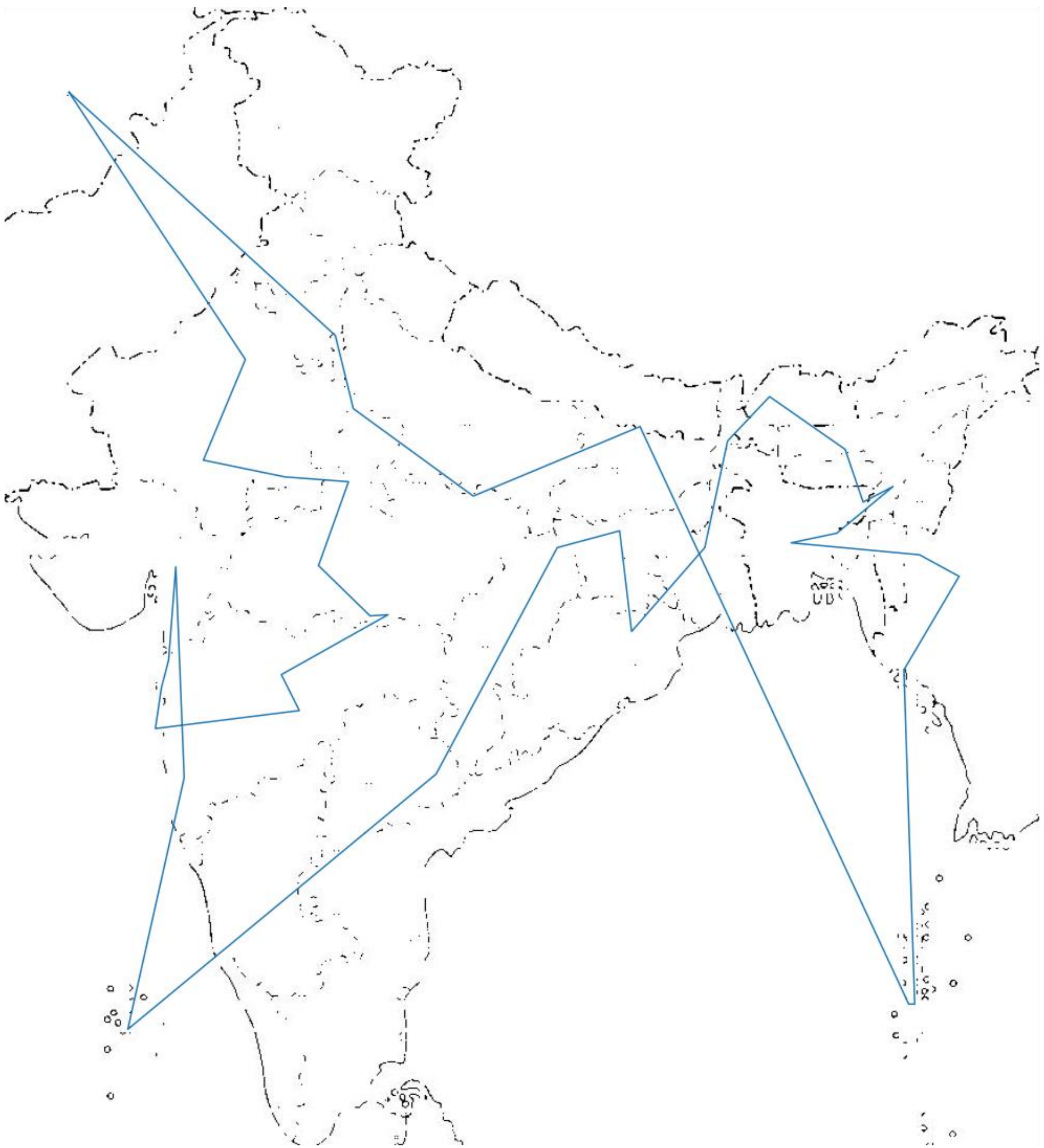
INITIAL MAP

MAP WITH CITIES





CPU Result



GPU Result

COMPARISONS

```
GPU Results
New Path Found !
With :
Path Cost : 6547 in 6282 epochs completing  1 round trips

CPU Results
New Path Found !
With :
Path Cost : 6748 in 6233 epochs completing  1 round trips

Time taken by GPU is :  31.35 s
Time taken by CPU is :  31.57 s
```

Time Taken by GPU is 0.7% less than CPU and the path cost is 2.97% less

```
GPU Results
New Path Found !
With :
Path Cost : 6923 in 6338 epochs completing  1 round trips

CPU Results
New Path Found !
With :
Path Cost : 7412 in 6631 epochs completing  1 round trips

New Path Found !
With :
Path Cost : 7239 in 6842 epochs completing  1 round trips

Time taken by GPU is :  31.13 s
Time taken by CPU is :  31.22 s
```

Time Taken by GPU is 0.28% less than CPU and the path cost is 4.36% less

```
GPU Results
New Path Found !
With :
Path Cost : 4523 in 3718 epochs completing  1 round trips

New Path Found !
With :
Path Cost : 4505 in 53500 epochs completing  14 round trips

New Path Found !
With :
Path Cost : 4503 in 61339 epochs completing  16 round trips

New Path Found !
With :
Path Cost : 4423 in 93288 epochs completing  24 round trips

New Path Found !
With :
Path Cost : 4365 in 397074 epochs completing  100 round trips

CPU Results
New Path Found !
With :
Path Cost : 4529 in 3724 epochs completing  1 round trips

New Path Found !
With :
Path Cost : 4528 in 11290 epochs completing  3 round trips

New Path Found !
With :
Path Cost : 4394 in 30755 epochs completing  8 round trips

New Path Found !
With :
Path Cost : 4298 in 249233 epochs completing  64 round trips

Time taken by GPU is :  4.28 s
Time taken by CPU is :  4.31 s
```

Overall time taken by GPU is 0.7% less than whereas time taken per epoch by GPU is ~50% less than time taken by CPU

```
GPU Results
New Path Found !
With :
Path Cost : 4505 in 3700 epochs completing  1 round trips

New Path Found !
With :
Path Cost : 4424 in 7343 epochs completing  2 round trips

New Path Found !
With :
Path Cost : 4274 in 64829 epochs completing  17 round trips

New Path Found !
With :
Path Cost : 4032 in 840052 epochs completing  212 round trips

CPU Results
New Path Found !
With :
Path Cost : 4529 in 3724 epochs completing  1 round trips

New Path Found !
With :
Path Cost : 4506 in 11160 epochs completing  3 round trips

New Path Found !
With :
Path Cost : 4418 in 49184 epochs completing  13 round trips

New Path Found !
With :
Path Cost : 4416 in 580603 epochs completing  149 round trips

Time taken by GPU is :  4.17 s
Time taken by CPU is :  4.23 s
```

Overall time taken by GPU is 1.41% less than whereas time taken per epoch by GPU is ~31% less than time taken by CPU

```
GPU Results
New Path Found !
With :
Path Cost : 4588 in 3746 epochs completing  1 round trips

New Path Found !
With :
Path Cost : 4529 in 7470 epochs completing  2 round trips

New Path Found !
With :
Path Cost : 4523 in 22871 epochs completing  6 round trips

New Path Found !
With :
Path Cost : 4433 in 26523 epochs completing  7 round trips

New Path Found !
With :
Path Cost : 4394 in 568613 epochs completing  143 round trips

CPU Results
New Path Found !
With :
Path Cost : 4528 in 3723 epochs completing  1 round trips

New Path Found !
With :
Path Cost : 4298 in 7240 epochs completing  2 round trips

New Path Found !
With :
Path Cost : 4291 in 131142 epochs completing  34 round trips

Time taken by GPU is :  4.11 s
Time taken by CPU is :  4.15 s
```

Overall time taken by GPU is 1% less than whereas time taken per epoch by GPU is ~77% less than time taken by CPU

```
GPU Results
New Path Found !
With :
Path Cost : 7234 in 6453 epochs completing  1 round trips

New Path Found !
With :
Path Cost : 6900 in 6548 epochs completing  1 round trips

CPU Results
New Path Found !
With :
Path Cost : 7100 in 6319 epochs completing  1 round trips

Time taken by GPU is :  47.13 s
Time taken by CPU is :  47.01 s
```

GPU Result

The path cost is 2.81% less in GPU and the time taken per epoch in GPU is around ~50% less

THANK YOU