ACCELERATING EVOLUTION:

PARALLELISATION OF EVOLUTIONARY ALGORITHMS ON GPU

ASHIRBAD SARANGI

SC23M002

CONTENT

OBJECTIVE

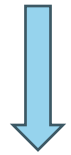
To try and implement evolutionary algorithms and compare the run time in CPU and GPU

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 indexing macro.

```
#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.

GENETIC ALGORITHM

Initialise population size, number of generations, cross over rate and mutation rate

Repeat till number of generations :

- check fitness
- Select Fit parents
- Generate Offsprings from cross over rate
- Mutate offsprings
- Replace Population

Get Best solution

ANT COLONY OPTIMIATION

```
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
```

PARTICLE SWARM OPTIMIZATION

```
Initialise number of particles, number of parameters, max iterations, cognitive parameters, social parameters, inertia weight

particles = initialize_particles(num_particles, num_dimensions)

For each particle initialize best position and fitness both personally and Collectively as Globally

Until max iterations:
    Update velocity
    Update position
    Check fitness
    Update personal best if current fitness is better than best fitness
    Update Global best

best_solution = global_best_position
```

PLAN OF ACTION

PLAN OF ACTION

GENERATE A PATH TO FOLLOW

To implement the algo, a unique path needs to be present.
Trying to generate a path using the India map

LEARN THE ALGORITHMS

Learn all the three Algorithms

IMPLEMENT THEM ON CPU

Implement the algos in the path decided through the map on CPU

IMPLEMENT ON GPU

Implement the same on GPU

COMPARE

Compare performance of both techniques and report

PLAN OF ACTION

GENERATE A PATH TO FOLLOW

To implement the algo, a unique path needs to be present.
Trying to generate a path using the India map

LEARN THE ALGORITHMS

Learn all the three Algorithms

IMPLEMENT THEM ON CPU

Implement the algos in the path decided through the map on CPU

IMPLEMENT ON GPU

Implement the same on GPU

COMPARE

Compare performance of both techniques and report

TIMELINE

TIMELINE

COMPLETELY LEARN THE ALGO

By 9th November

IMPLEMENTATION ON CPU

Start by 9th November
Complete by 10th November

IMPLEMENT THEM ON GPU

Start by 11th November
Complete by 15th November

COMPARISON

Start by 16th November
Complete by 17th November

REPORT

Start by 18th November
Complete by 20th November

THANK YOU