AMRUTVAHINI COLLEGE OF ENGINEERING, SANGAMNER

DEPARTMENT OF COMPUTER ENGINEERING

2023-2024

LP-V Mini Project report (HPC)

# on

‘Implement Non-Serial Polyadic Dynamic Programming with GPU Parallelization’



# BE Computer Engineering BY

Mr. Gaurav Bhima Chaudhari(4117) Mr. Harsh Sunil Chhajed(4118) **Guided by** - Prof. S. R. Pandit

* **Title:** Implement Non-Serial Polyadic Dynamic Programming with GPU Parallelization
* **Objectives:**
  1. Algorithm Design: Develop a detailed algorithm for NSP-DP that incorporates parallelization strategies suitable for GPU architecture. Consider efficient data structures and parallel computing techniques to exploit the GPU's parallel processing power.
  2. GPU Integration: Implement the NSP-DP algorithm using a GPU programming framework such as CUDA or OpenCL. Utilize GPU kernels to parallelize the computation of subproblems and leverage shared memory for efficient data access.
  3. Benchmarking and Performance Analysis: Conduct comprehensive benchmarking tests to evaluate the performance of the GPU-accelerated NSP-DP implementation. Compare its execution time and scalability with CPU-based implementations for various problem sizes and dimensions.
  4. Optimization Strategies: Explore optimization strategies to further enhance the performance of the GPU-accelerated NSP-DP implementation. This may include optimizing memory access patterns, reducing synchronization overhead, and exploiting GPU architecture features like warp divergence.

## Algorithm:

## Initialization:

## Initialize data structures and memory on the GPU.

## Transfer necessary input data to the GPU memory.

## Dynamic Programming:

## Iterate over stages and subproblems.

## Use GPU kernels to parallelize the computation of subproblems.

## Update the value function and policy in each stage.

## Termination:

## Retrieve results from GPU memory.

## Post-process and output the final solution.

## Implementation Steps:

## Initialize GPU Environment:

## Set up CUDA environment.

## Allocate GPU memory for input data and result storage.

## import numpy as np

## from numba import cuda

## cuda.select\_device(0) # Select GPU device 0

## cuda.close() # Close any existing CUDA contexts

## cuda.select\_device(0) # Re-select GPU device 0

## # Allocate GPU memory for data and result

## data\_gpu = cuda.to\_device(data)

## result\_gpu = cuda.device\_array(shape)

## Define CUDA Kernel for Parallel Computation:

## Implement CUDA kernel for parallel computation of subproblems

## @cuda.jit

## def nsp\_dp\_kernel(data\_gpu, result\_gpu):

## # Implementation of NSP-DP algorithm

## # Each thread handles a subproblem in parallel

## # Modify result\_gpu in-place

## Launch CUDA Kernel:

## Launch the CUDA kernel to perform parallel computation.

## # Define grid and block dimensions for CUDA kernel

## threads\_per\_block = 128

## blocks\_per\_grid = (data\_size + (threads\_per\_block - 1)) // threads\_per\_block

## # Launch CUDA kernel

## nsp\_dp\_kernel[blocks\_per\_grid, threads\_per\_block](data\_gpu, result\_gpu)

## Synchronize GPU:

## Ensure all GPU computations are completed before proceeding.

## cuda.synchronize()

## Retrieve Results from GPU:

## Copy the final results from GPU memory back to CPU memory.

## # Retrieve results from GPU

## cuda.memcpy\_dtoh(result, result\_gpu)

## Output Solution:

## Post-process and output the final solution.

## print("Optimal solution:", result)

## Outout :

