!pip install pycuda

import numpy as np

import pycuda.autoinit

import pycuda.driver as cuda

from pycuda.compiler import SourceModule

import time

# CUDA kernel for frequency counting

freq\_kernel\_code = """

\_\_global\_\_ void computeFrequencies(unsigned char \*input, int \*frequencies, int size) {

    int idx = blockIdx.x \* blockDim.x + threadIdx.x;

    if (idx < size) {

        atomicAdd(&frequencies[input[idx]], 1);

    }

}

"""

# CUDA kernel for prefix sum (scan) to build Huffman tree

scan\_kernel\_code = """

\_\_global\_\_ void prefixSum(int \*input, int \*output, int size) {

    int idx = blockIdx.x \* blockDim.x + threadIdx.x;

    if (idx < size) {

        output[idx] = input[idx];

        for (int i = 1; i <= idx; i++) {

            output[idx] += input[i-1];

        }

    }

}

"""

class HuffmanNode:

    def \_\_init\_\_(self, char, freq):

        self.char = char

        self.freq = freq

        self.left = None

        self.right = None

def build\_huffman\_tree(frequencies):

    nodes = [HuffmanNode(i, freq) for i, freq in enumerate(frequencies) if freq > 0]

    while len(nodes) > 1:

        nodes.sort(key=lambda x: x.freq)

        left = nodes.pop(0)

        right = nodes.pop(0)

        parent = HuffmanNode(None, left.freq + right.freq)

        parent.left = left

        parent.right = right

        nodes.append(parent)

    return nodes[0] if nodes else None

def generate\_codes(root, current\_code="", codes=None):

    if codes is None:

        codes = {}

    if root is None:

        return codes

    if root.char is not None:

        codes[root.char] = current\_code or "0"

    generate\_codes(root.left, current\_code + "0", codes)

    generate\_codes(root.right, current\_code + "1", codes)

    return codes

def huffman\_encode\_gpu(input\_string):

    # Convert input to numpy array

    input\_array = np.frombuffer(input\_string.encode(), dtype=np.uint8)

    input\_size = len(input\_array)

    # Allocate GPU memory

    input\_gpu = cuda.mem\_alloc(input\_array.nbytes)

    frequencies\_gpu = cuda.mem\_alloc(256 \* 4)  # 256 possible ASCII chars

    frequencies = np.zeros(256, dtype=np.int32)

    # Copy input to GPU

    cuda.memcpy\_htod(input\_gpu, input\_array)

    cuda.memcpy\_htod(frequencies\_gpu, frequencies)

    # Compile and run frequency kernel

    mod = SourceModule(freq\_kernel\_code)

    compute\_frequencies = mod.get\_function("computeFrequencies")

    block\_size = 256

    grid\_size = (input\_size + block\_size - 1) // block\_size

    compute\_frequencies(input\_gpu, frequencies\_gpu, np.int32(input\_size),

                       block=(block\_size, 1, 1), grid=(grid\_size, 1))

    # Copy frequencies back to host

    cuda.memcpy\_dtoh(frequencies, frequencies\_gpu)

    # Build Huffman tree and generate codes

    root = build\_huffman\_tree(frequencies)

    codes = generate\_codes(root)

    # Encode the input

    encoded = ""

    for char in input\_string:

        encoded += codes[ord(char)]

    return encoded, codes

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

    input\_string = "hello world this is a test string for huffman encoding"

    start\_time = time.time()

    encoded, codes = huffman\_encode\_gpu(input\_string)

    print(f"Original string: {input\_string}")

    print(f"Encoded string: {encoded}")

    print(f"Codes: {codes}")

    print(f"Time taken: {time.time() - start\_time:.4f} seconds")

Output:

Original string: hello world this is a test string for huffman encoding

Encoded string: 01010011011001101010111001011010011101101101111111000101100010111111000101111111010111110000111011110011110111100011110001001000011101001010011111101010010001000100000111101010011110011100100010101011011100010010000

Codes: {103: '0000', 99: '00010', 109: '00011', 117: '00100', 119: '00101', 101: '0011', 102: '0100', 104: '0101', 108: '0110', 114: '0111', 105: '1000', 110: '1001', 111: '1010', 115: '1011', 116: '1100', 97: '11010', 100: '11011', 32: '111'}

Time taken: 1.0239 seconds

**Instructions to execute the code:**

* Change the Runtime to GPU in google Colab