HPC Lab Exp No.5 (Mini Project – Huffman Encoding on GPU)

#include <iostream>

#include <vector>

#include <queue>

#include <unordered\_map>

#include <cuda\_runtime.h>

using namespace std;

struct HuffmanNode {

    char character;

    int frequency;

    HuffmanNode\* left;

    HuffmanNode\* right;

    HuffmanNode(char c, int freq) : character(c), frequency(freq), left(nullptr), right(nullptr) {}

    HuffmanNode(int freq, HuffmanNode\* leftNode, HuffmanNode\* rightNode)

        : character(0), frequency(freq), left(leftNode), right(rightNode) {}

    bool operator>(const HuffmanNode& other) const {

        return frequency > other.frequency;

    }

};

\_\_global\_\_ void count\_frequencies\_kernel(const char\* text, int\* freqs, int text\_length) {

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

    if (idx < text\_length) {

        atomicAdd(&freqs[(unsigned char)text[idx]], 1);

    }

}

void count\_frequencies(const char\* text, int\* freqs, int text\_length) {

    char\* d\_text;

    int\* d\_freqs;

    cudaMalloc(&d\_text, text\_length \* sizeof(char));

    cudaMalloc(&d\_freqs, 256 \* sizeof(int));

    cudaMemset(d\_freqs, 0, 256 \* sizeof(int));

    cudaMemcpy(d\_text, text, text\_length \* sizeof(char), cudaMemcpyHostToDevice);

    int block\_size = 256;

    int grid\_size = (text\_length + block\_size - 1) / block\_size;

    count\_frequencies\_kernel<<<grid\_size, block\_size>>>(d\_text, d\_freqs, text\_length);

    cudaDeviceSynchronize();

    cudaMemcpy(freqs, d\_freqs, 256 \* sizeof(int), cudaMemcpyDeviceToHost);

    cudaFree(d\_text);

    cudaFree(d\_freqs);

}

void generate\_huffman\_codes(HuffmanNode\* root, string code, unordered\_map<char, string>& huffman\_codes) {

    if (!root) return;

    if (root->left == nullptr && root->right == nullptr) {

        huffman\_codes[root->character] = code;

    }

    generate\_huffman\_codes(root->left, code + "0", huffman\_codes);

    generate\_huffman\_codes(root->right, code + "1", huffman\_codes);

}

HuffmanNode\* build\_huffman\_tree(const int\* freqs) {

    priority\_queue<HuffmanNode, vector<HuffmanNode>, greater<HuffmanNode>> min\_heap;

    for (int i = 0; i < 256; ++i) {

        if (freqs[i] > 0) {

            min\_heap.push(HuffmanNode(i, freqs[i]));

        }

    }

    while (min\_heap.size() > 1) {

        HuffmanNode\* left = new HuffmanNode(min\_heap.top());

        min\_heap.pop();

        HuffmanNode\* right = new HuffmanNode(min\_heap.top());

        min\_heap.pop();

        int combined\_freq = left->frequency + right->frequency;

        HuffmanNode\* combined = new HuffmanNode(combined\_freq, left, right);

        min\_heap.push(\*combined);

    }

    return new HuffmanNode(min\_heap.top());

}

void huffman\_encoding(const char\* text, int text\_length) {

    int freqs[256] = {0};

    count\_frequencies(text, freqs, text\_length);

    HuffmanNode\* root = build\_huffman\_tree(freqs);

    unordered\_map<char, string> huffman\_codes;

    generate\_huffman\_codes(root, "", huffman\_codes);

    cout << "Huffman Codes:" << endl;

    for (const auto& pair : huffman\_codes) {

        cout << pair.first << ": " << pair.second << endl;

    }

}

int main() {

    const char\* text = "this is a simple example of huffman encoding on gpu";

    int text\_length = strlen(text);

    huffman\_encoding(text, text\_length);

    return 0;

}

OUTPUT:







