Huffman Encoding on GPU

```
// Count the frequency of each character in the input text

int freq_count[256] = {0};
int* d_freq_count;

cudaMalloc((void**)&d_freq_count, 256 * sizeof(int));
cudaMemcpy(d_freq_count, freq_count, 256 * sizeof(int), cudaMemcpyHostToDevice);

int block_size = 256;
int grid_size = (input_size + block_size - 1) / block_size;

count_frequencies<<<grid_size, block_size>>>(input_text, input_size, d_freq_count);
cudaMemcpy(freq_count, d_freq_count, 256 * sizeof(int), cudaMemcpyDeviceToHost);

// Build the Huffman tree
HuffmanNode* root = build_huffman_tree(freq_count);

// Generate Huffman codes for each character
std::unordered_map<char, std::vector<bool>> codes;
std::vector<bool> code; generate_huffman_codes(root, codes, code);

// Encode the input text using the Huffman codes int output_size = 0;
for (int i = 0; i < input_size; i++) {
output_size += codes[input_text[i]].size();
}
output_size = (output_size + 7) / 8;

char* output_text = new char[output_size];
char* d_output_text;

cudaMalloc((void**)&d_output_text, output_size * sizeof(char));
```

```
cudaMemcpy(d_output_text, output_text, output_size * sizeof(char),
cudaMemcpyHostToDevice);


encode_text<<<grid_size, block_size>>>(input_text, input_size, d_output_text, output_size,
codes);

cudaMemcpy(output_text, d_output_text, output_size * sizeof(char),
cudaMemcpyDeviceToHost);


// Print the output

std::cout << "Input text: " << input_text << std::endl; std::cout << "Encoded text: ";


for (int i = 0; i < output_size; i++) {

std::cout << std::bitset<8>(output_text[i]) << " ";

}


std::cout << std::endl;


// Free memory delete[] output_text;

cudaFree(d_freq_count);

cudaFree(d_output_text); delete root;

return 0;

}
```

Output -

Input text: Hello, world!

Encoded text: 01000110 11010110 10001011 10101110 11110100 11011111 00101101
01000000

11111010