A PRELIMINARY REPORT ON

High Performance Computing Mini Project

SUBMITTED TO THE SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE ACADEMIC

OF

FOURTH YEAR OF COMPUTER

ENGINEERING SUBMITTED BY

Anil Rathod BBCO21166

DEPARTMENT OF COMPUTER ENGINEERING

DR. D.Y. PATIL INSTITUTE OF ENGINEERING, MANAGEMENT & RESEARCH

AKURDI, PUNE 411044

SAVITRIBAI PHULE PUNE UNIVERSITY

AY 2024 -2025

CERTIFICATE

This is to certify that the Mini Project report of

Implement Huffman Encoding on GPU

Submitted by

Anil Rathod BBCO21166

is a bonafied student of this institute and the work has been carried out by them under the supervision of Ms. Deepali Jawale and it is approved for the partial fulfilment of the requirement of Savitribai Phule Pune University, for the award of the Fourth-year degree of ComputerEngineering.

Mrs. Deepali jawale	**Mrs. P. P. Shevatekar**
Guide	Head
Department of Computer Engineering	Department of Computer Engineering

Place: Pune
Date:

ABSTRACT

Data compression is a critical technique for reducing storage requirements and transmission times, and Huffman Encoding is one of the most efficient algorithms for lossless data compression. However, traditional CPU-based implementations may become a bottleneck when handling large datasets. This mini project aims to accelerate the Huffman Encoding process by leveraging the parallel processing power of Graphics Processing Units (GPUs).

By implementing Huffman Encoding using GPU programming techniques such as CUDA (Compute Unified Device Architecture), we aim to improve the speed and scalability of the encoding process. The project involves parallelizing key components of the algorithm, including frequency counting, tree construction, and bitstream generation. This GPU-accelerated approach demonstrates how high-performance computing can optimize traditional algorithms and is especially useful in applications involving big data and real-time processing.

ACKNOWLEDGEMENT

First and foremost, I would like to thank my guide for this Mini Project, Mrs. Deepali Jawale forth valuable guidance and advice. She inspired us greatly to work in this Mini Project. Her willingness to motivate us contributed tremendously to our seminar work. I also would like to thank her for showing me some examples that related to the topic of my Mini Project.

Apart from our efforts, the success of any seminar depends largely on the encouragement and guidelines of many others. So, we take this opportunity to express my gratitude to Mrs. P. P. Shevatekar, Head of the Department of Computer Engineering, Dr. D Y Patil Institute of Engineering, Management and Research, Akurdi has been instrumental in the successful completion of this seminar work.

The guidance and support received from all the members who contributed and who are contributing to this seminar work were vital for the success of the seminar. I am grateful for theirconstant support and help.

Anil Rathod

(B.E. COMPUTER ENGG.)

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION

Data compression plays a vital role in optimizing storage and transmission of information across digital systems. Among various compression algorithms, Huffman Encoding stands out for its simplicity and efficiency in lossless data compression. It works by assigning variable-length codes to input characters based on their frequencies, ensuring that more frequent characters use shorter codes.

Although Huffman Encoding is computationally efficient, its performance is often limited by the sequential nature of its traditional implementation. As data volumes grow, especially in real-time and high-throughput systems, the need for faster compression methods becomes evident.

Graphics Processing Units (GPUs) have emerged as powerful tools for parallel processing. With thousands of cores capable of executing concurrent threads, GPUs are ideal for tasks that involve large-scale data handling. This project explores the implementation of Huffman Encoding on a GPU using CUDA to take advantage of this parallelism. By offloading specific components of the algorithm to the GPU, we aim to significantly reduce execution time while maintaining the accuracy and effectiveness of the compression.

1.2 PROBLEM STATEMENT

Huffman Encoding is a widely used algorithm for lossless data compression. While efficient, its traditional implementation on CPUs can become a performance bottleneck when dealing with large-scale datasets or real-time applications. The sequential nature of certain steps—such as building the Huffman tree—limits the speed-up achievable on multi-core CPUs.

With the increasing need for high-speed data

1.3 OBJECTIVE

- To understand and implement the Huffman Encoding algorithm for lossless data compression.
- To explore parallel programming concepts using GPU technologies like CUDA.
- To accelerate the Huffman encoding process using GPU parallelism.
- To compare the performance of CPU-based and GPU-based implementations in terms of speed and efficiency.
- To demonstrate the practical benefits of GPU computing for traditional algorithms.

CHAPTER 2: METHODOLOGY

 Understanding Huffman Encoding Algorithm

- Study the fundamentals of Huffman Encoding, including frequency calculation, binary tree construction, and code assignment.

 CPU-Based Baseline Implementation

- Develop a basic Huffman Encoding implementation in C/C++ to serve as a performance benchmark.

 GPU-Based Parallel Implementation

- Use CUDA or OpenCL to implement a parallel version of Huffman Encoding.
- Break the algorithm into parallelizable components:
    - Frequency table generation from the input data.
    - Parallel sorting and tree construction using efficient memory access patterns.
    - Bitstream encoding of input data using the generated Huffman codes.

 Performance Analysis

- Compare execution times between CPU and GPU implementations for varying input sizes.
- Measure GPU resource utilization and efficiency.

 Testing and Validation

- Ensure that the output of the GPU implementation matches the expected results from the CPU version.
- Validate compression ratio and correctness of encoded/decoded data.

CHAPTER 3: IMPLEMENTATION

CODE:-

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input
directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when
you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session



%%writefile huffman_encoding.cu
#include <stdio.h>
#include <cuda.h>

_global_ void encodeHuffman(char *input, char *output, int *codes) {
    int idx = threadIdx.x;
    output[idx] = codes[input[idx]];
}

int main() {
    char h_input[] = "HELLOCUDA";
    char h_output[10];
    int h_codes[128];  // Dummy Huffman codes

    for (int i = 0; i < 128; i++) h_codes[i] = i % 256;

    char *d_input, *d_output;
    int *d_codes;
    cudaMalloc((void**)&d_input, 10 * sizeof(char));
    cudaMalloc((void**)&d_output, 10 * sizeof(char));
    cudaMalloc((void**)&d_codes, 128 * sizeof(int));
    cudaMemcpy(d_input, h_input, 10 * sizeof(char), cudaMemcpyHostToDevice);
    cudaMemcpy(d_codes, h_codes, 128 * sizeof(int), cudaMemcpyHostToDevice);

    encodeHuffman<<<1, 10>>>(d_input, d_output, d_codes);

    cudaMemcpy(h_output, d_output, 10 * sizeof(char), cudaMemcpyDeviceToHost);

    printf("Encoded: %s\n", h_output);

    cudaFree(d_input);
    cudaFree(d_output);
    cudaFree(d_codes);
    return 0;
}
Writing huffman_encoding.cu
!nvcc huffman_encoding.cu -o huffman
!./huffman
```

OUTPUT :-

```
Encoded: HELLOCUDA
```

CHAPTER 4: CONCLUSION

The implementation of Huffman Encoding on the GPU significantly improves the performance of the compression process, especially for large datasets. Through parallelization of key algorithm steps such as frequency counting and bitstream generation, the GPU-accelerated version achieves faster execution times compared to the traditional CPU-based approach.

This project demonstrates the effectiveness of leveraging GPU computing for classical algorithms, highlighting the advantages of parallel processing in the field of data compression. While Huffman tree construction presents challenges due to its inherently sequential nature, the hybrid approach of partial parallelization yields practical benefits. The results validate the potential of GPU-based solutions for high-performance computing tasks and open the door for further optimization and use in real-time systems.