A MINI PROJECT REPORT ON

PARALLEL HUFFMAN ENCODING USING OPENMP

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

Fourth Year Computer Engineering Semester I, Academic Year 2021-2022

OF

Savitribai Phule Pune University.

SUBMITTED BY

RUTUJA KAWADE	**Roll. No.: 41233**
ROHITH KANGULDA	**Roll. No.: 41230**
FATEMA KATAWALA	**Roll. No.: 41232**

DEPARTMENT OF COMPUTER ENGINEERING

PUNE INSTITUTE OF COMPUTER TECHNOLOGY

DHANKAWADI, PUNE – 43
SAVITRIBAI PHULE PUNE UNIVERSITY

2021 -2022

CERTIFICATE

This is to certify that the project report

"PARALLEL HUFFMAN ENCODING USING OPENMP"

Submitted by

RUTUJA KAWADE	**Roll. No.: 41233**
ROHITH KANGULDA	**Roll. No.: 41230**
FATEMA KATAWALA	**Roll. No.: 41232**

have satisfactorily completed a mini project for Lab Practices I: 'High Performance Computing' subject under the guidance of Prof. R. A. Kulkarni in partial fulfillment of Fourth Year Computer Engineering Semester I, Academic Year 2020-2021 of Savitribai Phule Pune University.

(Prof. Tushar. S. Pinian)
Internal Guide
Department of Computer Engineering

(**Prof. Mrs. M. S. Takalikar)**
Head,
Department of Computer Engineering

Place: Pune

Date:

TABLE OF CONTENTS

1. INTRODUCTION

A **Huffman code** is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding or using such a code proceeds by means of **Huffman coding**, an algorithm developed by David A. Huffman while he was a Sc.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes".

The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file). The algorithm derives this table from the estimated probability or frequency of occurrence (*weight*) for each possible value of the source symbol. As in other entropy encoding methods, more common symbols are generally represented using fewer bits than less common symbols. Huffman's method can be efficiently implemented, finding a code in time linear to the number of input weights if these weights are sorted. However, although optimal among methods encoding symbols separately, Huffman coding is not always optimal among all compression methods - it is replaced with arithmetic coding or asymmetric numeral systems if better compression ratio is required.

It is a lossless data compression algorithm.

The application programming interface (API) **OpenMP** (**Open Multi-Processing**) supports multi-platform shared-memory multiprocessing programming in C, C++, and Fortran, on many platforms, instruction-set architectures and operating systems, including Solaris, AIX, HP-UX, Linux, macOS, and Windows. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behaviour

OpenMP uses a portable, scalable model that gives programmers a simple and flexible interface for developing parallel applications for platforms ranging from the standard desktop computer to the supercomputer.

2. HUFFMAN ENCODING: WORKING

In this algorithm a variable-length code is assigned to input different characters. The code length is related with how frequently characters are used. Most frequent characters have smallest codes, and longer codes for least frequent characters.

There are mainly two parts. First one to create Huffman tree, and another one to traverse the tree to find codes.

For an example, consider some strings “YYYZXXYYX”, the frequency of character Y is larger than X and the character Z has least frequency. So, the length of code for Y is smaller than X, and code for X will be smaller than Z.

Complexity for assigning code for each character according to their frequency is O(n log n)

Input – A string with different characters, say “ACCEBFFFFAAXXBLKE”
Output – Code for different characters:

```
Data: K, Frequency: 1, Code: 0000
Data: L, Frequency: 1, Code: 0001
Data: E, Frequency: 2, Code: 001
Data: F, Frequency: 4, Code: 01
Data: B, Frequency: 2, Code: 100
Data: C, Frequency: 2, Code: 101
Data: X, Frequency: 2, Code: 110
Data: A, Frequency: 3, Code: 111
```

3. HUFFMAN ENCODING: ALGORITHM

Input – A string with different characters.

Output – The codes for each individual characters.

```
Begin
  define a node with character, frequency, left and right child of the node for
Huffman tree.
  create a list 'freq' to store frequency of each character, initially all are 0
  for each character c in the string do
    increase the frequency for character ch in freq list.
  done
  for all type of character ch do
    if the frequency of ch is non zero then add ch and its frequency as a node of
priority queue Q.
  done
  while Q is not empty do
    remove item from Q and assign it to left child of node
    remove item from Q and assign to the right child of node
    traverse the node to find the assigned code
  done
End
```

traverseNode(n: node, code)

Input – The node n of Huffman tree, and code assigned from previous call

Output – Code assigned with each character

```
if left child of node n ≠ ϕ then
  traverseNode(leftChild(n), code+'0') //traverse through the left child
  traverseNode(rightChild(n), code+'1') //traverse through the right child
else
  display the character and data of current node.
```

4. NEED OF PARALLELIZATION

Parallelization is the use of multiple processing elements simultaneously for solving any problem. Problems are broken down into instructions and are solved concurrently as each resource which has been applied to work is working at the same time.

It saves time and money as many resources working together will reduce the time and cut potential costs. It can be impractical to solve larger problems on Serial Computing. It can take advantage of non-local resources when the local resources are finite. Serial Computing ‘wastes’ the potential computing power, thus Parallel Computing makes better work of hardware.

5. ADVANTAGES OF PARALLEL COMPUTING

- The whole real world runs in dynamic nature i.e. many things happen at a certain time but at different places concurrently. This data is extensively huge to manage.
- Real world data needs more dynamic simulation and modelling, and for achieving the same, parallel computing is the key.
- Parallel computing provides concurrency and saves time and money.
- Complex, large datasets, and their management can be organized only and only using parallel computing's approach.
- Ensures the effective utilization of the resources. The hardware is guaranteed to be used effectively whereas in serial computation only some part of hardware was used and the rest rendered idle.
- Also, it is impractical to implement real-time systems using serial computing.

6. LIMITATIONS OF PARALLEL COMPUTING

- It addresses such as communication and synchronization between multiple sub-tasks and processes which is difficult to achieve.
- The algorithms must be managed in such a way that they can be handled in the parallel mechanism.
- The algorithms or program must have low coupling and high cohesion. But it’s difficult to create such programs.
- More technically skilled and expert programmers can code a parallelism based program well.

7. PARALLEL HUFFMAN ENCODING

Library Used: OpenMP

Data: Characters are read from a text file

Sample Input: `aaabbbbbbbbbeeeeeee`

Sample Output:

e : 7
b : 8
a : 3

The content is b and its corresponding code is 0
The content is a and its corresponding code is 10
The content is e and its corresponding code is 11
Parallel Time: 0.00029872 seconds

Time Analysis:

NUMBER OF CHARACTERS	SERIAL TIME (seconds)	PARALLEL TIME (seconds)
3	0.0002	0.000292956
5	0.000252	0.000328582
8	0.000335	0.000399921
10	0.00054	0.000502613
22	0.000688	0.000849825

For n=10,

Speedup=1.08

8. TESTCASES

INPUT	ACTUAL OUTPUT	EXPECTED OUTPUT	RESULT
1. aaabbcc	a:0 c:10 b:11	a:0 c:10 b:11	**Pass**
2. aaaabbbbbbbbccccccccddddd	a:00 d:01 c:10 b:11	a:00 d:01 c:10 b:11	**Pass**
3. fffffffffffannnccdddwwwwqqqr	f:0 n:100 w:101 c:1100 r:11010 a:11011 q:1110 d: 1111	f:0 n:100 w:101 c:1100 r:11010 a:11011 q:1110 d: 1111	**Pass**

9. CONCLUSION

Hence, we have successfully parallelized the Huffman Encoding technique using OpenMP. The optimal speedup of 1.08 was recorded for 10 characters.