CHAPTER III
A PARALLEL HUFFMAN DECODING ALGORITHM

3.1. Introduction

One of the efficient techniques to determine an optimal set of codes for the transmission of \( n \) messages is Huffman code \([1]\). Huffman decoding is still one of the popular compression techniques and is widely used in different fields of science. The field of data compression constantly attracts the interest of many researchers both in theoretical foundations of computing and in application areas. In the last two decades, the fundamentals of data compression have been laid \([2,3,4]\) and efficiently applied to text and image compression. Currently, data compression is of increasing interest again because of the growing amount of data processed in applications and transferred over the internet. In particular, compression of geometric data is currently an active research area \([13]\), and it is also important to perform operations on the compressed data directly working in the compressed domain, instead of decompressing prior to any processing.

Ong and Huang \([17]\) developed a data compression scheme for Chinese text files. They adopted Huffman coding method, due to the skewness of the distribution of Chinese ideograms. By storing the frequencies of the encoding symbols rather than their Huffman codes in a dictionary, applying differential coding where it saves space, and structuring the dictionary in the Huffman coding scheme into a two-level dictionary structure, their algorithm produces significant improvement on the compression results. However, fast decoding and scanning
through compressed data are more important than code construction. This motivates us to design a parallel algorithm for Huffman decoding. Huffman decoding is subjected to numerous investigations in the past 52 years. Huffman coding creates minimal redundancy codes for a given set of symbols and their respective occurrence frequencies. Each code, in this technique, is a binary string, encoded form of the original message, which is decoded into actual message at the receiving end, again, using a decode tree. A decode tree is a binary tree in which external nodes represent messages. This chapter employs a parallel algorithm for Huffman decoding for CREW PRAM model, which is a Concurrent Read and Exclusive Write, Parallel Random Access Memory consisting of N processors.

3.2. Data Structure

Consider a Huffman tree T with n symbols (that represent messages). The symbols corresponding to leaves are labeled as \(m_1, m_2, \ldots, m_n\) from left to right. The root is said to be at level 0. The level of any other node is 1 more than the level of its father. The level of the Huffman tree is denoted by \(p\). The largest level is the depth \(d\) of the Huffman tree. The weight of a symbol is defined to be \(2^d\). Let \(W_i\) be the weight of the symbol \(m_i\), for \(i = 1, 2, \ldots, n\). Let us define the variable \(\text{cum}_1 = W_1\) and \(\text{cum}_i = \text{cum}_{i-1} + W_i\) for \(i = 2, 3, \ldots, n\). The Huffman tree with 10 leaves is shown in Figure 3.1 and the corresponding values of \(m_i, W_i\) and \(\text{cum}_i\) are shown in Table 3.1.
Figure 3.1. **Huffman tree with 10 leaves denoting 10 messages**
Table 3.1. The values of $m_i$, $W_i$, $cum_i$. 

<table>
<thead>
<tr>
<th>Index</th>
<th>$m_i$</th>
<th>$W_i$</th>
<th>$cum_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$m_1$</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>$m_2$</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>$m_3$</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>$m_4$</td>
<td>8</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>$m_5$</td>
<td>4</td>
<td>28</td>
</tr>
<tr>
<td>6</td>
<td>$m_6$</td>
<td>1</td>
<td>29</td>
</tr>
<tr>
<td>7</td>
<td>$m_7$</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>$m_8$</td>
<td>2</td>
<td>.32</td>
</tr>
<tr>
<td>9</td>
<td>$m_9$</td>
<td>16</td>
<td>48</td>
</tr>
<tr>
<td>10</td>
<td>$m_{10}$</td>
<td>16</td>
<td>64</td>
</tr>
</tbody>
</table>
3.3. Description of Algorithm

First, we compute value \( t \) using a formula \( t = (c + 1) \times 2^{d-b} \) from the codeword \( c \) given. Then \( t \) value is searched in ‘cum’ array and when a match is found (let the index be \( k \)), we check if \( w_k \) equals \( 2^{d-b} \). If it is so, the corresponding message \( m_k \) is the message for the codeword \( c \). Now, notice that ‘cum’ is the cumulative function of ‘\( W \)’ and it is strictly increasing function, since \( W_i \) can’t be less than 1. This implies that the elements of array ‘cum’ are distinct and in non-decreasing order. This is the requisite for using (\( N+1 \))-ary search (Parallel version of Binary Search), where \( N \) is the number of processors. Hence, in our parallel Huffman decoding algorithm, we employ the aforementioned search procedure in which, at each stage, the ‘cum’ array splits into \( N+1 \) subsequences of equal length and the \( N \) processors simultaneously search the elements at the boundary between successive subsequences. Once it is searched, we check the corresponding weight, and based on the flag, we get the message for the corresponding code word.

3.4. Our Parallel Algorithm

In this section, we present a Huffman decoding parallel algorithm using CREW PRAM model with \( N \) processors.
**ALGORITHM**

**Input:** The values of $r_{ii}$, $W_i$, $cum_i$ where $i = 1, 2, \ldots, n$ of a Huffman tree $T$ which contains symbols with depth $d$ and a binary codeword $c$.

**Output:** The corresponding symbol $m_k$ of $c$.

**METHOD:**

**Step 1**: compute $t \leftarrow (c+1) \times 2^{d-b}$ where $b$ is the number of binary digits in $c$.

**Step 2**: Search $t$ in the array ‘cum’

2.1) $q \leftarrow 1$

2.2) $r \leftarrow n$

2.3) $k \leftarrow 0$

2.4) $\text{stages} \leftarrow \left\lceil \frac{\log(n+1)}{\log(N+1)} \right\rceil$

2.5) while ($q \leq r$ and $k = 0$) do

\[ j_o \leftarrow q - 1 \]

For $i = 1$ to $N$ do in parallel

\[ x \leftarrow j_i \leftarrow (q-1) + i \times (N+1)^{\text{stages}-1} \]

{Processor $P_i$ compares $t$ to $cum_i$ and decides on the part to be retained}

if $j_i \leq r$

then if $cum_x = t$

then $k \leftarrow j_i$

else

if $cum_x > t$

68
then flag_i ← left

else

   flage_i ← right

endif

endif

else

   j_i = r + 1

   flag_i ← left

endif

{This indices of the subsequence to be searched in the next iteration are computed}

if flag_i ≠ flag_{i+1}

   then q ← j_{i-1} + 1

   r ← j_i - 1

endif

if ( i = N and flag_i ≠ flag_{i+1})

   then q ← j_i + 1

endif

endfor

stages ← stages -1

endwhile

Step 3: if k ≠ 0 and W_k = 2^{d-b} then
3.5. Time Complexity Analysis

Theorem 3.5.1:

The proposed parallel algorithm correctly finds the symbol for a given code on CREW PRAM model with $O(\log_{N+1}(n+1))$-time using $N$ processors.

Proof:

In step-1, the value of $t$ can be calculated in constant time. In step-3, we check the value of $k$, weight ($W_1$) and based on the flag, we find the message corresponding to the symbol. So, this step also takes constant time. In step-2, we employed $(N+1)$-ary search, which is the parallel version of binary search and in the worst case, the number of stages needed is $O(\log_{N+1}(n+1))$. Hence the time complexity of our Huffman decoding algorithm amounts to $O(\log_{N+1}(n+1))$, where $1 < N \leq n$. Hence the proof.

3.6. Illustration

We illustrate our algorithm with the following examples.

3.6.1. Example 1:

Let $c = 01011$ and $N$ be 2.

In step-1, $x = (11+1) \times 2^{6-5}$

$= 24$
In step-2, the sequence \( \text{cum}_i \) is split into \((8,12,16), (24,28,29), (30,32,48,64)\). and in the next iteration, the subsequence \((24,28,29)\) is split into \((24), (28), (29)\). Step-2 terminates on giving \( \text{cum}_4 = 24 \) (\( k \) being 4).

In step-3, \( W_4 (=8) \leftrightarrow 2^{6-5} \). Thus 01011 is not a codeword of \( T \).

### 3.6.2. Example 2:

Let \( c = 0110 \) and \( N \) be 2.

In step-1, \( x = (6+1)*2^{6-4} = 28 \).

In step-2, the sequence \( \text{cum}_i \) is split into \((8,12,16), (24,28,29), (30,32,48,64)\). In the next iteration of step-2, the subsequence \((24,28,29)\) is split into \((24), (28), (29)\). Step-2 terminates on giving \( \text{cum}_5 = 28 \) (\( k \) being 5).

In step-3, \( W_5 (=4) = 2^{6-4} \). Thus, 0110 is a codeword of \( T \) corresponding to message \( m_5 \).

### 3.7. Discussion

It is pertinent to point out here that the development of an exact or more efficient parallel algorithm for Huffman decoding is deemed of importance from the view of wide practical applications of the Huffman decoding as a very efficient technique for compressing data. Keeping this in view, our attention has been focused to present a parallel algorithm for Huffman Decoding using CREW PRAM model with time complexity \( O (\log_{N+1}(n+1)) \), where \( 1 < N \leq n \). When \( N = n \), the algorithm runs in constant time, since the elements of 'cum' array are distinct (which is the required condition for achieving this constant time).
In general, the sparsity in the Huffman tree causes a huge waste of memory space for array implementation. A thorough analysis reveals that there is an important issue in designing of parallel algorithms in order to reduce the memory size, which will in turn speed up the process. In view of this, we intended to design a memory-efficient parallel algorithm, which is discussed in the next chapter.