A Work-Efficient Algorithm for Constructing Huffman Codes

Marek Karpinski* Lawrence Larmore† Yakov Nekrich‡

Abstract

We present an algorithm for parallel construction of Huffman codes in $O(\frac{n}{\sqrt{p}} \log p)$ time with p processors, improving the previous result of Levcopoulos and Przytycka.

Keywords: Computational Complexity, Parallel Algorithms, Huffman Codes

*Dept. of Computer Science, University of Bonn. Work partially supported by a DFG grant , DIMACS, and IST grant 14036 (RAND-APX). Email marek@cs.uni-bonn.de.

†School of Computer Science, University of Nevada Las Vegas. Email larmore@earl.cs.unlv.edu

‡Dept. of Computer Science, University of Bonn. Work partially supported by IST grant 14036 (RAND-APX). Email yasha@cs.uni-bonn.de.

1 Introduction

A Huffman code for an alphabet $a_1, a_2, \ldots, a_n$ with weights $w_1, w_2, \ldots, w_n$ is a prefix code that minimizes weighted codeword length, defined as $\sum_{i=1}^{n} w_i l_i$, where l_i is the length of the i^{th} codeword.

The best known parallel algorithm [LP95] for this problem that uses n processors is due to Larmore and Przytycka and runs in $O(\sqrt{n}\log n)$ time. Their algorithm is based on reducing the problem of constructing a Huffman code to the concave least weight subsequence problem. Levcopoulos and Przytycka [LevPrz95] have presented an algorithm for the efficient construction of Huffman trees with the sublinear number of processors. Their algorithm runs in $O(\frac{n}{\sqrt{p\log(p+1)}}(\log^2 p + \log(n - \sqrt{p\log p})))$ with p processors. However, we observe that the algorithm of [LevPrz95] contains a technical flaw since construction of left-justified trees is not combined with the greedy Huffman algorithm in appropriate way.

In this paper we further improve the result of Levcopoulos and Przytycka and describe an algorithm that works in $O(\frac{n}{\sqrt{p}}\log p)$ time with p processors. Our algorithm uses a combination of ideas inspired by several prior approaches to the problem. In particular, we combine the methods used for the construction of almost-optimal Huffman codes (see [KP96] ,[BKN02]), the greedy paradigm of Huffman's algorithm [H51], and the CLWS approach from [LP95].

2 Algorithm Description

The technique for the construction of approximate Huffman codes in parallel given in [KP96, BKN02] is to divide elements a_i into classes W_j, $j = m, \ldots, 1$ such that $W_j = \{w_i | 1/2^j \leq w_i < 1/2^{j-1}\}$. This ensures that the sum of the weights of any two elements of a given class is greater than the weight of any other element of that same class. Basically, Huffman's algorithm can be reformulated as follows. Let W_k be the class which contains the smallest element, and initialize $\tilde{W}_k$ to be empty. For each j, in descending order, consecutive pairs of elements of the merged list $\tilde{W}_j + W_j$ are combined and stored as a list $\tilde{W}_{j-1}$, then $\tilde{W}_{j-1}$ and W_{j-1} are merged. It follows that elements from the same class can be processed in parallel. See [KP96] for a detailed description of this algorithm that also considers the case of an odd number of elements in $\tilde{W}_j + W_j$. A limitation of this approach is that the number of parallel steps is proportional to the number of different classes W_j.

In this paper we will show how we can reduce the number of steps for the classes of small size. A binary tree T is called *left-justified* if the height of every leaf of the left subtree of T is greater than or equal to the height of the right subtree of T, and if every proper subtree of T is left-justified. The following theorem was proven in [LP95].

Theorem 1 *A left-justified Huffman tree T for elements $a_1, a_2, \ldots, a_p$ can be constructed in $O(\sqrt{p}\log p)$ time with p processors.*

In general, a tree constructed by Huffman's original algorithm is not left-justified. We call a tree constructed by Huffman's algorithm a *greedy* Huffman tree.

To combine the algorithm of [LP95] with the approach of [KP96] and [BKN02] we need the following theorem

Theorem 2 *A greedy Huffman tree T for elements $a_1, a_2, \ldots, a_p$ can be constructed in $O(\sqrt{p}\log p)$ time with p processors.*

We begin by describing Huffman's algorithm by formulating of a lemma.

Let $b_1, \ldots b_{n-1}$ be the internal nodes of the greedy Huffman tree, and let v_i be the weight of b_i, defined to be the sum of the weights of the leaves of the subtree rooted at b_i. We index the internal nodes in the order in which Huffman's algorithm produces them; thus $v_1 \leq v_2 \leq \ldots v_{n-1}$.

Let F_r be the greedy Huffman forest of r roots, *i.e.*, the forest obtained after $n-r$ steps of Huffman's algorithm. Note that F_n is the forest consisting of n singleton nodes, while F_1 is greedy Huffman tree. The leaves of each F_r are the original items. Huffman's algorithm constructs F_{n-j} by combining the two least weight trees of F_{n-j+1} into a single tree, creating the new root b_j.

We will use the following tie-breaking rule. We assign a *t-value* to every node in a Huffman tree. The t-value of the leaf a_i will be 2^i, and the t-value of an internal node will be the sum of t-values of its children. Thus all nodes have distinct t-values. In case two nodes have the same weight, the one with the smaller t-value will be taken to be the smaller one.

This method reduces the problem with ties to the problem with no ties, by simply changing the weight of the a_i to $w_i + \epsilon 2^i$, where ϵ is some very small positive constant.

This tie-breaking scheme can be implemented at a cost of $O(1)$ per comparison, as follows. For each node, keep track of the index of the largest leaf in the subtree rooted at that node, and call that the *dominant* t-value of that node. If it becomes necessary to compare the t-values of two nodes, simply compare their dominant t-values. Since the subtrees are disjoint, they will have distinct dominant t-values.

Lemma 1

1. *F_n consists of the singleton trees $\{a_i\}$.*

2. *For $j > 1$, one of the roots of F_{n-j} is b_j. Furthermore, $F_{n-j} - b_j = F_{n-j+1}$, and the children of b_j are the roots of the two smallest weight trees in F_{n-j+1}.*

3. *If $k < \ell < j$ and b_k is a root of F_{n-j}, then b_ℓ is a root of F_{n-j}.*

4. *F_{n-j} has leaves $a_1, \ldots a_n$, internal nodes $b_1, \ldots b_j$, and roots $b_{i+1}, \ldots b_j$, $a_{2j-i+1}, \ldots a_n$, for some $i < j$.*

5. *Let B_{n-j} be the set of children of non-leaf roots of F_{n-j}. Then the sum of the weights of any two elements of B_{n-j} is greater than the weight of any other element of B_{n-j}.*

Proof:

Part 1 and Part 2 are the well-known observations which justify Huffman's algorithm. Part 3 is proved by contradiction, as follows. If $k < \ell < j$, and b_k is a root of F_{n-j} and b_ℓ is not, then F_{n-j} could be improved by exchanging b_k and b_ℓ, since $v_k < v_\ell$. Part 4 follows from Part 1, Part 2, Part 3, and a simple computation. Part 5 is proved by contradiction in the same manner as Part 3; if it were false, the smallest non-leaf root of F_{n-j} could be exchanged with the largest member of B_{n-j}, improving F_{n-j}. $\square$

We now summarize the reduction given in [LP95]. For $0 \le m < n$, let $S_m = \sum_{i=1}^m w_i$, the sum of the weights of the first m symbols. Let G be the weighted directed acyclic graph whose nodes are the integers $0, 1, \ldots n-1$, and whose edges are the pairs (i, j) such that $i < j$ and $2j - i \le n$, where the edge (i, j) has weight S_{2j-i}. Define $\mathit{back}_0 = 0$, and for any $0 < j < n$, let back_j be defined to be the next-to-the-last node in the minimum weight path from 0 to j. Let f_j be the total weight of that minimum weight path. More formally, $f_0 = 0$ and, for $j > 0$, $f_j = \min_{i<j} \{f_i + S_{2j-i} : i < j\}$, and back_j is that choice of i for which the minimum value of f_j is achieved.

We will use the following lemmas from [LP95].

Lemma 2 *The graph G has the concave Monge property.*

Lemma 3 *We can find minimum weight paths from 0 to j for $1 \le j \le n$ in $O(\sqrt{n})$ time with n processors.*

Algorithm CLWS described in the paper [LP] finds minimum weight paths for all j. According to theorem 2.7 of [LP] it works in $O(n \log n/m + n^2/mp + nm \log n/p)$ time with p processors for any $0 < m \leq n$. By setting $p = n$ and $m = \sqrt{n}$ we get the result of the Lemma.

Lemma 4

1. *The weighted path length of F_{n-j} is equal to f_j.*
2. *For $0 < j < n$, $f_j - f_{j-1} = v_j$.*
3. *If $back_j = i$, then the roots of F_{n-j} are $b_{i+1}, \ldots b_j, a_{2j-i+1}, \ldots a_n$.*
4. *For $0 < j < n$, $back_{j-1} \leq back_j \leq back_{j-1} + 2$. Let $i = back_{j-1}$. Then*
   (a) *If $back_j = i$ then the two children of b_j are a_{2j-i-1} and a_{2j-i}.*
   (b) *If $back_j = i + 1$ then the two children of b_j are b_{i+1} and a_{2j-i-1}.*
   (c) *If $back_j = i + 2$ then the two children of b_j are b_{i+1} and b_{i+2}.*

Proof:

We first prove Part 1 by strong induction. For $j = 0$ it is trivial, so assume $j > 0$. Part 3 of Lemma 1 allows us to choose $i < j$ such that b_k is a root of F_{n-j} if and only if $i < k \leq j$. F_{n-i} is then obtained from F_{n-j} by deleting the roots b_k for $i < k \leq j$. Since F_{n-j} must have $n - j$ roots, of which $j - i$ are the b_k, it must have $n - 2j + i$ roots which are original items. It follows from Part 5 of Lemma 1 that removal of b_k for $i < k \leq j$ causes all leaves which are not roots of F_{n-i} to move up one level in the tree. Hence the weighted path length is decreased by the sum of the weights of those leaves, which is S_{2j-i}, which is also the weight of the edge (i, j) in G. By the inductive hypothesis, the weighted path length of F_i is f_i, thus the weighted path length of F_j is $f_i + S_{2j-i} \geq f_j$.

To prove that f_j is also an upper bound for the weighted path length of F_j, let $i = back_j$. We consider two cases. If $i = 0$, then $f_j = S_{2j-i}$, which is the weighted path length of the forest obtained by combining $a_1, \ldots a_{2j}$ in pairs, resulting in a forest with $n - j$ roots and weighted path length f_j. Now suppose $i > 0$. By the inductive hypothesis, the weighted path length of F_i is f_i. Let $m = back_i$. We know that $2i - m \leq 2j - i$, since otherwise a smaller weight path from 0 to j could be obtained by replacing the edges (m, i) and (i, j) by the edges $(m, i - 1)$ and $(i - 1, j)$. The forest obtained from F_i by combining the roots $b_m + 1, \ldots b_i, a_{2i-m+1}, \ldots a_{2j-i}$ in pairs then has $n - j$ roots and weighted path length f_j.

Part 2 follows from Part 1 and Part 2 of Lemma 1, since the difference between the weighted path lengths of F_{n-j} and F_{n-j+1} is the weight of the subtree rooted at b_j.

Part 3 follows from the above discussion and from Part 4 of Lemma 1.

We now prove Part 4. The two children of b_j are the two roots of F_{j-1} of smallest weight, which must be in the set $\{b_{i+1}, b_{i+2}, a_{2j-i-1}, a_{2j-i}\}$. There are three possibilities, giving us the three cases.

This completes the proof of Lemma 4. $\square$

Theorem 2 follows from Lemma 3 and Part 4 of Lemma 4.

Now we finish the description of our algorithm. Suppose that classes $W_m, W_{m-1}, \ldots, W_{i+1}$ are already processed. If the number of elements in W_i exceeds p, we construct $\tilde{W}_{i-1}$ as described above. If $|W_i| < p$ then classes $W_i, W_{i-1}, \ldots, W_l$, such that $|W_i|+|W_{i-1}|+\ldots+|W_l| \leq p$ and $|W_i|+|W_{i-1}|+\ldots+|W_l| > p$ are considered. Using theorem 2 we can construct a greedy Huffman tree T' for elements of classes $W_i, W_{i-1}, \ldots, W_l$.

We consider the sets N of all nodes S in T', such that the weight of at least one son of S is in $[2^{l-2}, 2^{l-1})$. It is easy to see that the set N can be used instead of the set $\tilde{W}_{l-1}$ from the previous algorithm.

There are at most n/p large classes with more than p elements. Therefore there are at most n/p groups of small classes. The total time to process all groups of small classes can therefore be limited by $O((n/\sqrt{p}) \log n)$. The time necessary to process a large class is $O(\log\log p \cdot |W_i|/p)$ Hence the total time for processing all large classes is less than $O(n/p \log\log p)$.

Therefore the running time of the modified algorithm is $O((n/\sqrt{p}) \log p)$.

3 Open Problems

It is an open problem whether there exists an $O((n/\sqrt{p}))$ p-processor algorithm for Huffman coding. Such an algorithm can perhaps be constructed with a faster CLWS algorithm for a Monge graph with limited edge lengths.

Another open problem is the construction of faster parallel algorithms based on the CLWS approach, that would use the special properties of Monge graphs corresponding to Huffman codes (for instance, the fact that there are only a linear number of distinct weights on the edges.)

References

[BKN02] Berman, P., Karpinski, M., Nekrich, Y., *Approximating Huffman Codes in Parallel*, Proc. 29th ICALP, LNCS vol. 2380, Springer, 2002, pp. 845–855.

[H51] Huffman, D. A., *A method for construction of minimum redundancy codes*, Proc. IRE,40 (1951), pp. 1098–1101.

[KP96] Kirkpatrick, D., Przytycka, T., *Parallel Construction of Binary Trees with Near Optimal Weighted Path Length*, Algorithmica **15**(2) (1996), pp. 172–192.

[LP95] Larmore, L., Przytycka, T., *Constructing Huffman trees in parallel*, SIAM Journal on Computing **24**(6) (1995), pp. 1163–1169.

[LevPrz95] Levcopoulos, Ch., Przytycka, T. *A work-time trade-off in parallel computation of Huffman trees and concave least weight subsequence problem* , Parallel Processing Letters, **4**(1-2) (1994), pp. 37-43

[vL] van Leeuwen, J., *On the Construction of Huffman Trees*, Proc. 3rd ICALP, Edinburgh University Press, 1976, pp. 382–410.