Krzysztof RYBAK [1], **Ernest JAMRO** [2], Kazimierz WIATR [2]
[1] ACK CYFRONET AGH , ul. Nawojki 11, 30-950 Kraków
[2] AGH - AKADEMIA GÓRNICZO-HUTNICZA, KATEDRA ELEKTRONIKI, Al. Mickiewicza 30, 30-059 Kraków

Realizacja kompresji danych metodą Huffmana z ograniczeniem długości słów kodowych

Inż. Krzysztof RYBAK

Ukończył studia na AGH (2012), Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki na kierunku Elektronika i Telekomunikacja. Obecnie student I roku studiów II stopnia na tym samym kierunku.. Jego zainteresowania naukowe dotyczą kompresji danych.

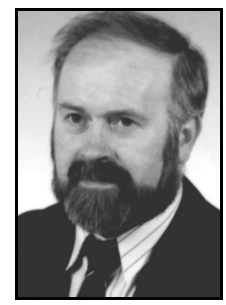
e-mail: gerlin@student.agh.edu.pl

Dr inż. Ernest JAMRO

Ukończył studia na AGH na kierunku Elektronika oraz na University of Huddersfield (UK) na kierunku Elektronika i Telekomunikacja. Obronił pracę doktorską w 2001 roku na AGH na wydziale Elektrotechniki, Automatyki, Informatyki i Elektroniki. Aktualnie jest adiunktem w Katedrze Elektroniki na AGH. Jego zainteresowania naukowe to sprzętowa akceleracja obliczeń, niskopoziomowe przetwarzanie obrazów, sieci neuronowe.

e-mail: jamro@agh.edu.pl

Prof. dr hab. inż. Kazimierz WIATR



Studia AGH Kraków (1980), dr nauk technicznych (1987), dr habilitowany (1999) i profesor (2002). Profesor zwyczajny na Akademii Górniczo-Hutniczej oraz Dyrektor Akademickiego Centrum Komputerowego Cyfronet AGH. Prowadzone prace badawcze dotyczą komputerowego sterowania procesami, systemów wizyjnych, systemów wieloprocesorowych, układów programowalnych, rekonfigurowalnych systemów obliczeniowych i sprzętowych metod akceleracji obliczeń.

e-mail: wiatr@agh.edu.pl

Streszczenie

Praca opisuje zmodyfikowany sposób budowania książki kodowej kodu Huffmana. Książka kodowa została zoptymalizowana pod kątem implementacji sprzętowej kodera i dekodera Huffmana w układach programowalnych FPGA. Opisano dynamiczną metodę kodowania – książka kodowa może się zmieniać w zależności od zmiennego formatu kompresowanych danych, ponadto musi być przesłana z kodera do dekodera. Sprzętowa implementacja kodeka Huffmana wymusza ograniczenie maksymalnej długości słowa, w przyjętym założeniu do 12 bitów, co pociąga za sobą konieczność modyfikacji algorytmu budowy drzewa Huffmana.

Słowa kluczowe: kompresja danych, FPGA, kodowanie Huffmana.

Implementation of Huffman compression with limited codeword length

Abstract

This paper presents a modified algorithm for constructing Huffman codeword book. Huffman coder, decoder and histogram calculations are implemented in FPGA similarly like in [2, 3]. In order to reduce the hardware resources the maximum codeword is limited to 12 bit. It reduces insignificantly the compression ratio [2, 3]. The key problem solved in this paper is how to reduce the maximum codeword length while constructing the Huffman tree [1]. A standard solution is to use a prefix coding, like in the JPEG standard. In this paper alternative solutions are presented: modification of the histogram or modification of the Huffman tree. Modification of the histogram is based on incrementing (disrupting) the histogram values for an input codeword for which the codeword length is greater than 12 bit and then constructing the Huffman tree from the very beginning. Unfortunately, this algorithm is not deterministic, i.e. it is not known how much the histogram should be disrupted in order to obtain the maximum codeword length limited by 12 bit. Therefore several iterations might be required. Another solution is to modify the Huffman tree (see Fig. 2). This algorithm is more complicated (when designing), but its execution time is more deterministic. Implementation results (see Tab. 1) show that modification of the Huffman tree results in a slightly better compression ratio.

Keywords: data compression, FPGA, Huffman coding.

1. Wstęp

Kompresja danych metodą Huffmana [1] polega na przyporządkowaniu każdemu znakowi słowa kodowego o różnej długości, w zależności od jego prawdopodobieństwa wystąpienia. Słowa występujące często mają krótsze słowa kodowe, a słowa występujące rzadko mają dłuższe słowa kodowe. Dzięki temu sumaryczna liczba bitów konieczna do zapisania ciągu znaku ulega zmniejszeniu.

Najbardziej czasochłonne obliczeniowo jest kodowanie i dekodowanie znaków oraz obliczanie histogramu. Dlatego zaproponowano aby niniejsze procedury zostały zaimplementowane sprzętowo w układach programowalnych FPGA [2, 3]. Aby to było możliwe konieczne stało się ograniczenie maksymalnej długości słowa kodowego do np. 12 bitów. Jak pokazano to w [2, 3], nie wpływa to znacząco na stopień kompresji a zdecydowanie upraszcza proces kodowania i dekodowania.

Niniejszy artykuł jest w głównej mierze poświęcony temu, w jaki sposób można ograniczyć maksymalną długość słowa kodowego na etapie budowania księgi kodowej. Budowa księgi kodowej odbywa się w sposób programowy (czyli jest projektowana w języku C/C++) niezależnie dla każdego bloku danych (czyli stosowane jest kodowanie dynamiczne). Najważniejszym etapem budowy księgi kodowej jest budowa drzewa Huffmana. Na każdym etapie tworzenia drzewa Huffmana łączy się dwa węzły o najmniejszej wartości histogramu (najmniejszym prawdopodobieństwie wystąpienia odpowiadającego symbolu / węzła) i tworzy nowy o wartości histogramu równej sumie wartości histogramów łączonych węzłów. Łączonym węzłom przypisywane są bity 0 i 1. Węzłami w pierwszej fazie tworzenia są liście (czyli znaki wejściowe, np. symbole ASCI). Słowo kodowe tworzy się przechodząc od ostatniego węzła (korzenia - o wartości równej liczbie wystąpień wszystkich znaków) do danego liścia reprezentującego konkretny znak ASCII. W ten sposób słowa kodowe mogą mieć maksymalną długość nawet 255 bitów (dla rozkładu wystąpień o wartościach kolejnych liczb ciągu Fibonacciego) [4]. Taka sytuacja jest niedopuszczalna ze względu na zaproponowaną konstrukcję kodera i dekodera, jak również format danych książki kodowej, która przesyłana jest do dekodera z zakodowanym tekstem.

Opracowano wiele metod ograniczających długość słów kodowych w drzewie. Jedną z nich jest metoda prefiksowa [5], która polega na zastosowaniu stałego prefiksu oraz transmisji słowa w sposób niezakodowany w przypadku kiedy długość tego słowa po zakodowaniu byłaby większa niż 12 bitów. Taka metoda kodowania może jednak wpływać na poziom kompresji – zarezerwowanie stałego 4-bitowego prefiksu dla symboli rzadkich powoduje, że symbole występujące częściej mają przypisane dłuższe słowa kodowe. Warto podkreślić, że np. w koderze/dekoderze JPEG maksymalna długość słowa wynosi 16 bitów, w zaproponowanym koderze/dekoderze maksymalna długość słowa wynosi

12 bitów, co powoduje, że kodowanie prefiksowe ma dużo większy wpływ na poziom kompresji. Dlatego w niniejszej pracy zostaną przedstawione dwie nowe metody ograniczania maksymalnej długości słowa: modyfikacja histogramu oraz modyfikacja drzewa Huffmana. Koder (oraz dekoder) z zastosowaniem obu metod został napisany w języku C w środowisku Visual Studio 2010 oraz w sposób sprzętowy zrealizowany w języku opisu sprzętu VHDL i środowisku EDK [2, 3].

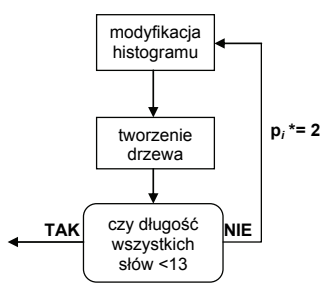
2. Zastosowane metody ograniczenia słów kodowych

2.1. Modyfikacja histogramu

Metoda modyfikacji histogramu polega na takiej modyfikacji wystąpień symboli aby w procesie tworzenia drzewa nie było sytuacji utworzenia słowa dłuższego niż 12 bitów. Wyrażając to innymi słowami zaburza się obliczony histogram, który koresponduje z prawdopodobieństwem p_i wystąpienia poszczególnego symbolu i. Dokonuje się tego poprzez podwyższenie wartości wystąpień dla symboli występujących rzadko i obniżenie dla symboli występujących często. W konsekwencji zaburza się rzeczywistą statystykę symboli wejściowych, co oczywiście pogarsza stopień kompresji.

Proces modyfikacji histogramu jest podzielony na kilka części:
- podwyższenie wartości dla symboli występujących rzadko (z prawdopodobieństwem niższym niż $p_i<2^{-12}$ oraz $p_i>0$ – nie ma sensu kodować symboli które w ogóle nie występują),
- posortowanie tablicy histogramu tak, aby znaki występujące najczęściej znalazły się na początku tablicy,
- obniżenie wartości histogramu dla najczęściej występujących znaków tak aby suma wszystkich prawdopodobieństw p_i wynosiła 1.

Taka procedura ma na celu zachowanie łącznej liczby wystąpień wszystkich znaków. Program zakłada plik wejściowy o rozmiarze maksymalnym $(2^{23}-1)B \approx 8MB$. Prawdopodobieństwo p_i wystąpienia znaku i (a w zasadzie liczba wystąpień znaku i – czyli wartość histogramu dla znaku i) jest podwyższane do $p_i= 2^{-12}$ jeśli $p_i<2^{-12}$ oraz $p_i>0$. Gdy po tej modyfikacji i tak powstanie słowo o słowie dłuższym niż 12 bitów, to procedura modyfikacji histogramu jest powtarzana, liczba wystąpień jest mnożona przez 2 czyli $p_i= 2 \cdot p_i$ (rys. 1). Pewną modyfikacją tego algorytmu (nie podaną w prezentowanych wynikach doświadczalnych) może być, zwiększenie p_i tak aby było ono równe najmniejszemu prawdopodobieństwu, dla którego długość słowa kodowego wynosi 12 bitów.

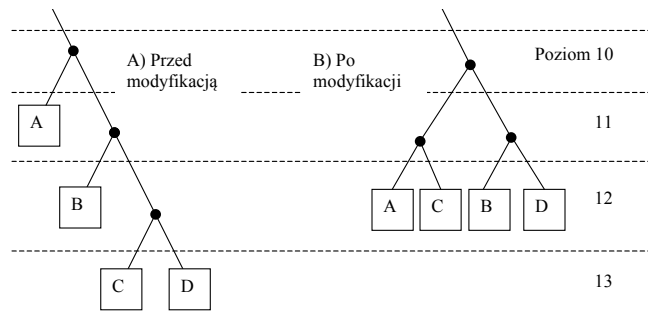


Rys. 1. Schemat modyfikacji histogramu
Fig. 1. Algorithm for histogram modification

Wadą takiego rozwiązania jest nieprzewidywalność czasu kodowania: czasochłonny proces tworzenia drzewa może być wykonywany wielokrotnie. Zaletą jest swoboda w ustaleniu wartości p_i, która decyduje o liczbie powtórzeń procesu modyfikacji i stopniu kompresji. Wartość ta nie musi być zmieniana poprzez jej podwojenie, ale może być np. zwiększana o stałą wartość.

2.2. Modyfikacja drzewa

Metoda ta polega na takiej modyfikacji drzewa Huffmana aby liczba poziomów tego drzewa (a tym samym długość najdłuższego słowa kodowego) nie przekraczała 12. W konsekwencji przenoszone są symbole, które znajdują się na poziomie 13 lub wyższym na poziom 12 lub niższy drzewa Huffmana (zob. rys. 2). Aby tego dokonać często konieczne jest zwolnienie miejsca na poziomie 12, czyli przeniesienie pewnych symboli z poziomu 11 na 12 lub w niektórych sytuacjach z poziomu 10 na 11, itd. Na tym etapie nie są modyfikowane poszczególne słowa kodowe a jedynie liczba słów kodowych występujących na każdym poziomie drzewa Huffmana. Słowa kodowe zostają przyznane w procesie tworzenia kodu kanonicznego [7], który pozwala znacznie zmniejszyć rozmiar książki kodowej przesyłanej z kodera do dekodera. Proces modyfikacji drzewa wymaga kilku poprzedzających go czynności:
- słowa dłuższe niż 12 bitów mają przypisywaną długość 13. Jest tak dlatego, że nie jest istotne jaką długość ma słowo kodowe, skoro wiadomo, że na pewno jest dłuższe niż 12 bitów. Zatem można przypisać im wspólną długość 13,
- tworzony jest histogram słów w funkcji ich długości, wszelka modyfikacja drzewa odbywa się właśnie na tym histogramie (drzewo jest tworzone poprzez odpowiednie przyporządkowanie znaków do histogramu zgodnie z wartością prawdopodobieństwa wystąpienia znaku – czym większe prawdopodobieństwo tym wcześniej następuje przypisanie).



Rys. 2. Fragment drzewa Huffmana przed i po modyfikacji
Fig. 2. A fragment of the Huffman tree before and after modification

Dopiero teraz można przystąpić do właściwej fazy modyfikacji.

Proces modyfikacji spełnia następujące założenia:
1) modyfikowana jest długość symboli o najdłuższych słowach kodowych mniejszych lub równych 11, tzn. jeżeli wszystkie słowa 13-bitowe można umieścić na drzewie wykorzystując słowa o długościach od *d* do *11*, to nie są modyfikowane słowa o długości mniejszej niż *d*. Na poziomie *d* nie są modyfikowane wszystkie znaki, tylko taka ilość, której modyfikacja pozwala umieścić wszystkie słowa 13-bitowe w drzewie Na rys. 2. odpowiada to przesunięciu symbolu A z poziomu 11 na poziom 12 w celu stworzenia dodatkowego miejsca na symbol C.
2) W powstałe punkcie 1 wolne miejsce umieszczane są symbole o długości 13. Na rys. 2 odpowiada to przesynięciu symbolu C z poziomu 13 do poziomu 12.
3) Przesunięcie symbolu w punkcie 2 powoduje, że towarzyszący mu symbol na poziomie 13 może być przesunięty na poziom 12. Na rys. 2 odpowiada to przesunięciu symbolu D z poziomu 13 na poziom 12.
4) Modyfikacja kończy się posortowaniem wszystkich znaków według prawdopodobieństwa wystąpienia. Operacja ta gwarantuje, że znaki, które wystąpiły rzadziej i miały przydzielone słowa dłuższe, po modyfikacji nadal będą miały przydzielone dłuższe słowa.

3. Dekoder

Zadaniem dekodera jest dekodowanie zakodowanego w koderze tekstu. Dekodowanie odbywa się przy użyciu tablic LUT [6]:
- tablice LUT są indeksowane słowami kodowymi i uzupełniane wszystkimi kombinacjami zer i jedynek
- wartością tablicy LUT jest znak ASCII przypisany do danego słowa oraz długość słowa podobnie jak to odbywało się w koderze [2, 3].

Przykład zapisu znaku o podanym słowie kodowym:

słowo kodowe: **1 1 1 0 0 0 1 1 0 0** (bity od korzenia do liścia)
wartość ASCII = 123
długość słowa = 10
słowo kodowe zostanie uzupełnione dwoma bitami, czyli: 00, 01, 10 i 11.
elementy tablicy LUT o indeksach :
od **1 1 1 0 0 0 1 1 0 0 0 0** do **1 1 1 0 0 0 1 1 0 0 1 1**
mają wartość w której zapisany jest kod ASCII i długość słowa kodowego.

1 1 1 0 0 0 1 1 0 0 0 0 = $2^4 + 2^5 + 2^9 + 2^{10} + 2^{11} = 3632$

LUT[3632] → ASCII i długość kodu (123 i 10)
LUT[3633] → ASCII i długość kodu (123 i 10)
LUT[3634] → ASCII i długość kodu (123 i 10)
LUT[3635] → ASCII i długość kodu (123 i 10)

Z dekodowanego tekstu zostaje odczytanych 12 bitów (maksymalna długość słowa). Do strumienia wyjściowego zostaje wysłany znak zapisany w tablicy LUT o indeksie równym wartości odczytanych bitów z zakodowanego tekstu. Na podstawie długości słowa zapisanego w tablicy LUT wiadomo o ile bitów ma zostać przesunięty wektor przeszukujący strumień wejściowy. Wektor przeszukujący tekst zostaje przesunięty i ponownie zostaje odczytanych 12 bitów.

Sprowadzenie kodu do postaci kanonicznej pozwala znacznie zaoszczędzić przestrzeń pamięci LUT. Bez tej modyfikacji potrzebna byłaby tablica LUT o rozmiarze 2^{12} (słowo wejściowe ma maksymalną długość 12 bitów). Procedura ta odbywa się zgodnie z metodą opisaną w RFC 1951 [7]. Postać kanoniczna kodu charakteryzuje się tym, że:
- wszystkie słowa kodowe tej samej długości mają następujące po sobie wartości w takiej samej kolejności jak symbole, które reprezentują
- krótsze słowa kodowe poprzedzają dłuższe słowa kodowe.

W konsekwencji krótsze słowa znajdują się po lewej stronie drzewa a dłuższe po prawej stronie drzewa. Taka modyfikacja pozwala określić początkowe bity słów w zależności od ich długości. Zoptymalizowano rozmiar tablicy LUT z 2^{12} na siedem tablic, każda o 2^7 elementach. Takie rozwiązanie zakłada, że:
1) nie może być 9-bitowego słowa zaczynającego się od 0…
2) nie może być 10-bitowego słowa zaczynającego się od 10…
3) nie może być 11-bitowego słowa zaczynającego się od 110…
4) nie może być 12-bitowego słowa zaczynającego się od 1110…

Warto podkreślić, że powyższe założenie można zawsze spełnić jeśli liczba wejściowych słów kodowych jest nie większa niż 256. Podobne rozwiązanie zostało zaprezentowane w [8].

4. Wyniki

Wykonano testy kompresji na przykładowych plikach tekstowych. Zastosowano opisane powyżej metody modyfikacji długości słów kodowych. Dla plików o dużych rozmiarach metoda modyfikacji histogramu dała lepsze wyniki niż modyfikacja drzewa.

Tab. 1. Rozmiar danych w bajtach przed i po kompresji dla różnych algorytmów i plików
Tab. 1. Data size in bytes before and after compression for different algorithms and files

plik \ metoda:	oryginalny	modyfikacja histogramu	modyfikacja drzewa
plik 1	6097310	3703774	3855040
plik 2	7429680	4435980	4533696

5. Wnioski

W wyniku badań stwierdzono, że zarówno algorytm modyfikacji drzewa jak i histogramu może być zaimplemetowany i przynosi podobne efekty. Pierwszym odczuciem autorów było jednak, że modyfikacja drzewa Huffmana da lepsze wyniki kompresji ponieważ modyfikuje drzewo Huffmana w sposób bardziej systematyczny. Wynik implementacji pokazał jednak że lepsze rezultaty daje modyfikacja histogramu, która jest prostsza w projektowaniu (prostszy kod programu). Niestety modyfikacja histogramu posiada niedeterministyczny czas wykonywania, nie wiadomo ile iteracji algorytmu należy dokonać aby najdłuższy kod był nie większy niż 12 bitów. W konsekwencji, algorytm ten może nie nadawać się do niektórych systemów wbudowanych, dla których czas obliczeń powinien być deterministyczny, ponieważ np. moce obliczeniowe oraz bufor wejścia /wyjścia takiego systemu są ograniczone.

Warto podkreślić, że nie ma sensu kompresować małego bloku danych, ponieważ dla każdego bloku danych konieczne jest wysłanie książki kodowej z kodera do dekodera. Książka kodowa zajmuje 256×(12-bitów słowa + 4-bity długości) = 512B. Poprzez zastosowanie kodowania kanonicznego możliwe jest wysłanie tylko długości słów kodowych, czyli 256×4bity = 128B. W tym wypadku dekoder musi sam zbudować książkę kodową na podstawie długości słów kodowych podobnie jak to odbywa się w koderze. W konsekwencji dekoder musi być wyposażony w procesor, np. soft-procesor MicroBlaze dostarczany w ramach pakietu EDK firmy Xilinx.

Praca finansowana ze środków NCBiR w ramach projektu SYNAT (SP/I/1/77065/10).

6. Literatura

[1] Huffman D.A.: A method for the construction of minimum-redundancy codes, Proc. Inst. Radio Eng, Vol.40, No.9, pp.1098-1101, Sep. 1952.
[2] Jamro E., Wielgosz M., Wiatr K.: Implementacja Adaptacyjnego Kodera Huffmana w Układach FPGA, Materiały Krajowej Konferencji „Reprogramowalne Układy Cyfrowe", Szczecin 12-13 Maj 2005, pp. 207-214.
[3] Jamro E., Wielgosz M., Wiatr K.: FPGA Implementation of the Dynamic Huffman Encoder, Proc. IFAC Workshop on Programmable Devices and Embedded Systems, Brno, Feb. 14-16, 2006, pp.60-65.
[4] Przelaskowski A.: Kompresja danych. Wydawnictwo BTC, Warszawa 2005.
[5] Drozdek A.: Wprowadzenie do kompresji danych, Wydawnictwa Naukowo-Techniczne, Warszawa 1999.
[6] Aspar Z., Mohd Yusof Z., Suleiman I.: Parallel Huffman decoder with an optimized look up table option on FPGA, Proc. TENCON 2000, vol.1, pp. 73-76.
[7] DEFLATE Compressed Data Format Specification version 1.3: RFC1951, 05/1996.
[8] Aspar, Z., Mohd Yusof, Z., Suleiman, I.: Parallel Huffman decoder with an optimized look up table option on FPGA, ENCON 2000. Proceedings , vol.1, no. vol.1, 2000, pp.73-76.

otrzymano / received: 02.04.2012
przyjęto do druku / accepted: 01.06.2012 *artykuł recenzowany / revised paper*