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

Optymalizacja sprzętowej architektury kompresji danych metodą słownikową

Inż. Maciej GWIAZDOŃ

Ukończył studia pierwszego stopnia na AGH w roku 2013 na wydziale Informatyki, Elektroniki i Telekomunikacji na kierunku Elektronika i Telekomunikacja. Obecnie student studiów II stopnia, na tym samym kierunku. Jego zainteresowania naukowe dotyczą kompresji danych oraz rekonfigurowalnych układów obliczeniowych.

e-mail: gwiazdon@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

Streszczenie

Niniejszy artykuł opisuje nową architekturę sprzętową kompresji słownikowej, np. LZ77, LZSS czy też Deflate. Zaproponowana architektura oparta jest na funkcji haszującej. Poprzednie publikacje były oparte na sekwencyjnym odczycie adresu wskazywanego przez pamięć hasz, niniejszy artykuł opisuje układ, w którym możliwe jest równoległe odczytywanie tego adresu z wielu pamięci hasz, w konsekwencji możliwa jest kompresja słownikowa z szybkością na poziomie 1B ciągu wejściowego na takt zegara. Duża szybkość kompresji jest okupiona nieznacznym spadkiem stopnia kompresji.

Słowa kluczowe: kompresja danych, FPGA, kodowanie Huffmana.

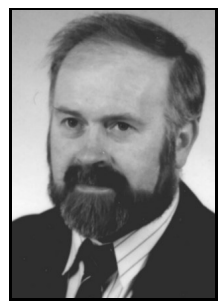
FPGA implementation of Deflate standard data decompression

Abstract

This paper describes a novel parallel architecture for hardware (ASIC or FPGA) implementation of dictionary compressor, e.g. LZ77 [1], LZSS [2] or Deflate [4]. The proposed architecture allows for very fast compression – 1B of input data per clock cycle. A standard compression architecture [8, 9] is based on sequential hash address reading (see Fig. 2) and requires *M* clock cycles per 1B of input data, where *M* is the number of candidates for string matching, i.e. hashes look ups (*M* varies for different input data). In this paper every hash address is looked up in parallel (see Fig. 3). The drawback of the presented method is that the number of *M* is defined (limited), therefore the compression ratio is slightly degraded (see Fig. 4). To improve compression ratio, a different sting length may be searched independently, i.e. not only 3B, but also 4B, … *N* B hashes (see results in Fig. 5, 6). Every hash memory (*M×(N-2)*) usually requires a direct look-up in the dictionary to eliminate hash false positive cases or to check whether a larger length sting was found. In order to reduce the number of dictionary reads, an additional pre-elimination algorithm is proposed, thus the number of dictionary reads does not increase rapidly with growing *N* (see Fig. 7).

Keywords: data compression, Deflate, LZ77, FPGA.

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

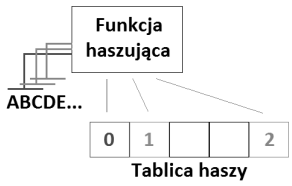
1. Wstęp

Słownikowa kompresja danych, np. LZ77 / LZSS [1, 2, 3] jest szeroko stosowaną bezstratną metodą kompresją danych między innymi w standardzie Deflate [4]. Standard Deflate jest używany np. w plikach .gz oraz .zip. Kodowanie słownikowe polega na wyszukiwaniu w poprzednio zakodowanych danych (czyli słowniku) takiego samego ciągu danych jak aktualny ciąg wejściowy. Na przykład, dane wejściowe: ***ABC****D****ABC****F*, można zakodować jako: *ABCD(l=3, d=4)F*, gdzie *l* oznacza długość znalezionego ciągu a *d* oznacza dystans (odległość) znalezionego ciągu w słowniku. Warto podkreślić, że według standardu Deflate, kodowanie słownikowe umożliwia również kodowanie długości serii - Run Length Encoding (RLE). Na przykład ciąg wejściowy *ABCABCABCA*, może być zakodowany jako: *ABC(l=7, d=3)*. Kodowanie RLE jest użyte dla *l>d*.

Kodowanie słownikowe zastosowane w standardzie Deflate jest bliższe kodowaniu LZSS niż kodowaniu LZ77 i jest skojarzone z kodowaniem Huffmana [5]. W porównaniu z kodowaniem LZSS, w algorytmie Deflate nie jest tworzony dodatkowy bit świadczący o tym, czy kodowany jest standardowy znak (literał) czy też kodowana jest para (*l, d*). Zamiast tego alfabet kodowanych literałów (zwykle znaki od 0 do 255) jest zwiększany o dodatkowe znaki takie jak: 256- End of Block czyli znak końca bloku; 257- długość znalezionego ciągu *l*= 3; 258 – *l*= 4; itd. Aby ograniczyć liczbę dodatkowych symboli, dla długości znalezionego ciągu większej niż 10, stosuje się dodatkowe bity *extra,* które nie podlegają kodowaniu Huffmana. Po każdym symbolu o wartości większej niż 256 kodowany jest dystans *d.* Podobnie jak dla długości kopiowanego ciągu, jest on kodowany w postaci odpowiedniego kodu (od 0 do 29) oraz dodatkowych bitów *extra.*

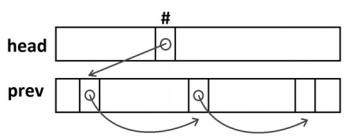
W kodowaniu słownikowym głównym problemem jest znalezienie jak najdłuższego ciągu danych w słowniku takiego samego jak aktualne dane wejściowe. W algorytmie naiwnym zgodnie ze standardem Deflate należałoby więc przeszukać cały słownik o rozmiarze 32kB mniej więcej dla każdego znaku (bajtu) wejściowego. W konsekwencji niniejsza operacja wiąże się z bardzo dużą ilością operacji przeszukiwania, co zajmowałoby znaczący czas procesora. Niemniej istnieją takie rozwiązania sprzętowe o ograniczonej wielkości słownika [6, 7]; rozwiązanie [7] zajmuje praktycznie całe zasoby układu FPGA.

Rozwiązaniem niniejszego problemu jest zastosowanie tablic haszujących (ang. hash) [3] i przeszukiwanie tylko tych elementów, które mają takie same hasze. W algorytmie Deflate minimalna długość ciągu kompresowana słownikowo wynosi 3B, dla mniejszej długości kodowanie pary (*l, d*) staje się często nieopłacalne w stosunku do 2B danych. W konsekwencji liczony jest hasz z 3B danych wejściowych i dalej sprawdzane są wszystkie wystąpienia w słowniku o tym samym haszu (zob. rys. 1).



Rys. 1. Przykład haszowania
Fig. 1. Example of hashing

Twórca standardu Deflate udostępnił odpowiedni program napisany w języku C, umożliwiający kompresję i dekompresję danych. Zastosowano tam mechanizm haszowania zwany chained hash table – tablicy łańcuchów haszy. Polega on na utworzeniu tablic *head* i *prev*. Tablica *head* ma 2^k wpisów, gdzie *k* to długość wyjściowa hasza w bitach, tablica *prev* ma ilość wpisów równą wielkości słownika w bajtach. Każdą trójkę danych wprowadzaną do słownika haszuje się, otrzymując indeks tablicy *head,* pod który zostanie wprowadzona jej pozycja w oknie. Poprzednia wartość tablicy *head* zostaje przepisana do tablicy *prev* pod adres wystąpienia wprowadzanej trójki. W ten sposób tworzony jest łańcuch powiązań, takich samych trójek (zob. rys. 2). Niniejsza metoda została sprzętowo zaimplementowana w [8, 9].

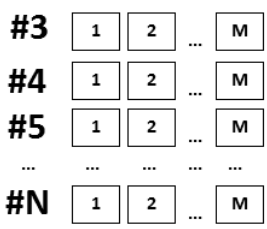


Rys. 2. Tablica łańcuchów haszy
Fig. 2. Table of hash sequence

W przypadku sprzętowej realizacji, wadą niniejszej metody jest to, że jest ona sekwencyjna: nie ma możliwości jej zrównoleglenia. Potrzeba co najmniej jednego taktu zegara w celu sprawdzenia pojedynczego wpisu w tablicy *prev*.

2. Zaproponowana architektura

Założeniem projektowym zaproponowanej nowej architektury jest to, aby zajmowała ona relatywnie niewiele zasobów sprzętowych oraz aby była ona w stanie kompresować z szybkością 1B ciągu wejściowego na takt zegara. Może to się odbywać kosztem niewielkiego spadku stopnia kompresji. W konsekwencji zaproponowano nową architekturę równoległą, która została przedstawiona na rys. 3.



Rys. 3. Schemat ogólny zaproponowanej architektury
Fig. 3. General scheme of the proposed architecture

W niniejszej architekturze używa się wielu tablic (pamięci) *head* osobnej, dla każdego wpisu (1..*M*). W konsekwencji w tablicy 1 wpisany jest adres pierwszego wystąpienia ciągu o takim samym haszu jak aktualny ciąg wejściowy. W tablicy 2 wpisany jest adres drugiego wystąpienia hasza, itd. Każde wystąpienie hasza musi być odpowiednio aktualizowane, czyli adres danej wejściowej jest wpisywany do tablicy 1, adres z tablicy 1 jest przepisywany do tablicy 2, itd. W układach FPGA pamięci BRAM mają osobną magistralę danych wejściowych i wyjściowych, także takie przepisywanie jest bezpośrednio skojarzone z odczytem danych i odbywa się w jednym takcie zegara (zapis i odczyt odbywa się pod ten sam adres – określony przez wartość hasza). Dodatkowo może być tworzony drugi wymiar tablic, osobno dla hasza z 3B danych wejściowych; osobno dla hasza z 4B danych wejściowych, itd. Sens drugiego wymiaru zostanie wyjaśniony później.

Zaletą niniejszego rozwiązania jest to, że mamy dostęp do wszystkich haszy równolegle, czyli możliwa jest duża szybkość działania niniejszego układu. Wadą niniejszego rozwiązania jest to, że liczba wpisów dla określonego hasza jest ograniczona do liczby pamięci *M*. W tym miejscu należy nadmienić, że w algorytmie tablicy łańcuchów haszy, w momencie kiedy optymalizowana jest szybkość a nie stopień kompresji, maksymalna liczba odczytów dla pojedynczego ciągu wejściowego jest ograniczana. Na przykład, dla ciągu wejściowego składającego się z 2^{15} takich samych symboli (czyli generującego takie same hasze dla 2^{15} pozycji), konieczne byłoby sprawdzenie wszystkich 2^{15} wpisów, co wydaje się być bezsensowne.

Następną wadą niniejszego rozwiązania jest to, że długość wyjściowa hasza wyrażona w bitach *k* (liczba bitów adresowych pamięci) jest z reguły duża i wynosi 12-15 bitów. Czym większe *k,* tym mniejsze prawdopodobieństwo błędnego wskazania hasza, które wynosi 2^{-k}. Błędne wskazanie polega na tym, że inny ciąg wejściowy generuje ten sam hasz, co wynika bezpośrednio z teorii haszy. W układach FPGA pojedyncza pamięć BRAM ma pojemność rzędu 4kB (32kb), czyli może pomieścić 2048 16-bitowych wpisów (adres kołowy jest 16-bitowy). Dlatego w proponowanym rozwiązaniu zaproponowano zmniejszenie szerokości bitowej hasza do *k*=11, co wiąże się z większym współczynnikiem błędnego wskazania, ale powoduje zdecydowane obniżenie wymaganej pojemności pamięci.

Zastosowanie różnych szerokości danych wejściowych, z których obliczany jest hasz 3..*N*, czyli drugi wymiar zaproponowanej architektury ma na celu lepsze wyszukiwanie dłuższych ciągów. W dalszej części tej pracy oznaczenie haszy jest następujące *H(i=3..N, j=1..M*) zob. rys. 3. Liczba pamięci haszy *j=1..M* jest ograniczona, dlatego może się zdarzyć, że pamięć haszy dla *i*=3 zostanie zapełniona samymi trójkami, natomiast ciągi 4B lub dłuższe nie będą znalezione. Zapobiega temu dodatkowy wymiar *i*=4..*N*, który znajduje tylko ciągi o długości 4B lub większej. Dodatkową funkcją tego rozwiązania jest preeliminacja błędnych haszy. Otóż zgodnie z teorią haszy, hasz może wskazywać błędne wskazanie, ale nie może pominąć poprawnego wskazania. Dlatego porównując adresy *H(i, j)* oraz *H(i+1,k)* można wskazać błędne wskazania haszy. Występują tutaj 3 róże sytuacje:

1) prawie pewne wskazanie: jeśli $H(i, j) = H(i+1, k)$, czyli hasz dla różnych *i* wskazują taki sam adres. W konsekwencji znaleziono zarówno *i* oraz *i+1* symboli co potwierdza poprawność wskazania. Prawdopodobieństwo mylnego wskazania spadło teraz do 2^{-2k} czyli jest bliskie zera.

2) poza zakresem (brak potwierdzenia): jeśli $H(i+1, j) > H(i, M)$, w takim wypadku wystąpiło dużo ciągów o długości *i* co powoduje, że adres wskazania *H(i+1, j)* nie jest już pokrywany przez krótsze hasze.

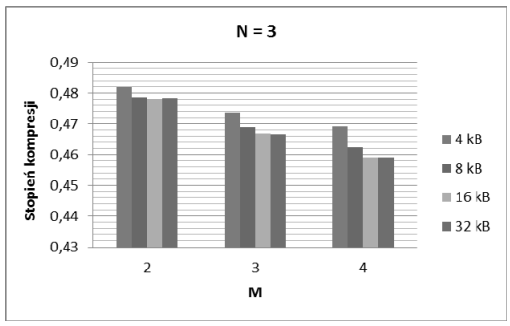
3) błędne wskazanie: jeśli $H(i+1, j) \neq H(i, k)$ dla każdego *k*=1..*M*. Nie ma możliwości aby znaleźć ciąg o długości *i+1* i równocześnie nie było ciągu o długości *i* (oczywiście pod warunkiem, że nie jest spełniony warunek 2).

Preeliminacja błędnych haszy jest ważną cechą niniejszej architektury ponieważ, każde wskazanie *H(i,j)* musi być dodatkowo sprawdzone z pamięcią słownika. Czyli pamięć słownika bez preeliminacji musi być $M \times (N-2)$ portowa. Liczba portów pamięci BRAM w układach FPGA jest ograniczona do 2, niemniej pamięć słownika składa się z wielu pamięci BRAM dlatego rzeczywista

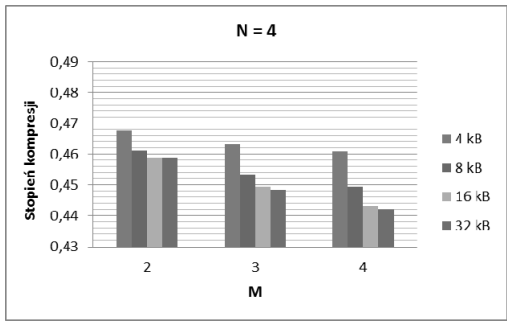
liczba portów pamięci słownika może być równa $2\times P$ (gdzie P-liczba pamięci BRAM słownika), pod warunkiem, że równomierne wykorzystanie wszystkich pamięci słownika. Ponadto odczyt słownika może odbywać się w różnych chwilach czasowych.

3. Wyniki implementacji

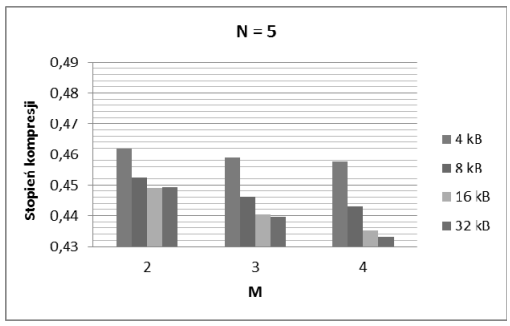
Testy przeprowadzono dla pliku tekstowego *book1*, który jest częścią Calgary Corpus [10]. Wyniki zostały zaprezentowane w formie wykresów, które pokazują stopień kompresji (Rozmiar Wyjściowy / Rozmiar Wyjściowy), rozmiar słownika (4kB, 8kB, 16kB, 32kB), N oraz M zgodnie z rys. 3.



Rys. 4. Stopień kompresji dla N=3
Fig. 4. Compression ratio for N=3



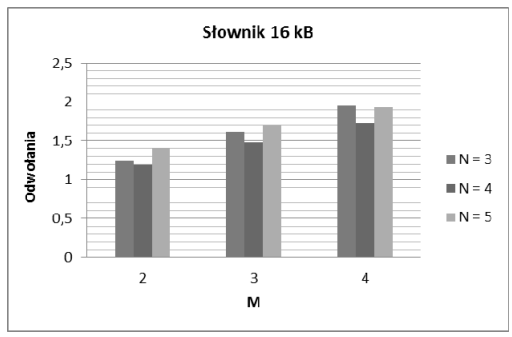
Rys. 5. Stopień kompresji dla N=4
Fig. 5. Compression ratio for N=4



Rys. 6. Stopień kompresji dla N=5
Fig. 6. Compression ratio for N=5

Uzyskany poziom kompresji jest zadowalający biorąc pod uwagę, że rozwiązanie programowe (program PKZIP) osiąga 40% kompresję dla trybu normalnego i 47,5% dla trybu szybkiego. Należy jednak podkreślić, że cechą tego rozwiązania jest równoległe, więc szybsze wyszukiwanie powtórzeń.

Słuszność przedstawionego rozwiązania dowodzi także uzyskana liczba odwołań do słownika (zob. rys. 7), unormowana do rozmiaru danych wejściowych. Dla N=3 nie jest możliwe zastosowanie mechanizmu preeliminacji (wyłącznie jeden poziom haszy), pokazuje to wzorcową liczbę sprawdzeń. W rzeczywistości liczba sprawdzeń powinna wynosić $(N\text{-}2)\cdot M$, ale w przypadku znalezienia ciągu w słowniku, nie ma sensu sprawdzanie kopiowanych znaków. Wyniki dla N=4 (dwa poziomy, dwa razy więcej danych, ale zastosowana preeliminacja) pokazują, że liczba odwołań, pomimo zwiększonej ilości możliwych wystąpień ciągów, nie rośnie znacząco. Dzięki preeliminacji odrzucone zostaje część błędnych haszy wynikających z kolizji bądź wskazań poza słownik.



Rys. 7. Ilość odwołań do słownika na 1B pliku wejściowego dla słownika 16 kB
Fig. 7. The number of dictionary memory reads per 1B of input data for dictionary size equal to 16kB

4. Podsumowanie

W niniejszym artykule zaproponowano nowy algorytm wyszukiwania ciągów w słowniku za pomocą równoległego haszowania. Dzięku temu szybkość kompresji wynosi 1B ciągu wejściowego na takt zegara. Algorytm ten jest dużo szybszy od porównywalnych algorytmów i zajmuje nieznacznie więcej zasobów. Odbywa się to niestety kosztem nieznacznego zmniejszenia stopnia kompresji.

5. Literatura

[1] Ziv, Jacob; Lempel, Abraham (May 1977): A Universal Algorithm for Sequential Data Compression. IEEE Transactions on Information Theory 23 (3): 337-343.
[2] Lempel–Ziv–Storer–Szymanski (LZSS) http://en.wikipedia.org/
[3] Salomon D.: A Concise Introduction to Data Compression, Springer, 2008.
[4] Deutsch P.: DEFLATE Compressed Data Format Specification version 1.3: RFC1951, 05/1996.
[5] 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.
[6] Morales-Sandoval M., Feregrino-Uribe C.: On the Design and Implementation of an FPGA based Lossless Data Compressor, Congreso Internacional de Cómputo Reconfigurable FPGAs, Colima, México, pp. 29-38, Sep. 2004.
[7] Mehboob C., Khan S. A., Ahmed Z., Jamal H., Shahbaz M.: Multigig Lossless Data Compression Device, Consumer Electronics, IEEE Transactions on, On page(s): 1927 - 1932 Volume: 56, Issue: 3, Aug. 2010.
[8] Rigler S., Bishop W., Kennings A.: FPGA-Based Lossless Data Compression using Huffman and LZ77 Algorithms, Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on, 22-26 April 2007, pp. 1235 – 1238.
[9] Shcherbakov I., Weis C., Wehn N.: A High-Performance FPGA-Based Implementation of the LZSS Compression Algorithm, 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, 2012, pp. 449-453.
[10] Arnold R., Bell T.: A corpus for the evaluation of lossless compression algorithms, http://corpus.canterbury.ac.nz/descriptions/#calgary.

otrzymano / received: 09.05.2013
przyjęto do druku / accepted: 03.07.2013 *artykuł recenzowany / revised paper*