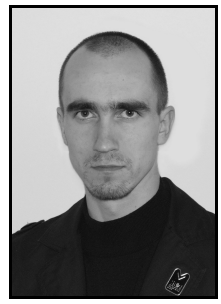
Marek GLISZCZYŃSKI, Alexandr ȚARIOV
ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY,
WYDZIAŁ INFORMATYKI

Szybki algorytm splotu kołowego dla $N = 2^m$

Marek GLISZCZYŃSKI

Jest studentem piątego roku Wydziału Informatyki Zachodniopomorskiego Uniwersytetu Technologicznego. Jego zainteresowania naukowe to algorytmy cyfrowego przetwarzania oraz transmisji sygnałów, sprzętowe wspomaganie oraz zrównoleglenie obliczeń, technologie mobilne, systemy i sieci komputerowe.



e-mail: mgliszczynski@wi.ps.pl

Dr hab. inż. Alexandr ȚARIOV

Ukończył studia na Wydziale Automatyki i Urządzeń Obliczeniowych Uniwersytetu Miernictwa w Sewastopolu, obronił pracę doktorską w 1984 r., habilitacyjną - w 2001 r. Jest kierownikiem katedry Architektury Komputerów i Telekomunikacji na Wydziale Informatyki Zachodniopomorskiego Uniwersytetu Technologicznego. Jego zainteresowania naukowe to algorytmy cyfrowego przetwarzania oraz transmisji sygnałów, sprzętowe wspomaganie oraz zrównoleglenie obliczeń.

e-mail: atariov@wi.ps.pl

Streszczenie

W pracy został przedstawiony szybki algorytm liczenia splotu kołowego *N*-elementowych wektorów danych ze zredukowaną liczbą operacji arytmetycznych (lub układów mnożących i sumatorów, jeśli chodzi o implementację sprzętową) w przypadku, gdy $N=2^m$, m – liczba całkowita. Pozwala to przy implementacji zmniejszyć nakłady obliczeniowe lub zapotrzebowanie na zasoby sprzętowe oraz stworzyć dogodne warunki do efektywnej realizacji operacji splotu kołowego w dowolnym sprzętowo-programowym środowisku implementacyjnym.

Słowa kluczowe: szybkie algorytmy, splot kolowy, notacja macierzowa.

Fast circular convolution algorithm for $N = 2^m$

Abstract

In the work the fast algorithm for 2^n-point circular convolution calculating with the reduced number of arithmetic operations (or multipliers and adders – in hardware implementation case) is presented. Computational procedure for describing the algorithm, based on the successful decomposition of the circulant matrix of arbitrary order is shown. This approach allows to lower hardware expenses and to create favorable conditions for effective convolution realization in the reprogrammable platform. Computational procedure for circular convolution realization can be described by means of matrix algebra notation. Matrix algebra offers not only a formalism for describing the algorithm, but it enables the derivation by pure algebraic manipulations of an algorithm that is well suited to be implemented in vector and matrix digital signal processors with various levels of parallelism. In addition, the mentioned procedures can be directly used for easy implementation in matrix-oriented languages like Matlab.

Keywords: fast algorithms, circular convolution, matrix notation.

1. Wstęp

Splot kołowy (Circular Convolution – CC) jest podstawową operacją cyfrowego przetwarzania sygnałów [1].

W przypadku operacji bazowej splotu kołowego mamy do czynienia z dwoma wektorami danych (sygnałami cyfrowymi) $\mathbf{X}_{N\times 1}=[x_0, x_1, ..., x_{N-1}]^{\mathrm{T}}$ i $\mathbf{H}_{N\times 1}=[h_0, h_1, ..., h_{N-1}]^{\mathrm{T}}$ oba o rozmiarze N elementów (próbek), przy czym operacja ta w postaci macierzowej jest opisana w sposób następujący:

$$\mathbf{Y}_{N\times 1} = \mathbf{H}_N \cdot \mathbf{X}_{N\times 1}, \tag{1}$$

gdzie: $\mathbf{Y}_{N\times 1}=[y_0, y_1, ..., y_{N-1}]^{\mathrm{T}}$ jest wektorem wyników, zaś macierz $\mathbf{H}_N = \blacksquare_{i=0}^{N-1}(\mathbf{I}_N^{(\leftarrow i)} \cdot \mathbf{H}_{N\times 1})$ - macierzą, której komponentami są w odpowiedni sposób ulokowane elementy wektora $\mathbf{H}_{N\times 1}$.

Znaczenie wykorzystanych tu i w dalszej części artykułu symboli przedstawiono poniżej:
$\mathbf{I}_N^{(\leftarrow\alpha)}$ - macierz jednostkowa o wymiarze określonym za pomocą dolnego indeksu, natomiast indeks górny (jeżeli jest) wskazuje liczbę pozycji cyklicznego przesunięcia wierszy macierzy jednostkowej w kierunku wskaźnika [2]; $\blacksquare$ - symbol poziomej konkatenacji dwóch lub więcej macierzy [3]; $\oplus$ - symbol sumy prostej (tensorowej) dwóch macierzy [6];

Wzór (1) przybiera wówczas postać:

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix} = \begin{bmatrix} h_0 & h_{N-1} & \cdots & h_1 \\ h_1 & h_0 & \cdots & h_2 \\ \vdots & \vdots & \vdots & \vdots \\ h_{N-1} & h_{N-2} & \cdots & h_0 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} \tag{2}$$

Łatwo zauważyć, że realizacja procedury (1) wymaga N^2 operacji mnożenia oraz $N(N-1)$ operacji dodawania. Tak duża liczba mnożeń stała się czynnikiem stymulującym do poszukiwań efektywniejszych rozwiązań algorytmicznych. Pojawiło się wiele algorytmów, których celem jest minimalizacja tych operacji [4]. Najbardziej znanym podejściem do efektywnego wyznaczania splotu kołowego dwóch *N*-elementowych wektorów jest sprowadzenie tego zadania do wyznaczenia iloczynu tych wektorów w domenie wybranej dyskretnej transformaty ortogonalnej, a następnie przeniesienie wyniku do domeny czasowej za pomocą odwrotnej transformaty. Zazwyczaj taka operacja może być efektywnie zrealizowana za pomocą szybkiej transformaty Fouriera (Fast Fourier Transform - FFT), szybkiej transformaty Hartleya (Fast Hartley Transform – FHT), transformat liczbowych Fermata (Fermat number transform -FNT) lub Mersene'a (Mersene number transform - MNT) oraz innych dobrze znanych oraz powszechnie stosowanych „szybkich" algorytmów dyskretnych transformat ortogonalnych. Do obliczania splotu kołowego bezpośrednio w domenie czasu również opracowano dość dużo algorytmów. Do rozwiązań algorytmicznych tego typu należą m.in.: algorytmy „krótkich" splotów Winograda, algorytm Agarwala-Cooleya, algorytmy Reeda-Truonga, Nussbaumera [5-10]. Przeważnie są to algorytmy, które zostały opracowane dla konkretnych długości *N* wektora danych wejściowych. Znany jest też algorytm wykorzystujący „krótkie" sploty do obliczania splotów wektorów długości *N* (wielokrotność „krótkich" ciągów) za pomocą metody „zagnieżdżania" [11-12]. Natomiast dla wektorów danych o dowolnej długości *N*, chociażby będącej potęgą dwójki, tak jak w FFT, ogólnego algorytmu bezpośredniego (w domenie czasowej) wyznaczania splotu kołowego bodajże jak dotąd nie opublikowano. Właśnie taki „szybki" algorytm syntetyzowany za pomocą metody [13] został przedstawiony w niniejszej pracy.

2. Synteza „szybkiego" algorytmu liczenia splotu kołowego dla $N = 2^m$

W celu syntezy struktury algorytmicznej specjalizowanej jednostki procesorowej realizującej operację bazową splotu kołowego wprowadźmy kilka konstrukcji macierzowych:

- macierze definiujące przekształcenia na iteracjach poprzedzających mnożenia:

$$\mathbf{A}^{(k)}_{K^{(k-1)}\times K^{(k)}} = (\mathbf{E}_2 \oplus (\mathbf{I}_{L^{(k)}} \otimes \mathbf{T}_{2\times 3})) \otimes \mathbf{I}_{2^{m-k}};$$

- macierze definiujące przekształcenia na iteracjach następujących po mnożeniu:

$$\widetilde{\mathbf{A}}^{(k)}_{K^{(k)}\times K^{(k-1)}} = (\mathbf{E}_2 \oplus (\mathbf{I}_{L^{(k)}} \otimes \mathbf{T}_{3\times 2})) \otimes \mathbf{I}_{2^{m-k}},$$

gdzie: $k = \overline{1,m}$, $L^{(k)} = \begin{cases} \sum_{i=0}^{k-2} 3^i, k>1 \\ 0, \quad k\le 1 \end{cases}$, $K^{(k)} = \begin{cases} 2^{m-k}(2+3\sum_{i=0}^{k-2} 3^i), k>1 \\ 2^m, \quad k\le 1 \end{cases}$;

$$\mathbf{E}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},\ \mathbf{T}_{2\times 3} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix},\ \mathbf{T}_{3\times 2} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix};$$

- diagonalną macierz $\mathbf{S}_M = diag(s_0, s_1, \cdots s_{M-1})$, rzędu $M = 1 + \sum_{i=0}^{m-1} 3^i$

której elementy wyznaczane są na podstawie elementów wektora $\mathbf{H}_{N\times 1}$ według metody [13].

Na rysunku 1 został pokazany program w języku MatLab do wyznaczania elementów macierzy $\mathbf{S}_M$.

```
function Y = divCmatrix(X)
l=1;
d = length(X);
ll=0;
while d/l~=1    %dopóki wymiar podmacierzy = 1
Y = [];
i = 0;
while i~=l
x = X(1+d/l*i:d/l+d/l*i, 1+d/l*i:d/l+d/l*i);
                                %i-ta podmacierz
a = x(1:d/l/2,1:d/l/2); %podział podmacierzy
                                              na 4
b = x(1:d/l/2,d/l/2+1:d/l);
c = x(d/l/2+1:d/l,1:d/l/2);
i=i+1;
if b == c        %wybór odpowiedniego wzoru
tmp = 0.5*pdiag(a+b,a-b);
ll = ll+1;
Else
tmp = pdiag(b-a,pdiag(c-a,a));
ll = ll+2;
End
Y = pdiag(Y,tmp);  %dopisanie nowych podmacie-
                                          rzy do Y
end
l = l+ll;   %aktualizacja liczby podmacierzy
ll = 0;
X = Y;
d = length(X);
end
Y = X;
```

Rys. 1. Program w MatLab wyznaczania wartości elementów macierzy diagonalnej S_M

Fig. 1. MatLab program listing for matrix S_M element calculation

Uwzględniając wprowadzone konstrukcje macierzowe, szybki algorytm liczenia splotu kołowego można przedstawić następująco:

$$\mathbf{Y}_{N\times 1} = \mathbf{A}^{(1)}_{K^{(0)}\times K^{(1)}} \mathbf{A}^{(2)}_{K^{(1)}\times K^{(2)}} \cdots \mathbf{A}^{(m)}_{K^{(m-1)}\times K^{(m)}} \mathbf{S}_M \times \\ \times \widetilde{\mathbf{A}}^{(m)}_{K^{(m)}\times K^{(m-1)}} \widetilde{\mathbf{A}}^{(m-1)}_{K^{(m-1)}\times K^{(m-2)}} \cdots \widetilde{\mathbf{A}}^{(1)}_{K^{(1)}\times K^{(0)}} \mathbf{X}_{N\times 1} \quad (3)$$

Rozpatrzmy przykład proponowanego algorytmu w przypadku, gdy N=8. Wówczas procedura (3) przybiera postać:

$$\mathbf{Y}_{N\times 1} = \mathbf{A}^{(1)}_{K^{(0)}\times K^{(1)}} \mathbf{A}^{(2)}_{K^{(1)}\times K^{(2)}} \mathbf{A}^{(3)}_{K^{(2)}\times K^{(3)}} \mathbf{S}_M \times \\ \times \widetilde{\mathbf{A}}^{(3)}_{K^{(3)}\times K^{(2)}} \widetilde{\mathbf{A}}^{(2)}_{K^{(2)}\times K^{(1)}} \widetilde{\mathbf{A}}^{(1)}_{K^{(1)}\times K^{(0)}} \mathbf{X}_{N\times 1} \quad (4)$$

poszczególne zaś macierze są postaci:

$$\mathbf{A}^{(1)}_{8\times 8} = \mathbf{E}_2 \otimes \mathbf{I}_4 = \widetilde{\mathbf{A}}^{(1)}_{8\times 8},$$

$$\mathbf{A}^{(2)}_{8\times 10} = (\mathbf{E}_2 \otimes \mathbf{I}_2) \oplus (\mathbf{T}_{2\times 3} \otimes \mathbf{I}_2),$$

$$\mathbf{A}^{(3)}_{10\times 14} = \mathbf{E}_2 \oplus (\mathbf{I}_4 \otimes \mathbf{T}_{2\times 3}),$$

$$\widetilde{\mathbf{A}}^{(3)}_{14\times 10} = \mathbf{E}_2 \oplus (\mathbf{I}_4 \otimes \mathbf{T}_{3\times 2}),$$

$$\widetilde{\mathbf{A}}^{(2)}_{10\times 8} = (\mathbf{E}_2 \otimes \mathbf{I}_2) \oplus (\mathbf{T}_{3\times 2} \otimes \mathbf{I}_2),$$

$$\mathbf{S}_M = diag(s_0, s_1, s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10}, s_{11}, s_{12}, s_{13}),$$

gdzie poszczególne elementy ostatniej macierzy są wyliczane za pomocą programu pokazanego na rysunku 1 na podstawie sumowania algebraicznego elementów wektora $\mathbf{H}_{8\times 1}$.

$$s_0 = h_0 + h_1 + h_2 + h_3 + h_4 + h_5 + h_6 + h_7,$$

$$s_1 = h_0 - h_1 + h_2 - h_3 + h_4 - h_5 + h_6 - h_7,$$

$$s_2 = -h_0 - h_1 + h_2 + h_3 - h_4 - h_5 + h_6 + h_7,$$

$$s_3 = -h_0 + h_1 + h_2 - h_3 - h_4 + h_5 + h_6 - h_7,$$

$$s_4 = h_0 - h_2 + h_4 - h_6,$$

$$s_5 = h_0 - h_1 + h_2 + h_3 - h_4 + h_5 - h_6 + h_7,$$

$$s_6 = h_0 - h_1 + h_2 - h_3 - h_4 + h_5 - h_6 - h_7,$$

$$s_7 = -h_0 - h_2 + h_4 + h_6,$$

$$s_8 = h_0 + h_1 - h_2 + h_3 - h_4 - h_5 + h_6 - h_7,$$

$$s_9 = h_0 - h_1 - h_2 + h_3 - h_4 + h_5 + h_6 - h_7,$$

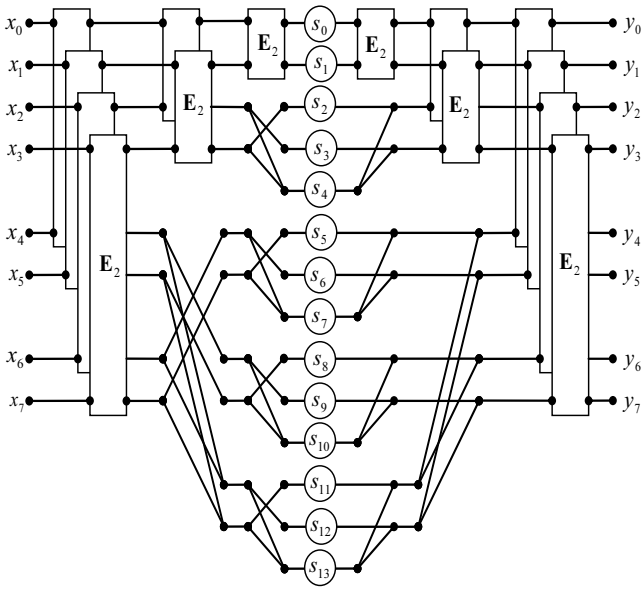
$$s_{10} = -h_0 + h_2 + h_4 - h_6,$$

$$s_{11} = -h_0 - h_3 + h_4 + h_7,$$

$$s_{12} = -h_0 + h_1 + h_4 - h_5,$$

$$s_{13} = h_0 - h_4$$

Rysunek 2 przedstawia model grafo-strukturalny ilustrujący organizacje procesu obliczeniowego wyznaczania splotu kołowego zgodnie z opracowaną procedurą dla N=8. Liniami prostymi oznaczone są operacji transferu danych. W przypadku tego modelu skupienie linii prostych w odpowiednich punktach oznacza operację dodawania, natomiast linie rozchodzące się (rozgałęzienia) – zwykłe operacje dublowania danych. Kółkami na tym grafie są przedstawione operacje mnożenia przez stałą w nich wpisaną, prostokątami zaś oznaczone zostały bloki mnożenia wpisanych w nich macierzy przez odpowiednie podwektory danych.



Rys. 2. Model grafostrukturalny organizacji procesu obliczeniowego wyznaczania splotu kołowego według procedury (3) dla przykładu N=8

Fig. 2. The graph-structural model of computation process organization for circular convolution calculation according to example (3) for N=8

3. Oszacowanie liczby operacji mnożenia oraz dodawania

Jak widać, zaproponowany w artykule algorytm pozwala zredukować łączną liczbę operacji mnożenia oraz dodawania względem metody „naiwnej". Dla rozważonego przykładu mamy 14 mnożeń zamiast 64 oraz 46 dodawań zamiast 56.

Dla dowolnego N liczba mnożeń niezbędnych do realizacji algorytmu może być opisana za pomocą następującego wzoru:

$$\theta_{\times} = 1 + \sum_{i=0}^{m-1} 3^i ,$$

natomiast liczba dodawań jest określona jako:

$$\theta_{+} = 2^{m+1} + \sum_{i=1}^{m-1} 2^{m-i} (2 + \frac{3}{2} \sum_{j=0}^{i-1} 3^j) .$$

W tabeli 1 przedstawione są wyniki oszacowania liczby operacji arytmetycznych w przypadku opracowanego algorytmu w porównaniu z algorytmem "naiwnym".

Tab. 1. Oszacowanie liczby operacji arytmetycznych dla różnych N

Tab. 1. Estimation of arithmetic operations for various N

N	mnożenia		dodawania	
	naiwny	proponowany	naiwny	proponowany
2	4	2	2	4
4	16	5	12	15
8	64	14	56	46
16	256	41	240	135
32	1024	122	992	394
64	4096	365	4032	1155
128	16384	1094	16256	3406
256	65536	3281	65280	10095
512	262144	9842	261632	30034
1024	1048576	29525	1047552	89595

4. Podsumowanie

W artykule opisano „szybki" algorytm liczenia splotu kołowego dla N-elementowych wektorów danych, gdy $N=2^m$.

Rozpatrzono przykład syntezy szybkiego algorytmu oraz zbudowano model grafo-strukturalny dla przykładu wektorów o długości N=8.

Oczywiste jest, że w podobny sposób mogą być realizowane efektywne (posiadające mniej operacji mnożenia i dodawania) algorytmy splotu kołowego dla dowolnych będących potęgą dwójki długości wektorów danych.

5. Literatura

[1] T. P. Zieliński, Cyfrowe przetwarzanie sygnałów: od teorii do zastosowań, WKŁ, Warszawa 2007.

[2] E.E. Dagman., G.A. Kukharev. Szybkie dyskretne transformaty ortogonalne, Wydawnictwo Nauka, 1983.

[3] A. Ţariov, Modele algorytmiczne i struktury wysokowydajnych procesorów cyfrowej obróbki sygnałów, Szczecin, Informa, 2001.

[4] R.E. Blahut, Fast algorithms for digital signal processing, Addison-Wesley Publishing company, Inc. 1985.

[5] R.C. Agarwal, J.W. Cooley, New algorithms for digital convolution. IEEE Transactions on Acoustics, speech, and Signal processing, vol ASSP-vol.AS25 (no.5), pp. 392-410, October 1977.

[6] H.J. Nussbaumer., Fast Fourier Transform and Convolution Algorithms, Springer-Verlag, 1982.

[7] C.S.Burrus, T.W. Parks, J.F. Potts, DFT/FFT and Convolution Algorithms and Implementation, John Willey & Sons, 1985.

[8] R. Tolimieri M. An, C. Lu, Algotithms for Discrete Fourier Transform and Convolution, Springer-Verlag, 1989.

[9] I.S. Reed, T. K. Truong, The Use of Finite Fields to Compute Convolutions, IEEE Trans. on Information Theory: IT-21, March 1975, pp. 208-213.

[10] J.H. McClellan, C.M. Rader, Number Theory in Digital Signal Processing, Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1979.

[11] W. Selesnik, C.S. Burrus, Extending Winograd's Small Convolution Algorithm to Longer Lengths, In Proceedings of the IEEE International Symposium on Circuits and Systems, June 1994, pp. 2.449-2.452.

[12] R. Stasiński, Extending sizes of effective convolution algorithms, Electronic letters, 1990, vol. 26, no 19, pp.1602-1604.

[13] A. Ţariov, Strategie racjonalizacji obliczeń przy wyznaczaniu iloczynów macierzowo-wektorowych. Metody Informatyki stosowanej, Nr 1, 2008, str. 147- 158.

Artykuł recenzowany