PROJEKT ZALICZENIOWY Z MATEMATYKI

Temat:

Algorytmy perfekcyjnego przetasowania

Michał Wasilkiewicz, 104109

1. **Wstęp**

Tematem projektu są algorytmy perfekcyjnego przetasowania, zaprezentowane na wykładzie z matematyki przez dr. Andrzeja Maćkiewicza. W projekcie omówiono zasadę, według której odbywa się tasowanie elementów wektora, oraz porównano czasy wykonania przetasowania, w zależności od długości wektora danych.

Odpowiednie przetasowanie, czyli ustawienie w pożądanej kolejności elementów wektora ma olbrzymie znaczenie przy obliczaniu szybkiej transformaty Fouriera (FFT) w sposób iteracyjny. Pozwala ono tak przygotować wektor danych, których FFT chcemy obliczyć, by zminimalizować liczbę operacji arytmetycznych wykonywanych przez komputer. Założono, że wykonujemy FFT na wektorach o liczbie elementów będącej potęgą liczby 2.

Wszystkie algorytmy zaimplementowano w programie Matlab.

1. **Algorytmy**
   1. **Przetasowanie bitowe**

Algorytm tasuje wektor, korzystając z bitowej reprezentacji indeksów tablicy. Sposób działania przedstawiono na przykładzie 8-elementwego wektora, wypełnionego dla ułatwienia i widoczności liczbami od 0 do 7. Dla każdego indeksu, wyznacza się indeks mu odpowiadający, będący lustrzanym odbiciem jego reprezentacji na najmniejszej wymaganej liczbie bitów. Po wyznaczeniu odpowiedniego numeru indeksu następuje zamiana miejsc elementów w wektorze. Poniżej wypisano w kolumnie elementy wektora w postaci binarnej oraz dziesiętnej przed oraz po wykonaniu algorytmu.

Wektor przed tasowaniem Wektor po tasowaniu

000 0 000 0

001 1 100 4

010 2 010 2

011 3 110 6

100 4 001 1

101 5 101 5

110 6 001 3

111 7 111 7

**Kod programu:**

function [x, time] = bit\_sh(x)

% Funkcja tasuje zamieniając bity indeksów

tic

n=length(x); %długość wektora

tmp = 0; %zmienna pomocnicza

for i=1:n/2

ind=bitswap(i-1,n); %wyznaczanie indeksu do zamiany

tmp = x(i); x(i) = x(ind+1); x(ind+1)=tmp; %operacja zamiany ( swap )

end

time = toc;

end

Program przyjmuje 1 argument – wektor danych, na którym wykonuje działania. Zwraca ten sam, przetasowany wektor, oraz czas wykonania (wyznaczony przy pomocy funkcji „tic” i „toc” zawartych w programie Matlab).

Wyróżniono w nim 1 główną pętlę, indeksowaną po **i**, która wykonuje się tylko do połowy długości wektora. Spowodowane jest to zależnością, że odpowiednik indeksu 1 wynosi 4, a odpowiednik indeksu **4** wynosi **1**. Pozwala ona, a nawet wymusza wykonanie pętli tylko do połowy, inaczej dwukrotnie zamienilibyśmy te elementy miejscami, przez co wektor po przetasowaniu byłby dokładnie tej samej postaci co oryginalny.

**Funkcja bitswap:**

function [index] = bitswap(arg, lenght)

mask = 1;

index = 0;

n=log2(lenght); %ilość bitów, potrzebnej do reprezentacji indeksów wektora

for i=1:n

index = bitshift(index,1);

if(bitand(arg,mask))

index = index+1;

end

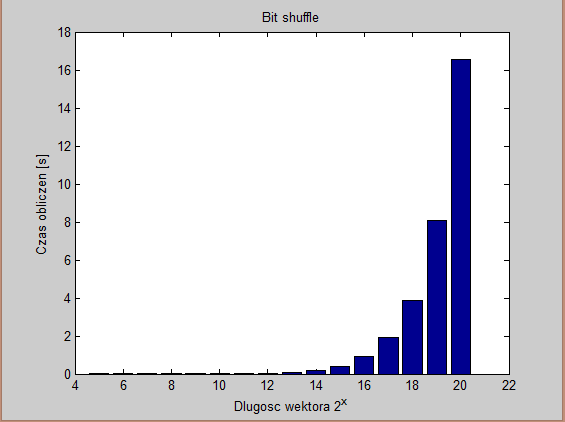
mask = bitshift(mask,1);

end

end

Funkcja „bitswap” służy do wyznaczenia lustrzanego odbicia indeksu, za pomocą bitów. Jako argumenty przyjmuje liczbę (indeks), oraz długość wektora danych, potrzebną do określenia minimalnej liczby bitów wymaganej do reprezentacji indeksów tablicy.

Na wykresie 1 zaprezentowano czas przetasowania wektora w zależności od jego długości. Program wykonano dla wektorów od długości od 2^5 do 2^20 elementów.



Wykres 1. Czas przetasowania w funkcji długości wektora

W tabeli 1 zamieszczono czasy przetasowań w zależności od długości wektora.

|  |  |
| --- | --- |
| Długość wektora | Czas tasowania [ms] |
| 2^5 | 0,2309 |
| 2^6 | 0,4922 |
| 2^7 | 1,1326 |
| 2^8 | 3,3724 |
| 2^9 | 5,1006 |
| 2^10 | 10,9193 |
| 2^11 | 19,746 |
| 2^12 | 43,1788 |
| 2^13 | 89,6246 |
| 2^14 | 187,8679 |
| 2^15 | 411,9267 |
| 2^16 | 947,4259 |
| 2^17 | 1840,4838 |
| 2^18 | 3892,6389 |
| 2^19 | 8096,4317 |
| 2^20 | 17013,5638 |

Tabela 1. Czasy przetasowań wektorów o danych długościach

Analizując dane zawarte w tabeli 1 łatwo zauważyć, że czas obliczeń wzrasta mniej więcej dwukrotnie, wraz ze wzrostem wykładnika potęgi o 1. Powoduje to bardzo szybki wzrost czasu obliczeń dla dużych wektorów danych. Wektor 2^16 tasowany jest około 1s, natomiast wektor 2^20 nieco ponad 17s.

* 1. **Przetasowanie z wykorzystaniem dodatkowego wektora**

Algorytm do przetasowania wektora wejściowego wykorzystuje dodatkowy wektor. Sposób działania został zobrazowany poniżej.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 etap – 2 sekcje, 4 podsekcje | | | 2 etap – 1 sekcja, 2 podsekcje | | |  | Wektor końcowy |
| S1 | PS1 | 0 | S1 | PS1 | 0 | PS | 0 |
| 1 | 2 | 4 |
| PS2 | 2 | 1 | 2 |
| 3 | 3 | 6 |
| S2 | PS1 | 4 | PS2 | 4 | 1 |
| 5 | 6 | 5 |
| PS2 | 6 | 5 | 3 |
| 7 | 7 | 7 |

Algorytm można podzielić na etapy, których ilość określa wzór E = log2(n)-1, gdzie n to długość wektora. Dla wektora o długości 8 elementów, tasowanie odbywa się w 2 etapach. Przed pierwszym etapem należy wyznaczyć liczbę sekcji S oraz liczbę podsekcji PS. Liczba sekcji jest zawsze 4-krotnie mniejsza od długości wektora. Liczba podsekcji jest 2 razy większa od liczby sekcji. Po każdym etapie liczba sekcji i podsekcji zmniejsza się dwukrotnie.

W pierwszym etapie, każde kolejne pary elementów tworzą podsekcje. Każde dwie podsekcje tworzą sekcje. W każdym kolejnym etapie sekcje z poprzedniego etapu zmieniają się w podsekcje. W każdym etapie obliczeń działamy osobno dla każdej sekcji, zapisując do wektora pomocniczego najpierw pierwszy element pierwszej podsekcji, potem pierwszy element drugiej podsekcji, następnie dugi element pierwszej i drugi element drugiej podsekcji. Czynność tę powtarzamy dla drugiej (i każdej kolejnej) sekcji. Po wykonaniu przepisania należy podmienić wektor pomocniczy z oryginalnym. W drugim, ostatnim etapie mamy tylko jedną sekcję. Do wektora pomocniczego zapisujemy kolejno pierwszy element pierwszej podsekcji, pierwszy element drugiej podsekcji, drugi element pierwszej podsekcji itd.

**Kod programu:**

function [vect, time]=sec\_vect\_sh(x)

tic

n=length(x);% długość wektora x

Ls=n/4;%liczba sekcji

Lps = 2 \* Ls; %Liczba podsekcji

E = log2(n)-1; %number of etapów

vect=zeros(1,n); % help vector

DLs = 4; %długość sekcji

DLps = 2; %długość podsekcji

for i=1:E %pętla po etapach

for j=1:Ls%pętla po sekcjach

for k=1+(j-1)\*DLs:DLs+(j-1)\*DLs % obliczanie indeksu tablicy pomocniczej do którego należy wstawić % konkretną liczbę

if (k==1+(j-1)\*DLs) || (k==DLs+(j-1)\*DLs)

vect(k)= x(k);

ip=k;inp=ip;

continue;

end

if mod(k,2)==0

vect(k)=x(ip+DLps);

ip=ip+1;

end

if mod(k,2)==1;

vect(k)=x(inp+1);

inp=inp+1;

end

end

end

x=vect;% podmiana wektora pomocniczego z oryginalnym

vect=zeros(1,n); % wyzerowanie wektora pomocniczego

Ls=Ls/2; % obliczenie nowej liczby sekcji

Lps = Ls\*2; % obliczenie nowej liczby podsekcji

DLs = DLs\*2;%obliczenie ilości elementów w sekcji

DLps = DLps\*2;% obliczenie ilości elementów w podsekcji

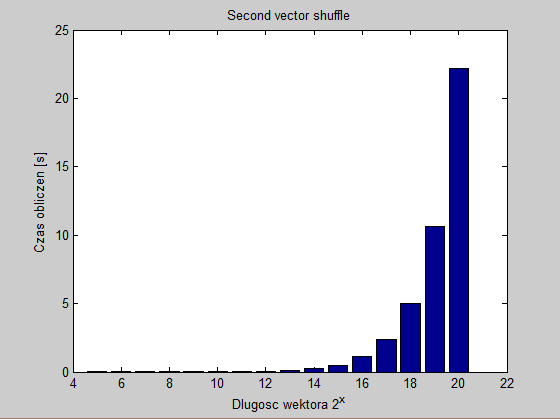
end

vect = x;% ostatnia podmiana wektora

time = toc;

end

Na wykresie 2 zaprezentowano czas przetasowania wektora w zależności od jego długości. Program wykonano dla wektorów od długości od 2^5 do 2^20 elementów.



Wykres 2. Czas przetasowania w funkcji długości wektora

W tabeli 2 zamieszczono czasy przetasowań w zależności od długości wektora.

|  |  |
| --- | --- |
| Długość wektora | Czas tasowania [ms] |
| 2^5 | 0,2281 |
| 2^6 | 0,4559 |
| 2^7 | 1,4463 |
| 2^8 | 3,4359 |
| 2^9 | 5,4843 |
| 2^10 | 11,2154 |
| 2^11 | 23,4086 |
| 2^12 | 52,5519 |
| 2^13 | 106,8715 |
| 2^14 | 234,4014 |
| 2^15 | 534,0078 |
| 2^16 | 1148,9233 |
| 2^17 | 2443,5751 |
| 2^18 | 5126,2504 |
| 2^19 | 10748,459 |
| 2^20 | 23082,4001 |

Tabela 2. Czasy przetasowań wektorów o danych długościach

Podobnie jak dla pierwszego algorytmu, także w tym przypadku czas obliczeń wzrasta mniej więcej dwukrotnie wraz ze wzrostem wykładnika potęgi o 1.

* 1. **Przetasowanie bez wykorzystania dodatkowego wektora (bjork)**

Kod algorytmu został zaprezentowany na wykładzie. Program jako argument przyjmuje wektor danych, na którym działa. Zwraca przetasowany wektor oraz czas obliczeń.

**Kod programu:**

function [x, time]=bjork\_sh(x)

tic;

n=length(x);

nv2=n/2;

nm1=n-1;

j=1;

for i=1:nm1

if(i<j)

tmp=x(i);x(i)=x(j);x(j)=tmp; %swap

end

k=nv2;

while(k<j)

j=j-k;k=k/2;

end

j=j+k;

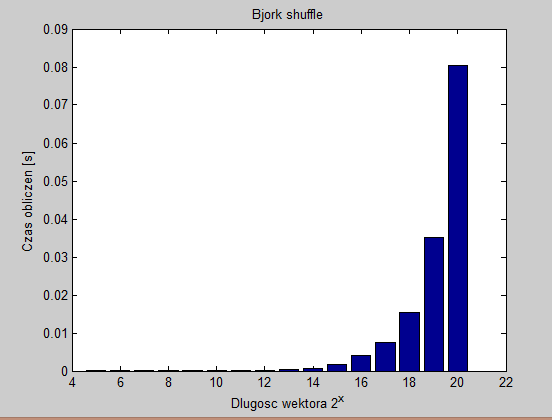
end

time = toc;

end

Program, z wykorzystaniem dwóch pętli i instrukcji warunkowych wyznacza indeks elementu, z którym należy dokonać zamiany pozycji w wektorze.

Na wykresie 3 zaprezentowano czas przetasowania wektora w zależności od jego długości. Program wykonano dla wektorów od długości od 2^5 do 2^20 elementów.



Wykres 3. Czas przetasowania w funkcji długości wektora

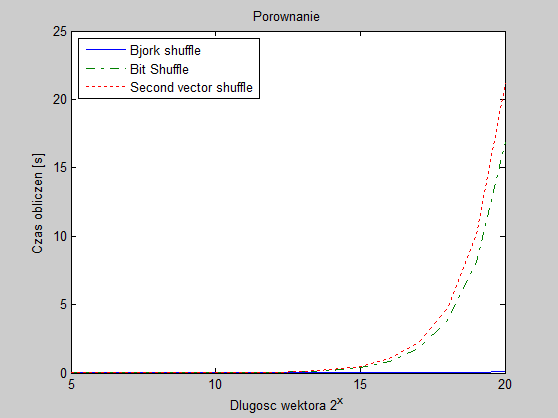
W tabeli 3 zamieszczono czasy przetasowań w zależności od długości wektora.

|  |  |
| --- | --- |
| Długość wektora | Czas tasowania [ms] |
| 2^5 | 0,0078 |
| 2^6 | 0,0055 |
| 2^7 | 0,0097 |
| 2^8 | 0,0161 |
| 2^9 | 0,0414 |
| 2^10 | 0,0517 |
| 2^11 | 0,0828 |
| 2^12 | 0,1602 |
| 2^13 | 0,3221 |
| 2^14 | 0,6304 |
| 2^15 | 1,5052 |
| 2^16 | 4,2594 |
| 2^17 | 7,6903 |
| 2^18 | 18,0251 |
| 2^19 | 35,0172 |
| 2^20 | 82,7333 |

Tabela 3. Czasy przetasowań wektorów o danych długościach

1. **Podsumowanie**

Na wykresie 4 przedstawiono porównanie czasów tasowania wszystkich algorytmów w funkcji długości wektora.



Wykres 4. Porównanie czasów działania algorytmów

Analizując wykres 4 łatwo zauważyć wyraźną przewagę algorytmu „bjork” nad pozostałymi. Nawet dla wektorów o długościach większych niż 2^15 jest on bardzo szybki. Jego szybkość jest spowodowana operowaniem na jednej tablicy – algorytm wykorzystujący dodatkową tablicę musi ją za każdym razem wypełniać, oraz zamieniać z oryginalną, co przy długich wektorach danych zajmuje dużo czasu. Przewaga nad przetasowaniem bitowym może wynikać z faktu, że w programie Matlab dostęp do bitów liczby jest utrudniony, przez co czas obliczeń może być większy. Do pewnego momentu efektywność algorytmu bitowego oraz wykorzysującego zapasowy wektor jest podobna, jednak przy wektorach rzędu 2^17 i większych szybsza okazuje się wersja działająca na bitach.