Implementacija algoritma Diskretne Furijeove transformacije kada je broj uzoraka prost broj

Seminarski rad u okviru kursa Naučno izračunavanje
Matematički fakultet

Čedomir Dimić

Septembar 2019.

Sadržaj

Abstrakt

Diskretna Furijeova transformacija niza od N tačaka, gde je N prost broj predstavlja cirkularnu konvoluciju [1]. Prerasporedjujući članove niza i koristeći primitivni koren broja N, moguće je izvršiti diskretnu Furijeovu transformaciju, ne gubeći na performansama iako je N prost broj.

1 Uvod

Algoritam implementiran u ovom radu, poznatiji kao Raderov algoritam, je algoritam brze Furijeove transformacije, koji računa diskretnu Furijeovu transformaciju kada je broj uzoraka prost, predstavljajući DFT kao cirkularnu konvoluciju [2]. U narednom pogavlju će prvo biti date neophodne definicije koje se koriste u implementaciji ovog problema.

2 Definicije

Neka su podaci koji treba da budu transformisani predstavljeni u obliku niza od N brojeva $\{a_i\}$, $i = 0, 1, ..., N-1$, gde zagrade predstavljaju ceo niz, a a_i predstavlja $i-ti$ član niza. Diskretna Furijeova transformacija je niz $\{ A_k\}$, $k = 0, 1, ..., N-1$ čiji članovi su dati sledećom jednakošću

$$A_k = \sum_{i=1}^{N-1} a_i exp(-j(2\pi/N)ik). \tag{1}$$

Ukoliko je N prost broj, onda postoji neki broj g, koji nije nužno jedinstven, takav da postoji 1 na 1 preslikavanje brojeva $i = 1, ..., N-1$ u $j = 1, ..., N-1$, tako da je $j = ((g^i))$. Dvostruke zagrade označavaju operaciju moduo:

$$((g)) = g \ moduo \ N \tag{2}$$

U teoriji brojeva, broj g se naziva **primitivnim korenom** broja N.

3 Računanje niza A_k kad je N prost broj

U jednakosti (1) je dat izraz za računanje A_k za svako k. Za A_0 je jednostavno,

$$A_0 = \sum_{i=0}^{N-1} a_i, \tag{3}$$

i to možemo izračunati direktno. Niz $A_k - a_0$, $k = 1, 2, ..., N-1$ se može izračunati na sledeći način

$$A_k - a_0 = \sum_{i=1}^{N-1} a_i exp(-j(2\pi/N)ik). \tag{4}$$

Možemo iskoristiti naredne jednakosti

$$i \to ((g^i)), k \to ((g^k)), ((g^{N-1})) = ((g^0)) \tag{5}$$

i formulu transformisati u ovaj oblik:

$$(A_{((g^k))} - a_0) = \sum_{i=0}^{N-1} a_{((g^i))} exp(-j(2\pi/N)g^{i+k}). \tag{6}$$

Sada se može videti da je niz $\{A_{((g^k))} - a_0\}$ cirkularna konvolucija niza $\{a_{((g^i))}\}$ i niza $\{exp(-j(2\pi/N)g^i\}$. Ovakve funkcije se mogu efikasno izračunati korišćenjem FFT algoritma. Ako je N prost broj, $N-1$ mora biti složen. Onda u tački $N-1$ cirkularna konvolucija može da se predstavi kao inverzna diskretna Furijeova transformacija proizvoda diskretne Furijeove transformacije niza$\{a_{((g^{-i}))}\}$ i niza $\{exp(-j(2\pi/N)g^i\}$. Naredne operacije koje su označene sa DFT su izvedene FFT algoritmom.

$$\{A_{((g^k))} - a_0\} = DFT^{-1}\Bigg\{(DFT\{a_{((g^{-i}))}\})\Bigg(DFT\Bigg\{exp\Bigg(-j(2\pi/N)g^i\Bigg)\Bigg\}\Bigg)\Bigg\} \tag{7}$$

Ovakva implementacija će biti efikasna ako je $N-1$ jako složen[1]. Ako je $N-1$ umereno složen, kao npr. $N = 563$, ušteda koju dobijemo koristeći FFT, će biti nedovoljna, jer će se više puta računati DFT. Drugi način se zasniva na činjenici da se cirkularna konvolucija može izračunati kao deo cirkularne konvolucije sa većim brojem tačaka. Neka je N' jako složen broj veći od $2N-4$, pravimo niz tačaka N', $\{b_i\}$ tako što umećemo $(N'-N+1)$ nula izmedju nultih i prvih tačaka $\{a_{((g^{-i}))}\}$ i pravimo drugi niz tačaka N', $\{c_i\}$, periodično ponavljajući niz $\{exp(-j(2\pi/N)g^i)\}$, sve dok je prisutno N' tačaka. Na ovaj način postižemo da inverzni DFT proizvoda DFT-a niza $\{b_i\}$ i niza $\{c_i\}$ sadrži $\{A_{((g^k))} - a_0\}$ kao podniz prvih $N-1$ tačaka. Kako se N' može izabrati tako da bude jako složen, čak i stepen dvojke, može se iskoristiti FFT algoritam da bi se izračunao DFT ovih nizova.

4 Implementacija

U ovom poglavlju je dat kod implementacije metode opisane u prethodnom poglavlju. Funkcija $isPrime(n)$ ispituje da li je broj prost, funkcija $primeFactors(factors, n)$ pronalazi sve faktore broja n, a funkcija $powerModulo(x, y, z)$ računa x^y po modulu z. Te tri funkcije se koriste u funkciji $smallestPrimitive(n)$, koja vraća najmanji primitivni koren broja n. To su sve pomoćne funkcije koje se koriste u funkciji $dftPrime(arr, n)$ koja računa diskretnu Furijeovu transformaciju kada je broj uzoraka prost broj. Funkcija $nextPowerOfTwo(n)$, za neki broj n, pronalazi najmanji broj koji je veći od njega i koji je stepen dvojke. Ona se koristi u funkciji $improvedDftPrime(arr, n)$, koja predstavlja poboljšanu verziju algoritma opisanog u poslednjem pasusu prethodnog poglavlja.

```
def dftPrime(arr, n):

    # find smallest primitive
```

[1]Jako složen broj je pozitivan prirodan broj koji ima više delilaca nego bilo koji drugi pozitivan prirodan broj manji od njega.

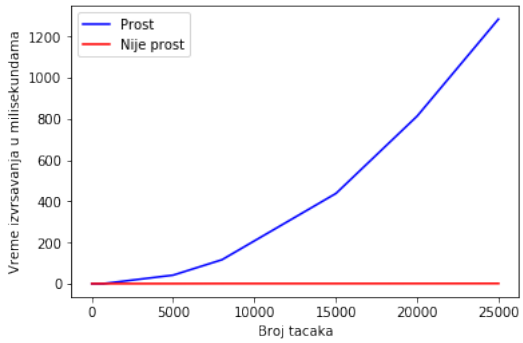
```
    g = smallestPrimitive(n)

    # make an array of g^i
    g_i = np.zeros(n-1, dtype=int)
    for i in range(0, n-1):
        g_i[i] = powerModulo(g, i+1, n)


    # make an array of g^(-i)
    g_minus_i = np.zeros(n-1, dtype=int)
    for i in range(0, n-1):
        g_minus_i[i] = powerModulo(g, n-i-2, n)

    # make the first product for ifft
    fft1 = arr[g_minus_i]

    # make the second product for ifft
    fft2 = []
    for i in g_i:
        fft2.append(np.exp(-2j*np.pi*(1/n)*i))

    # initialize the result
    A = np.zeros(n, dtype=complex)
    A[0] = np.sum(arr)

    # compute the ifft
    inv_dft_arr = fft.ifft(fft.fft(fft1)*fft.fft(fft2))

    # populate the result
    for k in range(1, n):
        A[powerModulo(g, k+1, n)] = arr[0] + inv_dft_arr[k-1]
    return A


def improvedDftPrime(arr, n):

    # find smallest primitive
    g = smallestPrimitive(n)

    # make an array of g^i
    g_i = np.zeros(n-1, dtype=int)
    for i in range(0, n-1):
        g_i[i] = powerModulo(g, i+1, n)


    # make an array of g^(-i)
```

```
g_minus_i = np.zeros(n-1, dtype=int)
for i in range(0, n-1):
    g_minus_i[i] = powerModulo(g, n-i-2, n)

# make the first product for ifft
fft1 = arr[g_minus_i]

# find next power of two greater than 2 * n - 4
nP = 2 * n - 3
nP = nextPowerOfTwo(nP)

# add zeros between zeroth and fisrt element in the first product
tmp_fft1 = np.zeros(nP, dtype=int)
tmp_fft1[0] = fft1[0]
for i in range(nP - n + 2, nP):
    tmp_fft1[i] = fft1[i - nP  + n - 1]

# make the second product for ifft
fft2 = np.zeros(n-1, dtype=complex)
j = 0
for i in g_i:
    fft2[j] = (np.exp(-2j*np.pi*(1/n)*i))
    j += 1

counter = n - 1
tmp_fft2 = np.zeros(nP, dtype=complex)
i = 0
while (i < nP):
    for j in range(0, n-1):
        if i >= nP:
            break
        tmp_fft2[i] = fft2[j]
        i += 1


# initialize the result
A = np.zeros(n, dtype=complex)
A[0] = np.sum(arr)

# compute the ifft
inv_dft_arr = fft.ifft(fft.fft(tmp_fft1)*fft.fft(tmp_fft2))

# populate the result
for k in range(1, n):
    A[powerModulo(g, k+1, n)] = arr[0] + inv_dft_arr[k-1]
return A
```

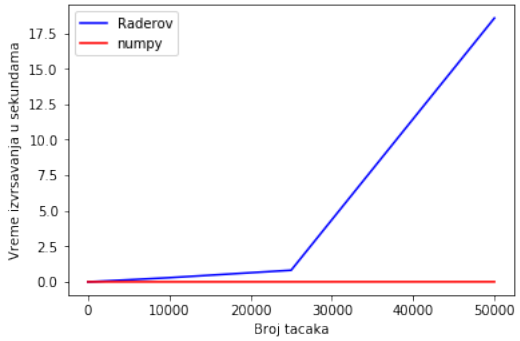
5 Rezultati

U ovom pogavlju će biti predstavljeni rezultati poredjenja vremena izvršavanja ugradjenog algoritma iz numpy biblioteke i algoritma implementiranog u ovom radu. Takodje će biti prikazano i poredjenje poboljšane verzije Raderovog algoritma u odnosu na običnu, kada $N-1$ nije jako složen broj, gde je N broj uzoraka.

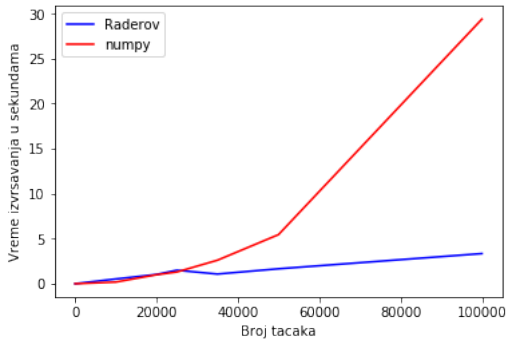
Na narednom grafiku je prikazano vreme izvršavanja numpy FFT algoritma kada je broj uzoraka prost i kada nije. Može se primetiti da je algoritam dosta sporiji kada je broj uzoraka prost broj.



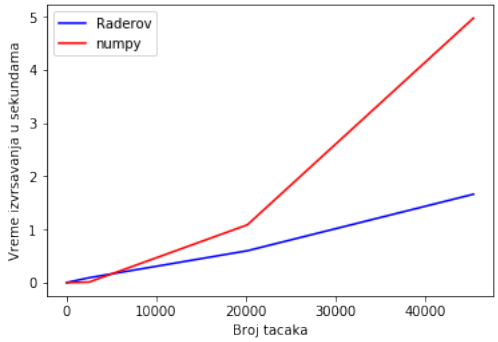
Na narednom grafiku je prikazano poredjenje Raderovog algoritma i ugradjenog numpy algoritma kada broj uzoraka nije prost broj. Sa grafika se vidi da je ugradjeni algoritam značajno brži.



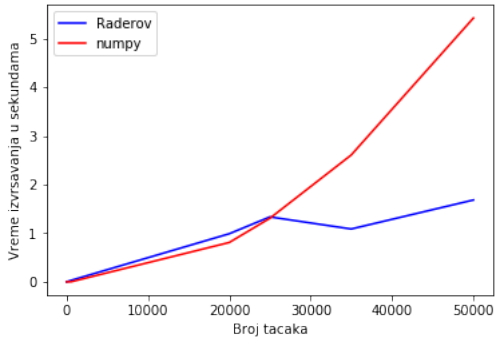
Na narednom grafiku je prikazano poredjenje Raderovog algoritma i ugradjenog numpy algoritma kada broj uzoraka nije prost broj. Za razliku od prethodnog grafika, ovde se vidi koliko je Raderov algoritam brži od ugradjenog kada je broj uzoraka prost broj.



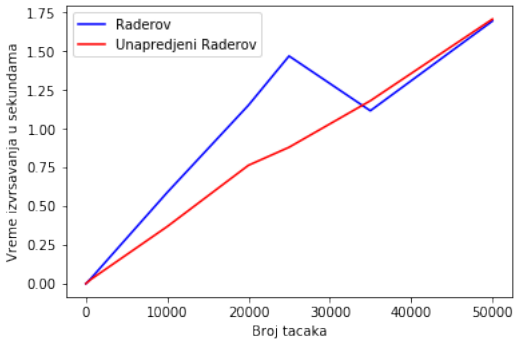
Na sledećem grafiku je prikazano poredjenje Raderovog algoritma i numpy algoritma, ali samo u slučaju kada $N - 1$ nije jako složen.



Na sledećem grafiku je prikazano poredjenje Raderovog algoritma i numpy algoritma, ali samo u slučaju kada $N - 1$ jeste jako složen.



Na narednom grafiku je prikazano poredjenje Raderovog algoritma sa poboljšanim algoritmom. Može se primetiti da kod poboljšanog algoritma vreme izvršavanja raste linearno i da je nešto brži dok se ne dostigne odredjeni broj tačaka.



U narednoj tabeli je prikazano vreme izvršavanja numpy algoritma, Raderovog algoritma kao i poboljšanog algoritma za različiti broj uzoraka. Iz nje se može primetiti da je ugradjeni numpy algoritam brži za mali broj uzoraka, a kako se broj uzoraka povećava Raderov algoritam postaje brži tako da će u slučaju da ima oko 100 hiljada uzoraka biti i do 10 puta brži. Takodje se može primetiti da je poboljšani algoritam brži od običnog kada $N-1$ nije jako složen broj (ako je $N = 20161$, odnosno $N-1$ je jako složen, vidi se da su vremena izvršavanja približna), ali da se povećavanjem broja uzoraka oba algoritma ponašaju slično i potrebno im je približno isto vremena da se izvrše. Vremena izvršavanja u tabeli su data u sekundama.

Broj uzoraka	numpy algoritam	Raderov algoritam	poboljšani algoritam
7	0.0001	0.0007	0.0005
181	0.0002	0.008	0.004
563	0.01	0.03	0.01
10007	0.18	0.5	0.3
19997	0.88	0.97	0.63
20161	1.01	0.67	0.64
24989	1.28	1.37	0.78
34981	2.59	1.08	1.16
45361	4.37	1.42	1.5
49999	5.31	1.67	1.83
99991	31.56	3.35	3.51
110881	34.16	3.74	3.85

6 Primena Raderovog algoritma na FPGA

Različiti FFT algoritmi se mogu koristiti za modelovanje FPGA čipova koristeći Verilog jezik. FPGA komponente se mogu koristiti za implementaciju logičkih funkcija koje mogu da se izvedu koristeći integrisana kola [4]. Raderov

algoritam je jedan od njih. Utvrdjeno je da je Raderov algoritam loš u poredjenju sa nekim drugim FFT algoritmima kada se posmatra operaciona frekvencija zbog vremena potrebnog za izračunavanje [3]. Medjutim, koristeći Raderov algoritam, potrebno je manje elemenata za implementaciju FPGA nego ako se koriste neki drugi FFT algoritmi kao što su Koli-Tukijev algoritam i Gud-Tomasov algoritam. Iako je u tom pogledu lošiji od Radiks-2 algoritma, za razliku od njega ne zahteva da broj uzoraka bude stepen dvojke [3].

7 Zaključak

U ovom radu je prikazano poredjenje Raderovog algoritma i ugradjenog algoritma iz numpy biblioteke i došlo se do zaključka da je Raderov algoritam značajno brži ako je broj uzoraka jako veliki. Takodje je dato i poredjenje poboljšane i obične verzije algoritma, gde se vidi da je ona brža od obične ako $N-1$ nije jako složen broj, gde je N broj uzoraka, ali da se povećavanjem broja tačaka ta razlika smanjuje i da je vreme izvršavanja obe verzije algoritma u tom slučaju približno isto.Danas se ovaj algoritam uglavnom predstavlja kao specijalan slučaj Vinogradovog algoritma koji je nadogradio Raderov algoritam, tako da može da računa DFT i u slučaju ako je broj uzoraka stepen prostog broja, gde je i eksponent takodje prost broj.

Literatura

[1] Charles Rader *Discrete Fourier Transforms When the Number of Data Samples is Prime* M.I.T. Lincoln Lab, Lexington, Massachusetts, 1968

[2] `https://en.wikipedia.org/wiki/Rader%27s_FFT_algorithm`

[3] `https://www.researchgate.net/publication/265283495_Implementing_FFT_algorithms_on_FPGA`

[4] `https://sr.wikipedia.org/wiki/FPGA`