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].

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 $smallestPrimitive(n)$, u koja vraća najmanji primitivni koren broja n. Funkcija $dftPrime(arr, n)$ izvršava Diskretnu Furijeovu transformaciju kad je n prost broj na način opisan u prethodnom poglavlju.

```
# coding: utf-8

import numpy as np
from numpy import fft

# check if number is Prime
def isPrime(n):
```

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

```
    if n > 1:
        for i in range(2, n // 2):
            if (n % i) == 0:
                return False
        return True
    else:
        return False

# find prime factors of a number
def primeFactors(factors, n):
    i = 2
    while i * i <= n:
        if n % i:
            i += 1
        else:
            n = n // i
            factors.append(i)
    if n > 1:
        factors.append(n)
    return factors

# calculate  x^y%z
def powerModulo(x, y, z):
    result = 1
    x = x % z
    while y > 0:
        if (y & 1):
            result = (result * x) % z

        y = y >> 1
        x = (x * x) % z
    return result

# find smallest primitive root of a number
def smallestPrimitive(n):
    factors = []

    if (isPrime(n) == False):
        return -1

    #using Euler function
    phi = n - 1
    primeFactors(factors, phi)

    for i in range(2, phi + 1):
        flag = False
```

```
        for j in factors:
            if (powerModulo(i, phi // j, n) == 1):
                flag = True
                break
        if flag == False:
            return i

    return -1

# compute the DFT
def dftPrime(arr, n):

    # find smallest primitive
    g = smallestPrimitive(n)

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


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

    # make the first product ffft
    fft1 = np.zeros(n, dtype=int)
    fft1 = arr[g_minus_i]

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

```python
        A[powerModulo(g, k, n)] = arr[0] + inv_dft_arr[k]
    return A


# test
arr = np.array([3, 2, 1, -3, 0, 4, 6])
A = dftPrime(arr, 7)
print(A)
print()

arr = np.array([3, 2, 1, -3, 0])
A = dftPrime(arr, 5)
print(A)
print()

arr = np.array([3, 2, 1, -3, 0, 4, 6, 13, -2, 0, 4])
A = dftPrime(arr, 11)
print(A)
```

5 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].

6 Zaključak

Zapažanje da DFT može da se izrazi kao konvolucija može biti od koristi, jer to znači da jedna mreža sa fiksiranim tačkama može da izračuna sve tačke DFT-a. 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$_{F}FT_{a}lgorithm$

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

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