5. EFIKASNO IZRAČUNAVANJE DISKRETNE FURIJEOVE TRANSFORMACIJE

5.1 UVOD

U prethodnoj glavi istaknut je značaj koji Diskretna Furijeova transformacija ima u obradi diskretnih i digitalnih signala. Zbog toga je pitanje efikasnog izračunavanja DFT na digitalnim računarima opšte namene, kao i u specijalizovanim sistemima za obradu signala, od velikog teorijskog i praktičnog značaja. U daljem izlaganju će prvo biti analizirana složenost izračunavanja DFT po definicionoj formuli (4.16) i ukazano na nekoliko jednostavnih mogućnosti za povećanje brzine izračunavanja. Zatim će biti objašnjen Gercelov (Goertzel) algoritam, koji se odlikuje povećanom efikasnošću, a koji je primenljiv u slučajevima kada se izračunava mali broj odbiraka DFT (u literaturi o DFT umesto termina odbirak često se koristi termin tačka). Proučavanje grupe algoritama koja je poznata pod opštim nazivom Brza Furijeova transformacija (Fast Fourier Transform - FFT) započeće se algoritmima za preuređivanje u vremenu i algoritmima za preuređivanje po frekvenciji, a obuhvatiće i mnogo generalnije, i ponekad brže, algoritme sa prostim faktorima. Na kraju, biće opisani i neki algoritmi za izračunavanje DFT koji koriste brzu konvoluciju, a koji mogu, zbog raspoložive hardverske podrške, u nekim slučajevima biti najbolja alternativa.

5.2 ARITMETIČKA SLOŽENOST IZRAČUNAVANJA DFT

Posmatrajmo definicioni izraz za izračunavanje Diskretne Furijeove transformacije (4.16) koji će, radi preglednosti, biti ponovo napisan:

$$X(\tfrac{2\pi}{N}k) = X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} = \sum_{n=0}^{N-1} x[n]W_N^{kn}, \quad k = 0,1,2,\ldots,N-1 \qquad (5.1)$$

U izračunavanju izraza (5.1) pojavljuju se četiri osnovne operacije: *izračunavanje trigonometrijskih funkcija*, *množenje*, *sabiranje* i *izračunavanje indeksa* (adresa podataka). Broj potrebnih operacija za izračunavanje svih N DFT odbiraka, u najopštijem slučaju kada je $x[n]$ kompleksna sekvenca, iznosi:

1. Broj izračunavanja trigonometrijskih funkcija - $2N^2$
2. Broj realnih množenja - $4N^2$
3. Broj realnih sabiranja - $2N(2N-1) \approx 4N^2$

Broj izračunavanja indeksa, kao i njihov uticaj na složenost izračunavanja, teško je proceniti jer zavisi od arhitekture računarskog sistema i programskog jezika koji se koristi. U svakom

slučaju, uticaj izračunavanja indeksa na složenost izračunavanja znatno je manji od uticaja prve tri klase računskih operacija.

Kao primer programske implementacije direktnog DFT algoritma, na slici 5.1 je prikazan potprogram DFT napisan na programskom jeziku FORTRAN. Potprogram obavlja direktnu ili inverznu DFT, u zavisnosti od vrednosti ulazne promenljive INV (INV = 0: DFT, INV = 1: IDFT). Ulazni podaci se nalaze u vektorima XR (realni deo) i XI (imaginarni deo). Izračunate vrednosti izlaznih odbiraka nalaze se u vektorima FR (realni deo) i FI (imaginarni deo). Trigonometrijske funkcije se izračunavaju uvek kada su potrebne. Zauzeće memorije za čuvanje promenljivih je $4N$, jer nije moguće *izračunavanje u mestu* (engl. in-place), tj. zamena vrednosti ulaznih vektora izračunatim vrednostima izlaznog vektora.

```
C   POTPROGRAM ZA DIREKTNO IZRACUNAVANJE DFT I IDFT.
C
C   XR     - REALNI DEO ULAZNOG VEKTORA.
C   XI     - IMAGINARNI DEO ULAZNOG VEKTORA.
C   N      - DUZINA SEKVENCI XR, XI, FR, FI.
C   FR     - REALNI DEO IZLAZNOG VEKTORA.
C   FI     - IMAGINARNI DEO IZLAZNOG VEKTORA.
C   INV = 0: DFT, INV = 1: IDFT.
C
        SUBROUTINE DFT (XR,XI,N,FR,FI,INV)
        DIMENSION XR(1),XI(1),FR(1),FI(1)
        WN = 6.283185307 / N
        IF (INV.EQ.1) WN = - WN
        DO 20  K = 1,N
           FR(K) = 0.
           FI(K) = 0.
           KM1 = K - 1
           DO 10  I = 1,N
              IM1 = I - 1
              ARG = WN * KM1 * IM1
              C = COS(ARG)
              S = SIN(ARG)
              FR(K) = FR(K) + XR(I)*C +XI(I)*S
              FI(K) = FI(K) - XR(I)*S +XI(I)*C
    10    CONTINUE
           IF (INV.EQ.1) THEN
              FR(K) = FR(K) / N
              FI(K) = FI(K) / N
           ENDIF
    20  CONTINUE
        RETURN
        END
```

Slika 5.1 FORTRAN potprogram za direktno izračunavanje DFT i IDFT.

5.3 JEDNOSTAVNE TEHNIKE ZA UBRZANJE IZRAČUNAVANJA DFT

Program za direktno izračunavanje DFT, koji je prikazan na slici 5.1, izrazito je neefikasan i može se lako poboljšati. Na primer, ukoliko je potrebno više puta izračunavati DFT sekvenci iste dužine N, kao što je to slučaj u spektralnoj analizi, onda je pogodno unapred izračunati potrebne vrednosti trigonometrijskih funkcija i smestiti ih u tabele koje se nalaze u memoriji računara. Šta više, zbog simetrije bi bilo dovoljno smestiti vrednosti samo sinusne ili samo kosinusne funkcije, i to za jednu četvrtinu periode, ali je program jednostavniji ako se zapamte sve potrebne vrednosti. Formiranje tabele trigonometrijskih funkcija i potprogram za izračunavanje DFT koristeći izračunate tabele prikazani su na slikama 5.2 i 5.3. *Tablični pristup izračunavanju trigonometrijskih funkcija* (engl. table look-up) je verovatno najbrži način za generisanje trigonometrijskih funkcija i

obavezno se primenjuje u slučajevima kada sistem za obradu signala treba da radi u realnom vremenu. Ubrzanje izračunavanja DFT plaćeno je, nažalost, povećanjem zauzeća memorije za $2N$ lokacija zbog potrebe za čuvanjem tablica tako da je ukupno potrebno $6N$ lokacija za čuvanje varijabli.

```
C   POTPROGRAM ZA FORMIRANJE TABLICA SINUSA I KOSINUSA
C
C   CT     - VEKTOR VREDNOSTI KOSINUSA.
C   ST     - VEKTOR VREDNOSTI SINUSA.
C   N      - DUZINA VEKTORA CT I ST.
C
       SUBROUTINE TRIGTAB (CT,ST,N)
       DIMENSION CT(1),ST(1)
       WN = 6.283185307 / N
       DO 10  K = 1,N
          ARG = WN * (K - 1)
          CT(K) = COS(ARG)
          ST(K) = SIN(ARG)
   10  CONTINUE
       RETURN
       END
```

Slika 5.2 Potprogram za generisanje tablice sinusa i kosinusa za direktno izračunavanje DFT.

```
C   POTPROGRAM ZA DIREKTNO IZRACUNAVANJE DFT KORISTECI
C   TABLICE SINUSA I KOSINUSA.
C
C   XR     - REALNI DEO ULAZNOG VEKTORA.
C   XI     - IMAGINARNI DEO ULAZNOG VEKTORA.
C   N      - DUZINA VEKTORA XR, XI, FR, FI, CT, ST.
C   FR     - REALNI DEO IZLAZNOG VEKTORA.
C   FI     - IMAGINARNI DEO IZLAZNOG VEKTORA.
C   CT     - VEKTOR VREDNOSTI KOSINUSA.
C   ST     - VEKTOR VREDNOSTI SINUSA.
C
       SUBROUTINE DFTTAB (XR,XI,N,FR,FI,CT,ST)
       DIMENSION XR(1),XI(1),FR(1),FI(1),CT(1),ST(1)
       DO 20  K = 1,N
          FR(K) = XR(1)
          FI(K) = XI(1)
          L = 1
          DO 10  I = 2,N
             L = L + K - 1
             IF (L.GT.N) L = L - N
             C = CT(L)
             S = ST(L)
             FR(K) = FR(K) + XR(I)*C +XI(I)*S
             FI(K) = FI(K) - XR(I)*S +XI(I)*C
   10     CONTINUE
   20  CONTINUE
       RETURN
       END
```

Slika 5.3 Potprogram za direktno izračunavanje DFT koristeći tablice sinusa i kosinusa.

U slučajevima kada se DFT izračunava samo jednom, ili kada se izračunava za različite vrednosti *N*, tablični pristup ne donosi nikakvo poboljšanje pa se trigonometrijske funkcije direktno izračunavaju. U numeričkoj analizi se za izračunavanje trigonometrijskih funkcija koriste razvoji u Tejlorov red gde broj članova zavisi od željene tačnosti. Taj pristup je primenjen i u izračunavanju sistemskih funkcija u FORTRAN-u `SIN` i `COS`. Kada se trigonometrijske funkcije izračunavaju uvek kada su potrebne, vreme potrebno za njihovo izračunavanje daleko nadmašuje vremena potrebna za dobijanje rezultata ostalih aritmetičkih operacija. Jedan efikasan način za povećanje

efikasnosti izračunavanja trigonometrijskih funkcija je rekurzivni metod. Naime, dve sukcesivne vrednosti eksponencijalnog faktora u (5.1) razlikuju se za konstantni faktor $e^{-j2\pi k/N}$, jer je:

$$e^{-j2\pi k(n+1)/N} = e^{-j2\pi kn/N} e^{-j2\pi k/N} \tag{5.2}$$

Tako se, za svaki indeks DFT odbirka, *k*, prvo izračuna konstantni faktor:

$$e^{-j2\pi k/N} = \cos(2\pi k/N) - j\sin(2\pi k/N) \tag{5.3}$$

a zatim se rekurzijom izračunavaju potrebne vrednosti sinusa i kosinusa:

$$\cos[2\pi k(n+1)/N] = \cos(2\pi kn/N)\cos(2\pi k/N) - \sin(2\pi kn/N)\sin(2\pi k/N) \tag{5.4}$$

$$\sin[2\pi k(n+1)/N] = \cos(2\pi kn/N)\sin(2\pi k/N) + \sin(2\pi kn/N)\cos(2\pi k/N) \tag{5.5}$$

Rekurzija se započinje sa početnim vrednostima $\cos(0) = 1$ i $\sin(0) = 0$. Na ovaj način se $2N^2$ izračunavanja trigonometrijskih funkcija zamenjuje sa $2N$ izračunavanja trigonometrijskih funkcija, $4N^2$ realnih množenja i $2N^2$ realnih sabiranja. Zauzeće memorije je isto kao kod direktnog postupka sa slike 5.1. Rekurzivni metod ima i jedan nedostatak koji se ogleda u akumulaciji grešaka tokom izračunavanja. Zbog toga se ovaj metod primenjuje samo na računarima opšte namene kod kojih se varijable predstavljaju u formatu sa pokretnom tačkom i velikim brojem bita u slučajevima kada dužina sekvence koja se transformiše *N* nije suviše velika.

5.4 GERCELOV ALGORITAM

Kao što se vidi iz prethodnih odeljaka, direktno izračunavanje DFT zahteva veliki broj računskih operacija čak i kada se koristi tablični ili rekurzivni pristup izračunavanju trigonometrijskih funkcija. Zbog toga će u narednim odeljcima biti razmotreni *efikasni postupci za izračunavanje svih odbiraka DFT*. Međutim, takvi postupci nisu efikasni kada je potrebno odrediti mali broj DFT odbiraka. U takvim slučajevima pogodno je upotrebiti poboljšani direktni metod koji koristi osobinu periodičnosti eksponencijalnog faktora u izrazu (5.1).

Koristeći činjenicu da je $W_N^{-kN} = e^{j2kN\pi/N} = 1$, iz (5.1) se dobija:

$$X[k] = W_N^{-kN} \sum_{m=0}^{N-1} x[m] W_N^{km} = \sum_{m=0}^{N-1} x[m] W_N^{-k(N-m)}, \quad k = 0,1,2,\ldots,N-1 \tag{5.6}$$

što ima oblik konvolucije sekvence $x[n]$ sa sekvencom $W_N^{-kn}u[n]$. Naime, ako definišemo sekvencu $y_k[n]$ kao konvoluciju konačne sekvence $x[n]$ i sekvence $W_N^{-kn}u[n]$:

$$y_k[n] = \sum_{m=0}^{N-1} x[m] W_N^{-k(n-m)} \tag{5.7}$$

onda je jasno da je:

$$X[k] = y_k[n]\big|_{n=N} = y_k[N] \tag{5.8}$$

tj. vrednost DFT na učestanosti $\omega_k = 2\pi k/N$ predstavlja odziv diskretnog sistema čiji je impulsni odziv $W_N^{-kn}u[n]$ na pobudu $x[n]$, a koji je izračunat u trenutku $n = N$.

Poznato je da je sistem čiji je impulsni odziv $W_N^{-kn}u[n]$ opisan diferencnom jednačinom:

$$y_k[n] = W_N^{-k} y_k[n-1] + x[n], \quad y_k[-1] = 0 \tag{5.9}$$

koja predstavlja alternativni način za izračunavanje $y_k[n] = X[k]$ rekurzivnim postupkom. Nažalost, ni jednačina (5.8) ni jednačina (5.9) ne smanjuju broj potrebnih računskih operacija za izračunavanje $X[k]$ u odnosu na direktni postupak izračunavanja. Međutim, izraz (5.9) se može preurediti na sledeći način. Iz (5.9) sledi:

$$y_k[n-1] = W_N^{-k} y_k[n-2] + x[n-1] \tag{5.10}$$

Množenjem obe strane jedn. (5.10) sa $-W_N^k$ i sabiranjem sa (5.9) dobija se:

$$y_k[n] = 2\cos(2\pi k/N)\, y_k[n-1] - y_k[n-2] + x[n] - W_N^k x[n-1] \tag{5.11}$$

Uvođenjem pomoćne promenljive $v_k[n]$, jednačina (5.11) se raspada na rekurzivni i nerekurzivni deo na sledeći način:

$$v_k[n] = 2\cos(2\pi k/N)\, v_k[n-1] - v_k[n-2] + x[n] \tag{5.12}$$

$$y_k[n] = v_k[n] - W_N^k v_k[n-1] \tag{5.13}$$

pri čemu su početni uslovi za pomoćnu promenljivu $v_k[n]$ jednaki nuli. Ovakvim preuređivanjem jednačine (5.9) postignuto je da se nerekurzivni deo algoritma, jedn. (5.13), izračunava samo jednom u trenutku $n = N$. U svakoj iteraciji (5.12) izvodi se jedno množenje i dva sabiranja. Ako je ulazna sekvenca $x[n]$ kompleksna, za izračunavanje jednog DFT odbirka $X[k]$, prema (5.12) i (5.13), potrebno je $2(N+2)$ realnih množenja i $4(N+1)$ realnih sabiranja, odnosno, broj množenja je smanjen približno dva puta u odnosu na direktni metod. Ako je ulazna sekvenca $x[n]$ realna, broj realnih množenja iznosi $N+1$, dok je broj realnih sabiranja $2N$. Zbog simetrije se u tom slučaju istovremeno određuje i $X[N-k]$.

```
C   POTPROGRAM ZA IZRACUNAVANJE K-TOG DFT ODBIRKA.
C   GERCELOV ALGORITAM DRUGOG REDA.
C
C   XR     - REALNI DEO ULAZNOG VEKTORA.
C   XI     - IMAGINARNI DEO ULAZNOG VEKTORA.
C   N      - DUZINA VEKTORA XR, XI.
C   FRK    - REALNI DEO K-TOG DFT ODBIRKA.
C   FIK    - IMAGINARNI DEO K-TOG DFT ODBIRKA.
C   K      - REDNI BROJ DFT ODBIRKA
C
        SUBROUTINE DFTG (XR,XI,N,FRK,FIK,K)
        DIMENSION XR(1),XI(1)
        WN = 6.283185307 / N
        VR1 = 0.
        VR2 = 0.
        VI1 = 0.
        VI2 = 0.
        ARG = WN * (K - 1)
        C = COS(ARG)
        S = SIN(ARG)
        DO 10  I = 1,N
           TEMP = VR1
           VR1 = 2.0 * C * VR1 - VR2 + XR(I)
           VR2 = TEMP
           TEMP = VI1
           VI1 = 2.0 * C * VI1 - VI2 + XI(I)
           VI2 = TEMP
    10  CONTINUE
        FRK = C * VR1 - VR2 - S * VI1
        FIK = C * VI1 - VI2 + S * VR1
        RETURN
        END
```

Slika 5.4 Potprogram za izračunavanje DFT po Gercelovom algoritmu.

Algoritam opisan jednačinama (5.12) i (5.13) poznat je u literaturi kao *Gercelov* (Goertzel) *algoritam* i približno je dva puta efikasniji od direktnog metoda. Međutim, Gercelov algoritam ima još neke prednosti. Za određivanje $X[k]$ potrebno je pamtiti samo vrednosti trigonometrijskih funkcija $\cos(2\pi k/N)$ i $\sin(2\pi k/N)$, koje se, ukoliko je potrebno više odbiraka, mogu i rekurzivno izračunavati koristeći jednačine (5.4) i (5.5). Druga prednost Gercelovog algoritma je što se rekurzivni deo izvodi stalno, a nerekurzivni deo samo po jednom na kraju algoritma. To je veoma pogodno za primene u realnom vremenu, gde se ne može čekati kraj sekvence da bi se započela njena obrada. FORTRAN potprogram za izračunavanje DFT po Gercelovom algoritmu prikazan je na slici 5.4.

Sa porastom broja DFT odbiraka koje treba izračunati, *M*, Gercelov algoritam ostaje približno dvostruko efikasniji od direktnog algoritma, ali broj operacija raste srazmerno proizvodu MN. Znači, Gercelov algoritam je pogodan za izračunavanje malog broja DFT odbiraka, tj. kada je $M << N$, ili preciznije $M < \log_2 N$. Za izračunavanje većeg broja DFT odbiraka pogodniji su FFT algoritmi kod kojih je broj operacija srazmeran sa $N \log_2 N$, a koji će biti detaljnije opisani u izlaganjima koja slede.

5.5 FFT ALGORITAM ZA $N = 2^p$ SA PREUREĐIVANJEM U VREMENU

Periodičnost i simetričnost DFT koeficijenata može biti iskorišćena u još većoj meri za redukciju broja aritmetičkih operacija ako se primeni *princip dekompozicije*. Naime, u odeljku 5.2 pokazano je da je pri direktnom izračunavanju DFT broj aritmetičkih operacija srazmeran sa N^2. Ako se ulazna sekvenca podeli na dve parcijalne sekvence, i za svaku od njih odredi DFT direktnom metodom, ukupan broj operacija množenja iznosi $2*[4*(N/2)^2] = 2N^2$, odnosno upola je manji nego za direktno izračunavanje DFT cele ulazne sekvence. Naravno, dobijeni rezultati u oba slučaja nisu identični, pa je zato potrebno izvršiti dodatne aritmetičke operacije nad parcijalnim DFT da bi se dobio ispravan rezultat. Ukoliko je broj dodatnih množenja znatno manji od $2N^2$ dobija se značajna ušteda u broju potrebnih množenja, ako je dužina sekvence velika. Princip dekompozicije se može dalje primeniti na parcijalne sekvence, sve dok se ne dođe do sekvence koja se ne može više deliti. Na principu dekompozicije su zasnovani svi tzv. brzi algoritmi za izračunavanje DFT, poznati pod opštim nazivom *Brza Furijeova Transformacija* (engl. *Fast Fourier Transform*).

5.5.1 OPIS ALGORITMA

Pretpostavimo da je dužina ulazne sekvence $N = 2^p$. Ovo ograničenje nije strogo, jer se sekvenca uvek može dopuniti potrebnim brojem nula, a znatno doprinosi jednostavnosti izvođenja i implementacije algoritma zbog toga što omogućava više uzastopnih podela sekvence na dve podsekvence iste dužine. Iz izraza (5.1) se vidi da koeficijenti kojima se množe odbirci ulaznog signala imaju jasno izraženu simetriju. Na primer, koeficijenti kojima se množe članovi ulazne sekvence sa parnim indeksima isti su za članove sa indeksima k i $k+(N/2)$, gde je $0 \le k \le (N/2)-1$. Za neparne vrednosti indeksa *k* koeficijenti imaju istu apsolutnu vrednost ali su suprotnog znaka. Zbog toga se ulazna sekvenca može razbiti na dve parcijalne sekvence: $x_{10}[n]$ koju čine elementi sa parnim indeksima i $x_{11}[n]$ koju čine elementi sa neparnim indeksima. Dakle, imamo:

$$X[k]=\sum_{n=0}^{N-1}x[n]W_N^{kn}=\sum_{n=2i}x[n]W_N^{kn}+\sum_{n=2i+1}x[n]W_N^{kn},\quad k=0,1,2,\ldots,N-1 \tag{5.14}$$

Kako je $W_N^2=e^{-j4\pi/N}=e^{-j2\pi/(N/2)}=W_{N/2}$, to je $W_N^{2ik}=W_{N/2}^{ik}$ i $W_N^{(2i+1)k}=W_N^kW_{N/2}^{ik}$, pa se iz (5.14), posle smena $x_{10}[i]=x[2i]$ i $x_{11}[i]=x[2i+1]$, dobija:

$$X[k]=\sum_{i=0}^{N/2-1}x_{10}[i]W_{N/2}^{ki}+W_N^k\sum_{i=0}^{N/2-1}x_{11}[i]W_{N/2}^{ki},\quad k=0,1,2,\ldots,N/2-1 \tag{5.15}$$

U poslednjem izrazu pojavljuju se dve sume koje predstavljaju DFT sekvenci $x_{10}[n]$ i $x_{11}[n]$ čija je dužina $N/2$ članova. Dakle, imamo:

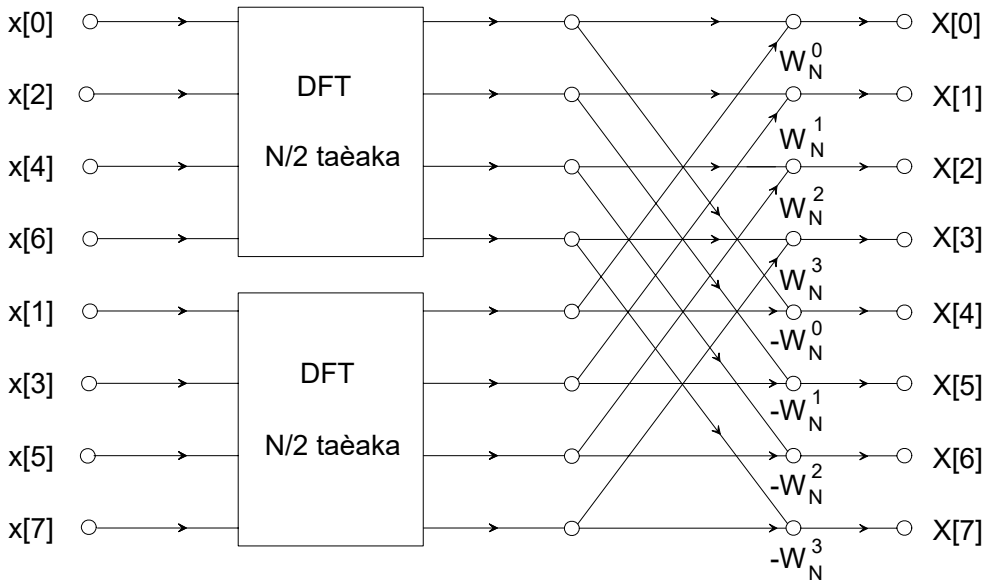
$$X[k]=X_{10}[k]+W_N^kX_{11}[k],\quad k=0,1,2,\ldots,N/2-1 \tag{5.16}$$

Za vrednosti indeksa koje su veće od $N/2$, umesto jednačine (5.16) važi:

$$X[k+N/2]=X_{10}[k+N/2]+W_N^{k+N/2}X_{11}[k+N/2],\quad k=0,1,2,\ldots,N/2-1 \tag{5.17}$$

odnosno, zbog periodičnosti funkcija $X_{10}[k]$ i $X_{11}[k]$ sa periodom $N/2$ i činjenice da je $W_N^{k+N/2}=-W_N^k$:

$$X[k+N/2]=X_{10}[k]-W_N^kX_{11}[k],\quad k=0,1,2,\ldots,N/2-1 \tag{5.18}$$



Slika 5.5 Prvi korak u razvoju FFT algoritma sa preuređivanjem ulazne sekvence.

Iz jednačina (5.16) i (5.18) može se uočiti da se izrazi za izračunavanje DFT odbiraka čiji su indeksi k i $k+N/2$ razlikuju samo po znaku drugog sabirka. Način izračunavanja DFT odbiraka $X[k]$ i $X[k+N/2]$ na osnovu $X_{10}[k]$ i $X_{11}[k]$ naziva se *leptir operacija* (engl. *butterfly*) i može se grafički prikazati kao na slici 5.5. Uočava se da *u leptir operaciji postoji samo jedno kompleksno množenje i 2 kompleksna sabiranja*, odnosno *4 realna množenja i 6 realnih sabiranja*. Množenje sa eksponencijalnim faktorom W_N^k menja samo argument kompleksnog broja $X_{11}[k]$, pa se W_N^k naziva *rotacioni faktor*. U leptiru se takođe izračunavaju i dve trigonometrijske funkcije.

Pošto je za izračunavanje svih DFT odbiraka sekvence od $N/2$ tačaka potrebno $4(N/2)^2=N^2$ množenja, ukupno je potrebno $2N^2$ množenja za izračunavanje sekvenci $X_{10}[k]$ i $X_{11}[k]$ i dodatnih $4(N/2)=2N$ množenja za kombinovanje sekvenci $X_{10}[k]$ i $X_{11}[k]$.

Dakle, za izračunavanje DFT podelom sekvence na parni i neparni deo ukupno je potrebno $2N(N+1)$ realnih množenja, što je manje od $4N^2$ ako je $N>2$. Pošto je po pretpostavci dužina ulazne sekvence $N=2^p$, sekvence $x_{10}[n]$ i $x_{11}[n]$ imaju paran broj elemenata pa se mogu, u cilju daljeg smanjenja broja operacija, ponovo podeliti na parcijalne sekvence dužine $N/4$ po osnovu parnosti indeksa. Tako se dobija:

$$X_{10}[k]=\sum_{n=0}^{N/4-1}x_{10}[2n]W_{N/2}^{2kn}+\sum_{n=0}^{N/4-1}x_{10}[2n+1]W_{N/2}^{k(2n+1)}=$$

$$=\sum_{n=0}^{N/4-1}x_{20}[n]W_{N/4}^{kn}+W_N^{2k}\sum_{n=0}^{N/4-1}x_{21}[n]W_{N/4}^{kn},\quad k=0,1,2,\ldots,N/4-1 \tag{5.19}$$

$$X_{11}[k]=\sum_{n=0}^{N/4-1}x_{22}[n]W_{N/4}^{kn}+W_N^{2k}\sum_{n=0}^{N/4-1}x_{23}[n]W_{N/4}^{kn},\quad k=0,1,2,\ldots,N/4-1 \tag{5.20}$$

gde je:

$$\begin{aligned}&x_{20}[n]=x_{10}[2n],\quad x_{21}[n]=x_{10}[2n+1]\\&x_{22}[n]=x_{11}[2n],\quad x_{23}[n]=x_{11}[2n+1]\end{aligned} \tag{5.21}$$

i $n=0,1,2,\ldots,N/4-1$. Iz jednačina (5.19) i (5. 20) takođe sledi:

$$\begin{aligned}&X_{10}[k]=X_{20}[k]+W_N^{2k}X_{21}[k],\quad X_{10}[k+N/4]=X_{20}[k]-W_N^{2k}X_{21}[k]\\&X_{11}[k]=X_{22}[k]+W_N^{2k}X_{23}[k],\quad X_{11}[k+N/4]=X_{22}[k]-W_N^{2k}X_{23}[k]\end{aligned} \tag{5.22}$$

za $k=0,1,2,\ldots,N/4-1$.

Podelom ulazne sekvence na četiri parcijalne sekvence dalje se smanjuje broj potrebnih aritmetičkih operacija. Pošto je $N=2^p$, postupak podele se može nastaviti. U opštem slučaju, posle *m* koraka dekompozicije ulazna sekvenca podeljena je na parcijalne sekvence:

$$\begin{aligned}&x_{m0}[n]=x_{(m-1)0}[2n]\\&x_{m1}[n]=x_{(m-1)0}[2n+1]\\&x_{m2}[n]=x_{(m-1)1}[2n]\\&x_{m3}[n]=x_{(m-1)1}[2n+1]\\&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\end{aligned} \tag{5.23}$$

gde je $n=0,1,\ldots,N/2^m-1$. U frekvencijskom domenu se dobija:

$$\begin{aligned}&X_{(m-1)0}[k]=X_{m0}[k]+W_N^{2^{m-1}k}X_{m1}[k]\\&X_{(m-1)0}[k+N/2^m]=X_{m0}[k]-W_N^{2^{m-1}k}X_{m1}[k]\\&X_{(m-1)1}[k]=X_{m2}[k]+W_N^{2^{m-1}k}X_{m3}[k]\\&X_{(m-1)1}[k+N/2^m]=X_{m2}[k]-W_N^{2^{m-1}k}X_{m3}[k]\\&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\end{aligned} \tag{5.24}$$

gde je $k=0,1,\ldots,N/2^m-1$.

Postupak dekompozicije može se izvršiti p puta, tj. sve dok se ne dobiju parcijalne sekvence od samo jednog elementa, čija je DFT jednaka samom elementu. Dakle, u poslednjem koraku dekompozicije se ima:

$$X_{pi}[0] = x_{pi}[0], \quad i = 0,1,\ldots,N-1 \tag{5.25}$$

Kompletan proces izračunavanja DFT za ulaznu sekvencu od 8 elemenata prikazan je na slici 5.6. Sa slike se vidi da ukupno postoji $p = \log_2 N$ stepeni u kojima se rastavljaju sekvence na dve podsekvence. U svakom stepenu postoji $N/2$ leptir operacija, tako da je ukupan broj leptir operacija:

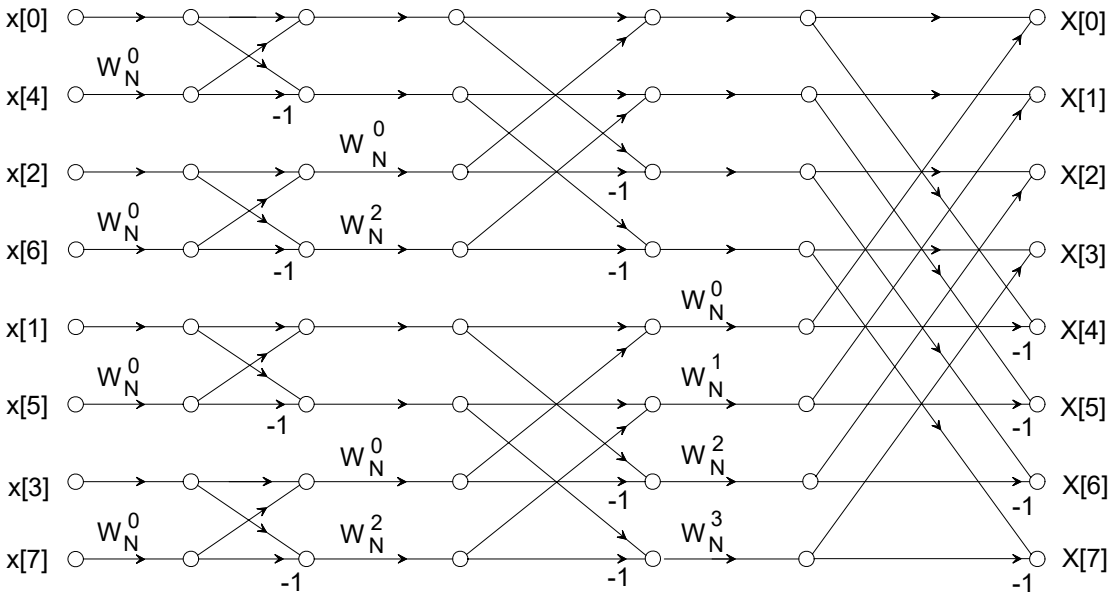
$$N_B = \frac{N}{2}\log_2 N \tag{5.26}$$

a ukupan broj realnih množenja, sabiranja i izračunavanja trigonometrijskih funkcija je:

$$N_M = 2N\log_2 N \tag{5.27}$$

$$N_A = 3N\log_2 N \tag{5.28}$$

$$N_T = N\log_2 N \tag{5.29}$$



Slika 5.6 Kompletan dijagram toka FFT algoritma sa preuređivanjem ulazne sekvence za N = 8.

Dakle, broj potrebnih operacija množenja, sabiranja i izračunavanja trigonometrijskih funkcija je znatno smanjen u odnosu na direktni metod izračunavanja prema izrazu (5.1). Poređenje broja operacija u direktnom i brzom izračunavanju DFT prikazano je u Tabeli 5.1. Na primer, ako ulazna sekvenca ima 512 elemenata, broj operacija množenja u opisanom algoritmu iznosi svega 0.88% od broja množenja u direktnom metodu izračunavanja. Slična ušteda se postiže i za operacije sabiranja i izračunavanja trigonometrijskih funkcija. Zbog toga je opisani algoritam tipičan predstavnik algoritama za efikasno izračunavanje DFT poznatih pod nazivom *Brza Furijeova Transformacija* (*Fast Fourier Transform - FFT*).

Tabela 5.1 Broj aritmetičkih operacija kod DFT i FFT algoritama.

		DFT			DIT FFT		
p	N	N_T	N_M	N_A	N_T	N_M	N_A
2	4	32	64	48	8	16	24
3	8	128	256	224	24	48	72
4	16	512	1024	960	64	128	192
5	32	2048	4096	3968	160	320	480
6	64	8192	16384	16128	384	768	1152
7	128	32768	65536	65024	896	1792	2688
8	256	131072	262144	261120	2048	4096	6144
9	512	524288	1048576	1046528	4608	9216	13824
10	1024	2097152	4194304	4190208	10240	20480	30720
11	2048	8388608	16777216	16769024	22528	45056	67584
12	4096	33554432	67108864	67092480	49152	98304	147456

5.5.2 PREUREĐIVANJE ULAZNIH PODATAKA

Sa slike 5.6 se može uočiti da je izmenjen redosled ulaznih podataka što je posledica niza uzastopnih podela sekvenci na dva dela po osnovu parnosti indeksa. Proces preuređivanja odbiraka ulazne sekvence dužine $N = 2^p$ se u opštem slučaju vrši na osnovu jednačine:

$$x_{br}[j] = x[i] \tag{5.30}$$

gde su j i i indeksi preuređene i originalne ulazne sekvence. Proces preuređivanja obavlja se na sledeći način. Prvo se decimalne vrednosti indeksa i napišu u binarnom obliku sa minimalno potrebnim brojem bita $p = \log_2 N$. Zatim se binarne sekvence brojeva koje predstavljaju indekse invertuju i ponovo konvertuju u decimalni oblik. Opisana procedura preuređivanja ulazne sekvence naziva se *bit inverzija* (engl. *bit reversal*). Postupak inverzije bita i preuređivanja indeksa u slučaju $N = 8$ ($p = 3$) prikazan je u Tabeli 5.2.

Tabela 5.2 Formiranje indeksa preuređene sekvence.

Indeks ulazne sekvence		Indeks preuređene sekvence	
Decimalni	Binarni	Binarni	Decimalni
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Pošto se u opisanom algoritmu preuređuje ulazna sekvenca, tj, preuređivanje se vrši u vremenskom domenu, ovakav FFT algoritam naziva se *FFT algoritam sa preuređivanjem u vremenu*. U literaturi na engleskom jeziku ovaj algoritam naziva *decimation-in-time* (*DIT*) *algoritam*, pa se i u našoj terminologiji odomaćio naziv *decimacija* (*desetkovanje*) *u vremenu*. Kako se u poslednje vreme u literaturi iz obrade signala pod decimacijom uglavnom podrazumeva redukcija broja odbiraka, pravilnije je i pogodnije koristiti naziv algoritam sa preuređivanjem u vremenu, ili kraće, DIT algoritam.

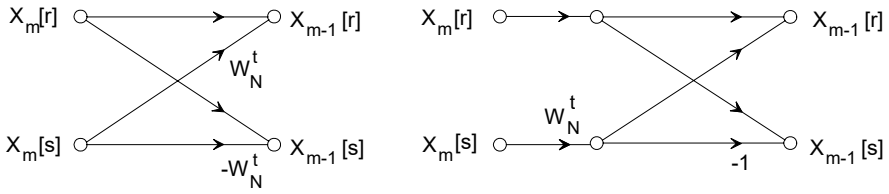
5.5.3 POTREBNA MEMORIJA ZA IZRAČUNAVANJE DFT

U programskoj realizaciji direktnog algoritma za izračunavanje DFT (slike 5.1 i 5.3) za smeštanje ulaznih i izlaznih podataka bili su potrebni posebni vektori. Dakle, potreban memorijski prostor za smeštanje podataka iznosio je $4N$, gde je N dužina ulazne kompleksne sekvence. Međutim, opisani FFT algoritam sa preuređivanjem u vremenu ima jednu interesantnu osobinu. Da bi je ispitali, posmatrajmo leptir operaciju u m-tom stepenu dekompozicije koja je posebno prikazana dijagramom toka na slici 5.7, a opisana je jednačinama:

$$X_{m-1}[r] = X_m[r] + W_N^t X_m[s] \tag{5.31}$$

$$X_{m-1}[s] = X_m[r] - W_N^t X_m[s] \tag{5.32}$$

gde je $s = r + N/2^m$ i $t = -2^{m-1}k,\ k = 0,1,\ldots, N/2^m - 1$.



Slika 5.7 Dijagram toka leptir operacije FFT algoritma sa preuređivanjem u vremenu.

Dakle, u bilo kom stepenu izvršavanja algoritma, podaci čiji su indeksi r i s potrebni su za izračunavanje samo dva nova podatka na istim lokacijama. Ako se izračunavanje obavlja kolonu po kolonu sa slike 5.6, $2N$ memorijskih lokacija u kojima se nalazi ulazna sekvenca može se koristiti za smeštanje međurezultata i izlazne DFT sekvence. Ustvari, potrebna je još samo jedna dodatna memorijska lokacija za smeštanje međurezultata kod izvršavanja svake od leptir operacija. Ovakav efikasan postupak za korišćenje memorije naziva se *izračunavanje u mestu* (engl. *in-place computation*) i veoma je važan kada je memorijski prostor ograničen.

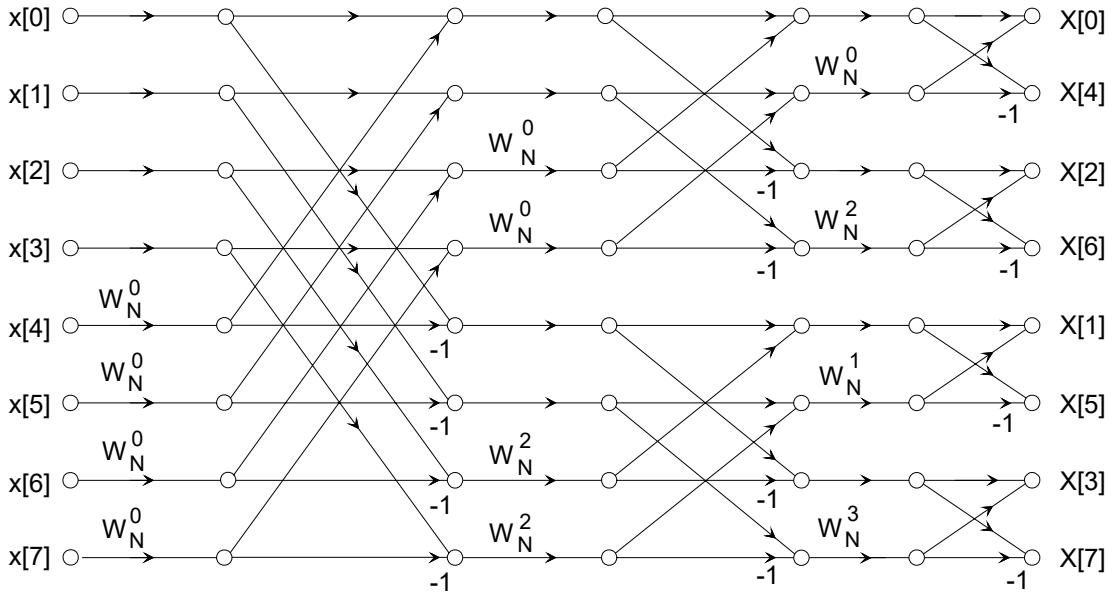
U primenama gde se ne može tolerisati izmena redosleda elemenata ulazne sekvence, opisani algoritam se može restrukturirati tako da i ulazna i izlazna sekvenca budu u prirodnom redosledu. Takvi algoritmi (engl. *not-in-place*) su brži, jer se ne vrši operacija preuređivanja, ali zahtevaju dodatnih $2N$ memorijskih lokacija za smeštanje međurezultata i izlaznih podataka. Međutim, struktura takvih algoritama je komplikovanija, pa se zbog toga retko koriste.

5.5.4 MODIFIKACIJE OSNOVNOG DIT ALGORITMA

U novijoj literaturi o digitalnoj obradi signala opisan je veći broj različitih algoritama za efikasno izračunavanje DFT, a izvršena je i njihova klasifikacija. U takvoj klasifikaciji se u klasu

algoritama sa preuređivanjem u vremenu svrstavaju svi algoritmi kod kojih se *prilikom izvođenja algoritma koristi dekompozicija ulazne sekvence*, odnosno leptir kao na slici 5.7, bez obzira na to da li su na kraju ulazni podaci u prirodnom ili bit inverznom redosledu.

Ako se u dijagramu toka koji je prikazan na slici 5.6 izvrši preuređivanje čvorova, rezultat izračunavanja se neće promeniti ako se pri pomeranju svakog čvora pomeraju i grane koje izlaze ili ulaze u njega. Jedina promena nastaje u načinu smeštanja ulaznih i izlaznih podataka, kao i međurezultata. Da bi se očuvalo izračunavanje u mestu potrebno je da ulazni i izlazni čvorovi svake leptir operacije leže na istoj horizontali. Na primer, preuređivanjem dijagrama toka sa slike 5.6 može se dobiti dijagram toka prikazan na slici 5.8 u kome je ulazna sekvenca u prirodnom, a izlazna u bit inverznom redosledu. Dijagram toka na slici 5.8 i dalje predstavlja algoritam sa preuređivanjem u vremenu [C-38], jer su leptir operacije na slikama 5.6 i 5.8 identične.

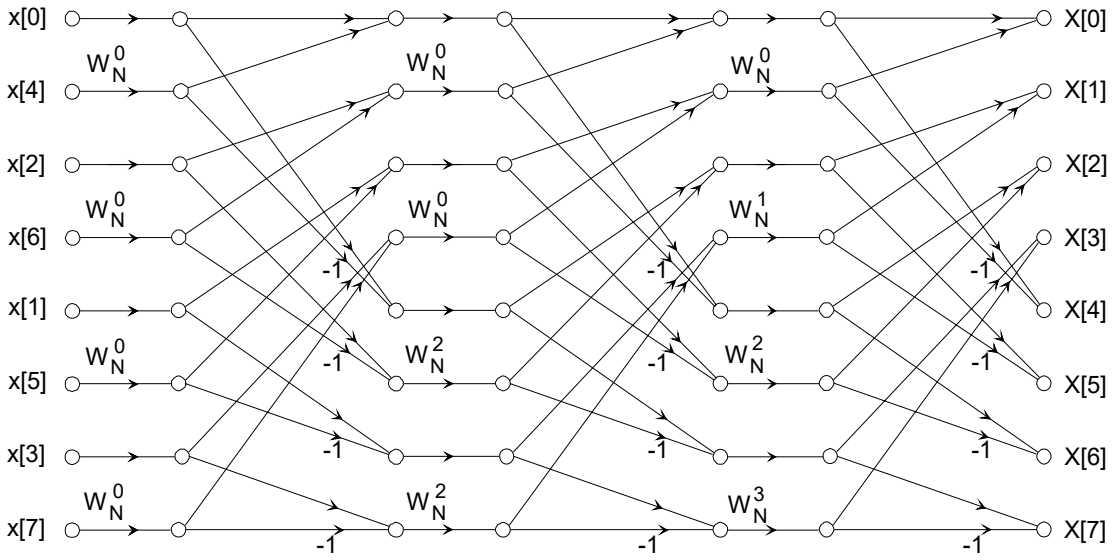


Slika 5.8 Dijagram toka DIT algoritma sa ulaznom sekvencom u normalnom i izlaznom u bit inverznom redosledu

Preuređivanjem na opisani način može se dobiti veliki broj različitih alternativnih verzija algoritma sa preuređivanjem u vremenu. Većina takvih struktura nema nikakav praktični značaj zbog toga što je redosled elemenata ulazne ili izlazne sekvence neregularan, ili zbog toga što se izračunavanje ne vrši u mestu. Na primer, može se napraviti struktura kod koje su obe sekvence u normalnom redosledu, ali je, nažalost, za izračunavanje u svakom koraku potrebno dodatno zauzeće memorije za 2 vektora od N elemenata.

Još jedna alternativna struktura koja ima korisne primene, opisana u [S-9], prikazana je na slici 5.9. Sa slike 5.9 se uočava da svi leptirovi u dijagramu toka imaju istu geometriju. Ulazni podaci u leptir imaju susedne memorijske lokacije, dok se indeksi izlaznih podataka iz leptira uvek razlikuju za $N/2$. Zbog toga je optimalna organizacija podataka u memoriji takva da su ulazni i izlazni podaci smešteni u četiri kompleksna vektora (osam realnih) dužine $N/2$. Na početku se jedna polovina ulaznih podataka u bit inverznom redosledu nalazi u vektoru 1, dok se druga nalazi u vektoru 2. Izlazni rezultati iz leptirova se naizmenično smeštaju u vektore 3 i 4. U sledećem stepenu izračunavanja vektori 3 i 4 predstavljaju ulaze leptirova, dok se vektori 1 i 2 koriste za smeštaj rezultata izračunavanja. Ovakva struktura je pogodna zbog toga što se ulazni podaci

uzimaju u račun na sekvencijalni način. Takav pristup podacima je od posebnog interesa kada se obrađuju dugačke sekvence koje ne mogu da se smeste u operativnu memoriju računara.



Slika 5.9 Dijagram toka DIT algoritma sa fiksnom geometrijom leptirova.

5.6 FFT ALGORITAM $N = 2^p$ SA PREUREĐIVANJEM U FREKVENCIJSKOM DOMENU

Algoritmi za brzo izračunavanje DFT koji su zasnovani na dekompoziciji izlazne sekvence poznati su pod opštim nazivom *algoritmi sa preuređivanjem u frekvencijskom domenu* (engl. *decimation-in-frequency* - *DIF*). U izvođenju osnovnog DIF algoritma obično se polazi od sekvence dužine $N = 2^p$ jer je tada proces dekompozicije najjednostavniji. Proces dekompozicije izlazne sekvence vrši se njenom podelom prema parnosti indeksa DFT odbiraka. Odbirci sa parnim indeksima dati su izrazom:

$$X[2k] = \sum_{n=0}^{N-1} x[n] W_N^{2kn} = \sum_{n=0}^{N/2-1} x[n] W_N^{2kn} + \sum_{n=N/2}^{N-1} x[n] W_N^{2kn} =$$

$$= \sum_{n=0}^{N/2-1} x[n] W_N^{2kn} + \sum_{n=0}^{N/2-1} x[n+N/2] W_N^{2k(n+N/2)}, \quad k = 0,1,2,\ldots,N/2-1 \tag{5.33}$$

Zbog periodičnosti funkcije W_N^n, kao i zbog jednakosti $W_N^2 = W_{N/2}$, ima se:

$$W_N^{2k(n+N/2)} = W_N^{2kn} W_N^{kN} = W_N^{2kn} = W_{N/2}^{kn} \tag{5.34}$$

odnosno, iz (5.24) se dobija:

$$X[2k] = \sum_{n=0}^{N/2-1} \big(x[n] + x[n+N/2]\big) W_{N/2}^{kn}, \quad k = 0,1,2,\ldots,N/2-1 \tag{5.35}$$

Na sličan način se za izlazne odbirke sa neparnim indeksima dobija:

$$X[2k+1] = \sum_{n=0}^{N-1} x[n] W_N^{n(2k+1)} = \sum_{n=0}^{N/2-1} \big(x[n] - x[n+N/2]\big) W_N^n W_{N/2}^{kn}, \quad k = 0,1,2,\ldots,N/2-1 \tag{5.36}$$

Jednačinama (5.35) i (5.36) može se dati odgovarajuća interpretacija. Na primer, suma u jednačini (5.35) predstavlja DFT sekvence $x_{10}[n]$ od $N/2$ članova koja se dobija sabiranjem odgovarajućih članova prve i druge polovine ulazne sekvence. Interpretacija sume u izrazu (5.36) nešto je komplikovanija. Naime, to je ponovo DFT sekvence $x_{20}[n]$ od $N/2$ članova, koja se dobija oduzimanjem odgovarajućih članova iz druge polovine od članova prve polovine ulazne sekvence i množenjem dobijene razlike sa W_N^n. Dakle, iz (5.35) i (5.36) sledi:

$$X[2k] = \sum_{n=0}^{N/2-1} x_{10}[n] W_{N/2}^{kn}, \quad k = 0,1,2,\ldots, N/2-1 \tag{5.37}$$

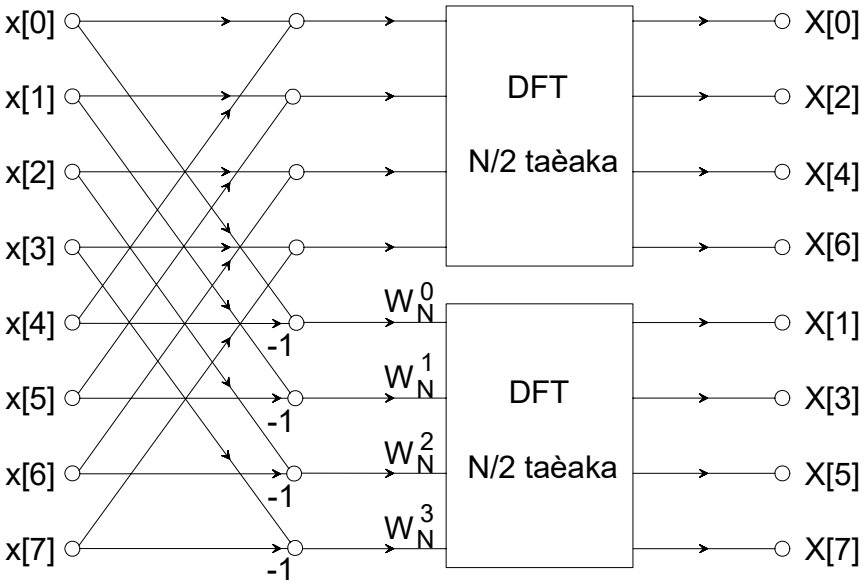
$$X[2k+1] = \sum_{n=0}^{N/2-1} x_{11}[n] W_{N/2}^{kn}, \quad k = 0,1,2,\ldots, N/2-1 \tag{5.38}$$

gde je:

$$x_{10}[n] = x[n] + x[n+N/2] \tag{5.39}$$

$$x_{11}[n] = \left(x[n] - x[n+N/2]\right) W_N^n \tag{5.40}$$

Opisani postupak dekompozicije za slučaj $N = 8$ se može predstaviti dijagramom toka na slici 5.10.



Slika 5.10 Prvi korak u razvoju FFT algoritma sa preuređivanjem u frekvencijskom domenu.

Kao i u slučaju DIT algoritma, proces dekompozicije se može dalje nastaviti, jer je i $N/2$ paran broj zbog uslova $N = 2^p$. Posle *m* koraka dekompozicije, dobija se:

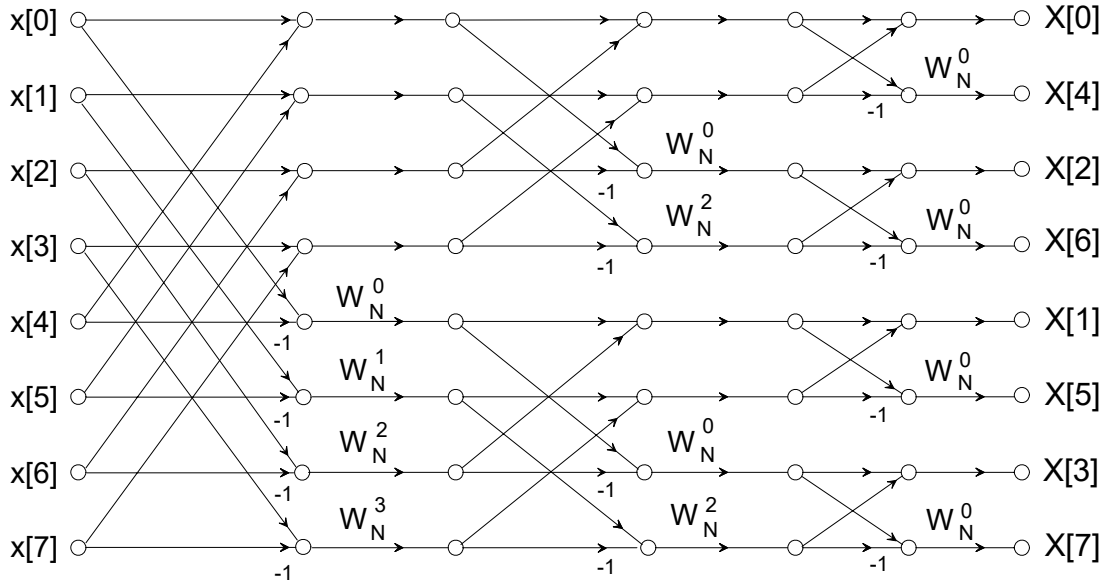
$$\begin{aligned} X[2^m k] &= \sum_{n=0}^{N/2^m-1} x_{m0}[n] W_{N/2^m}^{kn} \\ X[2^m k + 2^{m-1}] &= \sum_{n=0}^{N/2^m-1} x_{m1}[n] W_{N/2^m}^{kn} \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \end{aligned} \tag{5.41}$$

gde je $k = 0,1,\ldots, N/2^m - 1$, a takođe:

$$x_{m0}[n] = x_{(m-1)0}[n] + x_{(m-1)0}[n + N/2^m]$$
$$x_{m1}[n] = \left(x_{(m-1)0}[n] - x_{(m-1)0}[n + N/2^m]\right)W_N^{2^{m-1}n} \qquad (5.42)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

gde je $n = 0,1,\ldots, N/2^m - 1.$

Postupak dekompozicije se nastavlja sve dok se ne dobiju parcijalne sekvence od samo jednog elementa, čija je DFT jednaka samim elementima. Kompletan dijagram toka DIF algoritma za $N = 8$ prikazan je na slici 5.11. Sa slike se vidi da su ulazni podaci u prirodnom, a izlazni u bit inverznom redosledu.



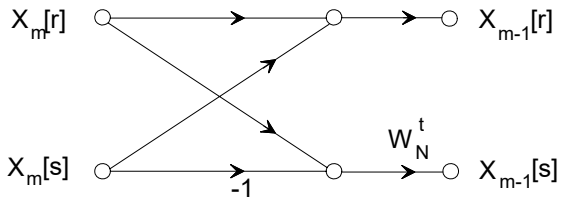
Slika 5.11 Dijagram toka DIF algoritma za *N* = 8.

Lako se može uočiti da se i kod DIF algoritma izračunavanje sastoji od izračunavanja $N/2$ leptira čija je struktura prikazana na slici 5.12. I ova konfiguracija leptira, koja je u opštem slučaju opisana jednačinama:

$$X_{m-1}[r] = X_m[r] + X_m[s] \qquad (5.43)$$

$$X_{m-1}[s] = \left(X_m[r] - X_m[s]\right)W_N^t \qquad (5.44)$$

gde je $s = r + N/2^m$ i $t = -2^{m-1}k$, $k = 0,1,\ldots, N/2^m - 1$, zahteva izračunavanje dve trigonometrijske funkcije, jedno kompleksno (četiri realna) množenje i dva kompleksna (šest realnih) sabiranja. Dakle broj aritmetičkih operacija je isti kao kod DIT algoritama i dat je jednačinama (5.21), (5.22) i (5.23).



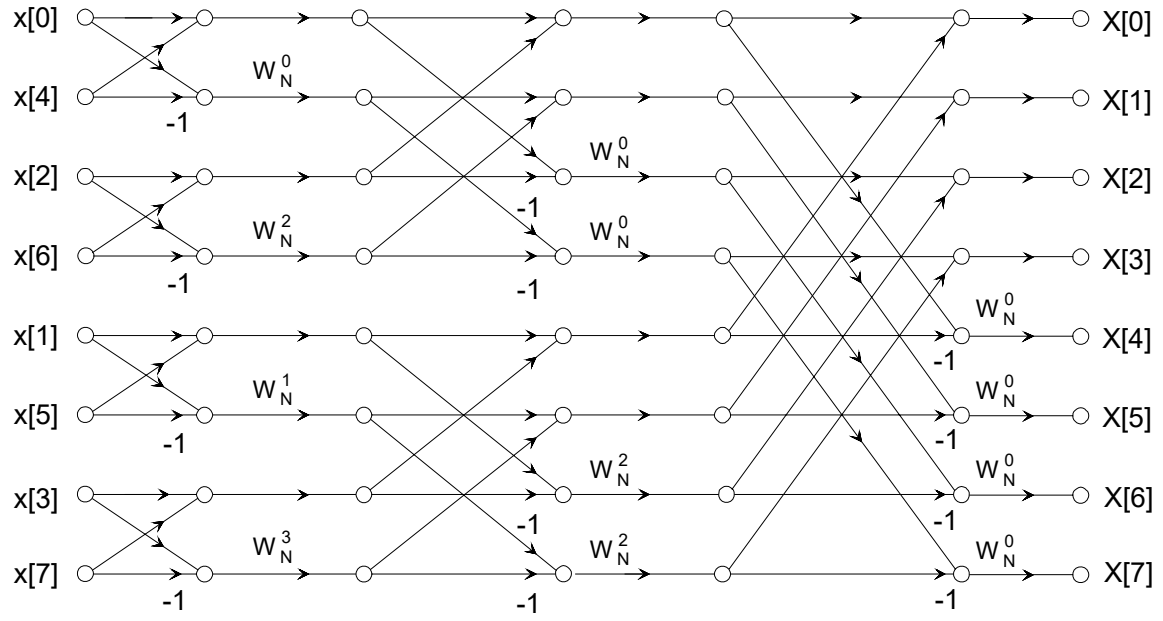
Slika 5.12 Dijagram toka leptir operacije kod DIF algoritma.

Leptir za DIF algoritam takođe poseduje osobinu izračunavanja u mestu, jer se ulazni podaci koriste samo u okviru jednog leptira. Dakle, i zauzeće memorije je identično kao kod DIT

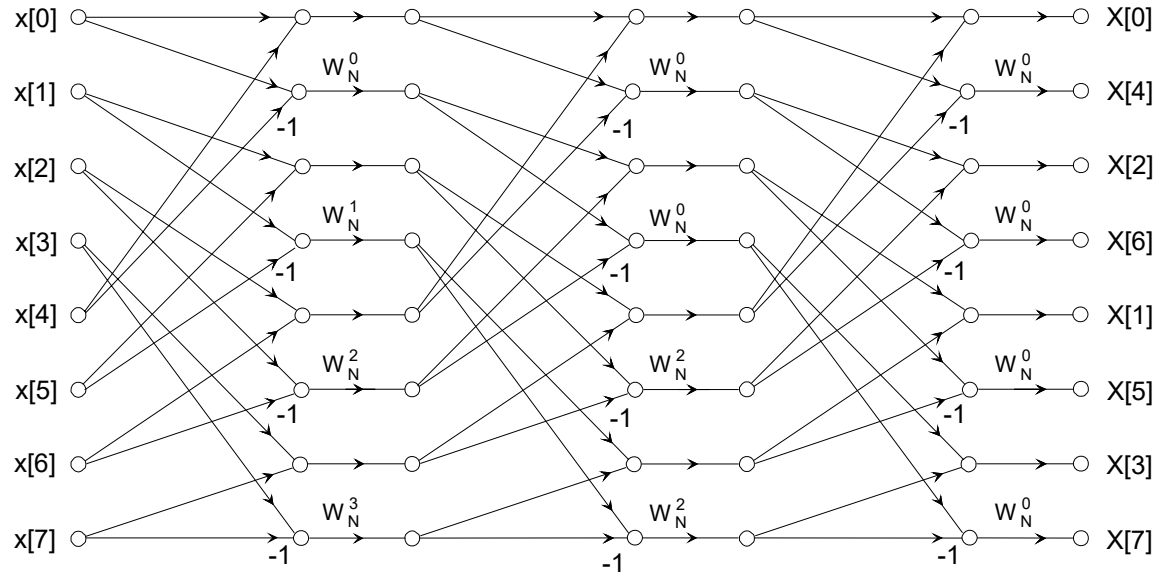
algoritama i iznosi $2N$ memorijskih lokacija za ulazno-izlazne podatke i jednu memorijsku lokaciju za međurezultate.

Na prvi pogled, glavna razlika između osnovnih verzija DIT i DIF algoritma je u uređenju ulaznih i izlaznih podataka. Međutim, kako se uređenje podataka u slučaju DIT algoritma može menjati, kako je to pokazano na slikama 5.6, 5.8 i 5.9, jasno je da takav zaključak ne važi. *Osnovna razlika između DIT i DIF algoritama je u konstrukciji leptira*. Nažalost, ova činjenica se relativno retko ističe u literaturi iz digitalne obrade signala, gde uglavnom preovlađuje nekorektno objašnjenje zasnovano na uređenju ulaznih ili izlaznih podataka.

Na sličan način kao u prethodnom odeljku, transformacijama dijagrama toka može se doći do alternativnih formi DIF algoritma. Kao primer, na slici 5.13 je prikazana verzija DIF algoritma kod koje su ulazni podaci u bit inverznom a izlazni podaci u normalnom redosledu, dok je na slici 5.14 prikazana verzija DIF algoritma sa konstantnom geometrijom leptira.



Slika 5.13 Dijagram toka DIF algoritma sa ulaznom sekvencom u bit inverznom i izlaznom sekvencom u normalnom redosledu.



Slika 5.14 Dijagram toka DIF algoritma sa leptirima konstantne geometrije.

5.7 PROGRAMSKA IMPLEMENTACIJA FFT ALGORITAMA

5.7.1 OSNOVNA PROGRAMSKA IMPLEMENTACIJA DIT ALGORITMA

Opisani FFT algoritam sa preuređivanjem u vremenu lako je pretočiti u programsku implementaciju u nekom višem programskom jeziku. Kao primer, na slici 5.15 prikazana je osnovna verzija programa napisana u programskom jeziku FORTRAN. Prikazani program je demonstracionog karaktera jer se trigonometrijske funkcije izračunavaju svaki put kada su potrebne, što se, kao što će biti kasnije objašnjeno, izbegava u efikasnim realizacijama.

```
C
C   BRZA FURIJEOVA TRANSFORMACIJA, DIT ALGORITAM
C
C   XR       - REALNI DEO VEKTORA PODATAKA.
C   XI       - IMAGINARNI DEO VEKTORA PODATAKA.
C   M   = LOG2(N)
C
        SUBROUTINE DITFFT(XR,XI,M)
        DIMENSION XR(1), XI(1)
        N=2**M
        DO 30  K = 1,M
           LEP2 = 2**K
           PHI  =  6.283185307 / LEP2
           LEP = LEP2 / 2
           DO 20  J = 1,LEP
              ARG = (J-1)*PHI
              C = COS(ARG)
              S = -SIN(ARG)
              DO 10  I = J,N,LEP2
                 IP = I + LEP
                 TR = XR(IP)*C - XI(IP)*S
                 TI = XR(IP)*S + XI(IP)*C
                 XR(IP) = XR(I) - TR
                 XI(IP) = XI(I) - TI
                 XR(I) = XR(I) + TR
                 XI(I) = XI(I) + TI
   10         CONTINUE
   20      CONTINUE
   30   CONTINUE
        RETURN
        END
```

Slika 5.15 FORTRAN implementacija DIT algoritma.

Kao što se vidi, struktura programa je izuzetno jednostavna. Pretpostavljeno je da se ulazni podaci već nalaze u bit inverznom redosledu. U spoljnjoj petlji (`DO 30`), koja se izvršava $M = \log_2 N$ puta, bira se stepen iz dijagrama toka sa slike 5.8 i određuje vrednost inkrementa faznog ugla prilikom izračunavanja rotacionih faktora u unutrašnjim petljama. U drugoj petlji (`DO 20`), koja se izvršava `2**`K puta, gde je K redni broj stepena izračunavanja, u svakom prolazu izračunaće se vrednosti onih leptirova koji imaju isti rotacioni faktor. U unutrašnjoj petlji (`DO 10`) izvršava se sama leptir operacija sa već pripremljenim podacima za vrednosti trigonometrijskih funkcija.

5.7.2 OSNOVNA PROGRAMSKA IMPLEMENTACIJA DIF ALGORITMA

Kao što je to bio slučaj sa DIT algoritmom, i osnovni DIF algoritam se može lako programirati u nekom višem programskom jeziku. Kao primer, na slici 5.16 prikazana je osnovna verzija programa napisana u programskom jeziku FORTRAN.

```
C
C   BRZA FURIJEOVA TRANSFORMACIJA, DIF ALGORITAM
C
C   XR    - REALNI DEO VEKTORA PODATAKA.
C   XI    - IMAGINARNI DEO VEKTORA PODATAKA.
C   M     = LOG2(N)
C
      SUBROUTINE DIFFFT(XR,XI,M)
      DIMENSION XR(1), XI(1)
      N=2**M
      DO 30  K = 1,M
         LEP2 = 2**(M+1-K)
         PHI = 6.283185307 / LEP2
         LEP = LEP2 / 2
         DO 20  J = 1,LEP
            ARG = (J-1)*PHI
            C = COS(ARG)
            S = -SIN(ARG)
            DO 10  I = J,N,LEP2
               IP = I + LEP
               TR = XR(I) - XR(IP)
               TI = XI(I) - XI(IP)
               XR(I) = XR(I) + XR(IP)
               XI(I) = XI(I) + XI(IP)
               XR(IP) = TR*C - TI*S
               XI(IP) = TR*S + TI*C
   10       CONTINUE
   20    CONTINUE
   30 CONTINUE
      RETURN
      END
```

Slika 5.16 FORTRAN implementacija DIF algoritma.

```
C
C   PREUREDJIVANJE VEKTORA PO BIT INVERZNOM REDOSLEDU INDEKSA
C
C   XR    - REALNI DEO VEKTORA PODATAKA.
C   XI    - IMAGINARNI DEO VEKTORA PODATAKA.
C   M     = LOG2(N)
C
      SUBROUTINE BITREV (XR,XI,M)
      DIMENSION XR(1), XI(1)
      N=2**M
      N2 = N / 2
      LAST = N - 1
      J = 1
C   BROJAC PO I BROJI U PRIRODNOM REDOSLEDU
      DO 10  I = 1,LAST
         IF (I.LT.J) THEN
            T = XR(I)
            XR(I) = XR(J)
            XR(J) = T
            T = XI(I)
            XI(I) = XI(J)
            XI(J) = T
         ENDIF
         K = N2
C   BROJAC PO J BROJI U BIT INVERZNOM REDOSLEDU
         DOWHILE (K.LT.J)
            J = J - K
            K = K / 2
         ENDDO
         J = J + K
   10 CONTINUE
RETURN
END
```

Slika 5.17 Potprogram za realizaciju preuređivanja podataka bit inverzijom indeksa.

5.7.3 IMPLEMENTACIJA ALGORITMA ZA PREUREĐIVANJE PODATAKA

Kao što je već rečeno u prethodnim odeljcima, svi algoritmi kod kojih se izračunavanje obavlja u mestu zahtevaju preuređivanje ulaznih ili izlaznih podataka. Kako pri formiranju sekvence sa indeksima u bit inverznom redosledu uvek po dva elementa međusobno zamenjuju mesta, zamena mesta elemenata u sekvenci se vrši ako su zadovoljeni sledeći uslovi:

$$i = br[j], \quad i \neq j, \quad i < j \tag{5.45}$$

```
C
C   PREUREDJIVANJE VEKTORA PO BIT INVERZNOM REDOSLEDU INDEKSA
C
C   XR     - REALNI DEO VEKTORA PODATAKA.
C   XI     - IMAGINARNI DEO VEKTORA PODATAKA.
C   M      = LOG2(N)
C
        SUBROUTINE BITREV (XR,XI,M)
        DIMENSION XR(1), XI(1)
        N2=2**(M-1)
        N4 = N2 / 2
        N21 = N2 + 1
        LAST = N2 - 1
        J = 1
C   BROJAC PO I BROJI U PRIRODNOM REDOSLEDU
        DO 10  I = 1,LAST,2
           IF (I.LT.J) THEN
              T = XR(I)
              XR(I) = XR(J)
              XR(J) = T
              T = XI(I)
              XI(I) = XI(J)
              XI(J) = T
              T = XR(I+N21)
              XR(I+N21) = XR(J+N21)
              XR(J+N21) = T
              T = XI(I+N21)
              XI(I+N21) = XI(J+N21)
              XI(J+N21) = T
           ENDIF
           T = XR(I+1)
           XR(I+1) = XR(J+N2)
           XR(J+N2) = T
           T = XI(I+1)
           XI(I+1) = XI(J+N2)
           XI(J+N2) = T
           K = N4
C   BROJAC PO J BROJI U BIT INVERZNOM REDOSLEDU
           DOWHILE (K.LT.J)
              J = J - K
              K = K / 2
           ENDDO
           J = J + K
   10   CONTINUE
        RETURN
        END
```

Slika 5.18 Poboljšani potprogram za preuređivanje podataka bit inverzijom indeksa.

gde operacija $br[j]$ označava generisanje broja čija je binarna predstava bit inverzni oblik binarne predstave broja j U većini dosadašnjih programskih realizacija FFT algoritama upotrebljen je algoritam za preuređivanje koji je zasnovan na korišćenju dva brojača, od kojih jedan broji u normalnom a drugi u bit inverznom redosledu [G-11], čija je programska realizacija prikazana na slici 5.17. Pošto se pri preuređivanju izvode samo operacije izračunavanja indeksa i zamene mesta podataka, trajanje operacije preuređivanja predstavlja mali deo (oko 10%) trajanja FFT algoritma. Zbog toga se dugi niz godina nije posvećivala skoro nikakva pažnja ubrzanju operacije

preuređivanja. Međutim, poslednjih nekoliko godina, zbog znatnog ubrzanja izvođenja osnovnog dela FFT algoritama, udeo preuređivanja u ukupnom trajanju algoritma postao je značajniji, pa se u literaturi pojavilo nekoliko metoda koje ubrzavaju proces preuređivanja opisan u [G-11]. Jedna grupa metoda zasnovana je na poboljšanju originalne metode sa bit inverznim brojačem, a druga na podeli binarne predstave indeksa i nalaženju parcijalnih bit inverznih vrednosti. Na slici 5.18 prikazana je FORTRAN verzija algoritma opisanog u [Y-8], koji spada među najjednostavnije i najefikasnije algoritme za preuređivanje vektora po bit inverznom redosledu indeksa.

5.7.4 JEDNOSTAVNA POBOLJŠANJA PROGRAMSKE IMPLEMENTACIJE

Programska implementacija DIT algoritma, prikazana na slici 5.15, nije optimalna sa gledišta brzine izvršavanja. Ona je, nema sumnje, daleko brža od direktnog izračunavanja (slika 5.1), ali se može dosta poboljšati. Pre svega, može se primeniti rekurzivno izračunavanje trigonometrijskih funkcija prema formuli:

$$W_K^{kl} = W_K^k W_K^{k(l-1)} \tag{5.46}$$

Program sa rekurzivnim izračunavanjem trigonometrijskih funkcija, koji je prikazan na slici 5.19, razlikuje se od programa sa slike 5.15 samo u detaljima u spoljnjoj i srednjoj petlji (`DO 30` i `DO 20`).

```
C
C   BRZA FURIJEOVA TRANSFORMACIJA, DIT ALGORITAM
C   REKURZIVNO IZRACUNAVANJE TRIGONOMETRIJSKIH FUNKCIJA
C
C   XR     - REALNI DEO VEKTORA PODATAKA.
C   XI     - IMAGINARNI DEO VEKTORA PODATAKA.
C   M      = LOG2(N)
C
        SUBROUTINE DITFFT(XR,XI,M)
        DIMENSION XR(1), XI(1)
        N=2**M
        DO 30  K = 1,M
           LEP2 = 2**K
           LEP = LEP2 / 2
           UR = 1.
           UI = 0.
           ARG = 6.283185307 / LEP2
           C = COS(ARG)
           S = -SIN(ARG)
           DO 20  J = 1,LEP
              DO 10  I = J,N,LEP2
                 IP = I + LEP
                 TR = XR(IP)*UR - XI(IP)*UI
                 TI = XR(IP)*UI + XI(IP)*UR
                 XR(IP) = XR(I) - TR
                 XI(IP) = XI(I) - TI
                 XR(I) = XR(I) + TR
                 XI(I) = XI(I) + TI
    10        CONTINUE
              TR=UR
              UR = TR*C - UI*S
              UI = TR*S + UI*C
    20     CONTINUE
    30  CONTINUE
        RETURN
        END
```

Slika 5.19 FORTRAN implementacija rekurzivnog DIT algoritma.

Osnovni nedostatak rekurzivnog izračunavanja trigonometrijskih funkcija je akumulacija grešaka izračunavanja, što je posebno izraženo ako su promenljive predstavljene malim brojem bita i ako se radi sa dugačkim sekvencama. U takvim slučajevima se nekim pogodnim koeficijentima mogu dati unapred definisane vrednosti (na primer $W_K^{K/4} = -j$), čime se svakako popravlja tačnost izračunavanja.

Još brže i tačnije izračunavanje DFT može se postići ako se sve vrednosti trigonometrijskih funkcija unapred izračunaju i smeste u tabelu (dva vektora dimenzije $N/2$). Koeficijenti mogu biti izračunati i zapamćeni u normalnom ili bit inverznom redosledu. Ovaj pristup je pogodan ako se program često koristi za određivanje DFT sekvenci iste dužine. Primer implementacije DIT algoritma sa trigonometrijskim funkcijama u tabeli prikazan je na slici 5.20. Inicijalizacija tabela sinusa i kosinusa može se izvršiti pomoću potprograma sa slike 5.2 pri čemu se koriste samo prve polovine generisanih vektora.

```
C
C   BRZA FURIJEOVA TRANSFORMACIJA, DIT ALGORITAM
C   TRIGONOMETRIJSKE FUNKCIJE U TABELAMA
C
C   XR    - REALNI DEO VEKTORA PODATAKA.
C   XI    - IMAGINARNI DEO VEKTORA PODATAKA.
C   M     = LOG2(N)
C   WR    - TABLICA VREDNOSTI KOSINUSA.
C   WI    - TABLICA VREDNOSTI SINUSA.
C
        SUBROUTINE DITFFT(XR,XI,M,WR,WI)
        DIMENSION XR(1), XI(1), WR(1), WI(1)
        N=2**M
        DO 30  K = 1,M
           LEP2 = 2**K
           LEP = LEP2 / 2
           IS = N /LEP2
           IN = 1
           DO 20  J = 1,LEP
              C = WR(IN)
              S = -WI(IN)
              IN=IN+IS
              DO 10  I = J,N,LEP2
                 IP = I + LEP
                 TR = XR(IP)*C - XI(IP)*S
                 TI = XR(IP)*S + XI(IP)*C
                 XR(IP) = XR(I) - TR
                 XI(IP) = XI(I) - TI
                 XR(I) = XR(I) + TR
                 XI(I) = XI(I) + TI
    10        CONTINUE
    20     CONTINUE
    30  CONTINUE
RETURN
END
```

Slika 5.20 FORTRAN implementacija DIT algoritma sa tablicom trigonometrijskih funkcija.

5.7.5 REDUKCIJA ARITMETIČKE SLOŽENOSTI ALGORITAMA

Analizom potrebnog broja operacija za izvršavanje FFT algoritma sa preuređivanjem u vremenu došlo se do izraza (5.29), prema kome broj izračunavanja trigonometrijskih funkcija iznosi $N \log_2 N$. Međutim, u toj analizi je zanemareno da u svim leptirima nije potrebno generisati po dve vrednosti trigonometrijskih funkcija. Naime, u leptirima u prvom stepenu izračunavanja kod DIT algoritma (ili poslednjem stepenu kod DIF algoritma) vrednost rotacionog faktora je jednaka 1

(W_K^0). Broj takvih leptira je $N_{B1} = N/2$ pa se broj izračunavanja trigonometrijskih funkcija može smanjiti za $2N_{B1} = N$, broj množenja za $4N_{B1} = 2N$, a broj sabiranja za $2N_{B1} = N$. Programska implementacija DIF algoritma sa dodatnim leptirom koji se izvršava kada je vrednost rotacionog faktora jednaka 1 prikazana je na slici 5.21.

```
C
C   BRZA FURIJEOVA TRANSFORMACIJA.
C   DIF ALGORITAM SA DVA LEPTIRA
C   TRIGONOMETRIJSKE FUNKCIJE U TABELAMA.
C
C   XR     - REALNI DEO VEKTORA PODATAKA.
C   XI     - IMAGINARNI DEO VEKTORA PODATAKA.
C   M      = LOG2(N)
C   WR     - TABLICA VREDNOSTI KOSINUSA.
C   WI     - TABLICA VREDNOSTI SINUSA.
C
        SUBROUTINE DIF2FFT(XR,XI,M,WR,WI)
        DIMENSION XR(1), XI(1), WR(1), WI(1)
        N=2**M
        DO 40  K = 1,M
           LEP2 = 2**(M+1-K)
           LEP = LEP2 / 2
C
C   LEPTIR ZA MNOZENJE SA 1
C
           DO 10  I = 1,N,LEP2
              IP = I + LEP
              TR = XR(I) - XR(IP)
              TI = XI(I) - XI(IP)
              XR(I) = XR(I) + XR(IP)
              XI(I) = XI(I) + XI(IP)
              XR(IP) = TR
              XI(IP) = TI
   10      CONTINUE
           IF (K.LT.M) THEN
              KP = 2**(K-1)
              IARG = 1
              DO 30  J = 2,LEP
                 IARG = IARG + KP
                 C = WR(IARG)
                 S = -WI(IARG)
C
C   STANDARDNI LEPTIR
C
                 DO 20  I = J,N,LEP2
                    IP = I + LEP
                    TR = XR(I) - XR(IP)
                    TI = XI(I) - XI(IP)
                    XR(I) = XR(I) + XR(IP)
                    XI(I) = XI(I) + XI(IP)
                    XR(IP) = TR*C - TI*S
                    XI(IP) = TR*S + TI*C
   20            CONTINUE
   30         CONTINUE
           ENDIF
   40   CONTINUE
        RETURN
        END
```

Slika 5.21 Realizacija DIF algoritma sa dva leptira.

Iz dijagrama toka DIT i DIF algoritama uočava se da u nekim leptirima vrednost rotacionog faktora iznosi $j = W_K^{K/4}$, tako da se broj potrebnih izračunavanja trigonometrijskih funkcija, kao i broj realnih množenja i sabiranja može dalje smanjiti uvođenjem još jednog posebnog leptira. Broj takvih leptira iznosi:

$$N_{B2} = \frac{N}{2}\sum_{i=0}^{p-2} 2^{-i} = 2^p - 2 = N - 2 \tag{5.47}$$

pa se broj izračunavanja trigonometrijskih funkcija može smanjiti za $2N_{B2} = 2N - 4$, broj množenja za $4N_{B2} = 4N - 8$, a broj sabiranja za $2N_{B2} = 2N - 4$. Naravno, programska implementacija ima tri različita leptira i prikazana je na slici 5.22 za slučaj DIF algoritma.

Dalje smanjenje broja izračunavanja trigonometrijskih funkcija, uz povećanje složenosti programa, može se izvršiti ako se uoči da su vrednosti rotacionog faktora u specijalnim slučajevima $W_K^{K/8} = \sqrt{2}(1+j)$ i $W_K^{3K/8} = \sqrt{2}(-1+j)$, odnosno da zahtevaju samo po jedno izračunavanje trigonometrijskih funkcija, dva realna množenja i četiri realna sabiranja po leptiru. Kako je broj takvih leptirova:

$$N_{B3} = \frac{N}{4}\sum_{i=0}^{p-3} 2^{-i} = 2^{p-1} - 2 = N/2 - 2 \tag{5.48}$$

broj izračunavanja trigonometrijskih funkcija smanjen je za $N_{B3} = N/2 - 2$, a broj množenja i sabiranja za po $2N_{B3} = N - 4$. Programska implementacija onda ima pet različitih leptirova pa se zbog toga retko koristi.

U slučajevima kada se izračunavanje po FFT algoritmu izvršava na sistemu kod koga je trajanje operacije množenja znatno duže od trajanja operacije sabiranja, može se postići znatna ušteda modifikacijom algoritma za množenje kompleksnih brojeva. Naime, uobičajeni način za množenje kompleksnih brojeva koji zahteva četiri množenja realnih brojeva i dva sabiranja realnih brojeva (tzv. $4/2$ *algoritam*):

$$E + jF = (A + jB)(C + jD) = (AC - BD) + j(AD + BC) \tag{5.49}$$

može se zameniti ekvivalentnim izrazom:

$$E + jF = [A(C - D) + D(A - B)] + j[B(C + D) + D(A - B)] \tag{5.50}$$

koji, kao što se vidi, zahteva samo tri množenja realnih brojeva i pet sabiranja realnih brojeva ($3/5$ *algoritam*). Međutim, ako se FFT algoritam realizuje sa tabličnim očitavanjem vrednosti trigonometrijskih funkcija, umesto da se u memoriji čuvaju vrednosti za C i D, mogu se unapred izračunati i pamtiti vrednosti $C + D$ i $C - D$, tako da se broj sabiranja smanjuje na tri (tzv. $3/3$ algoritam). Ovakav način izvođenja množenja kompleksnih brojeva ima prednosti na svim računarskim sistemima kod kojih je trajanje operacije množenja duže od trajanja operacije sabiranja, a to su praktično svi sistemi bez posebne hardverske podrške za operaciju množenja.

5.8 FFT ALGORITMI ZA $N = 4^p$ I $N = 8^p$

FFT algoritmi sa preuređivanjem u vremenu i po frekvenciji koji su opisani u prethodna tri odeljka bili su razvijeni za dužinu sekvence $N = 2^p$ pa se stoga nazivaju *algoritmi sa osnovom* 2, ili, prema engleskoj terminologiji, *radiks*-2 *algoritmi*.

U literaturi o efikasnom izračunavanju DFT poznati su i drugačiji načini dekompozicije. Na primer, sekvenca se može deliti na četiri podsekvence (ako je $N = 4^p$) ili na osam podsekvenci (ako je $N = 8^p$). Kako se time mogu postići izvesne uštede u broju izračunavanja, interesantno je razmotriti i klase *radiks*-4 i *radiks*-8 algoritama.

```
C  BRZA FURIJEOVA TRANSFORMACIJA, DIF ALGORITAM SA TRI LEPTIRA.
C  TRIGONOMETRIJSKE FUNKCIJE U TABELAMA.
C
C  XR        - REALNI DEO VEKTORA PODATAKA.
C  XI        - IMAGINARNI DEO VEKTORA PODATAKA.
C  M         = LOG2(N)
C  WR        - TABLICA VREDNOSTI KOSINUSA.
C  WI        - TABLICA VREDNOSTI SINUSA.
C
         SUBROUTINE DIF3FFT(XR,XI,M,WR,WI)
         DIMENSION XR(1), XI(1), WR(1), WI(1)
         N=2**M
         DO 50  K = 1,M
            LEP2 = 2**(M+1-K)
            LEP = LEP2 / 2
            JLAST = LEP + 1
C
C  SPECIJALNI LEPTIR ZA MNOZENJE SA 1
C
            DO 10  I = 1,N,LEP2
               IP = I + LEP
               TR = XR(I) - XR(IP)
               TI = XI(I) - XI(IP)
               XR(I) = XR(I) + XR(IP)
               XI(I) = XI(I) + XI(IP)
               XR(IP) = TR
               XI(IP) = TI
   10       CONTINUE
            IF (K.LT.M) THEN
               KP = 2**(K-1)
               IARG = 1
               DO 40  J = 2,LEP
                  IARG = IARG + KP
                  IF (J.NE.JLAST) THEN
                     C = WR(IARG)
                     S = -WI(IARG)
C
C  STANDARDNI LEPTIR
C
                     DO 20  I = J,N,LEP2
                        IP = I + LEP
                        TR = XR(I) - XR(IP)
                        TI = XI(I) - XI(IP)
                        XR(I) = XR(I) + XR(IP)
                        XI(I) = XI(I) + XI(IP)
                        XR(IP) = TR*C - TI*S
                        XI(IP) = TR*S + TI*C
   20                CONTINUE
                  ELSE
C
C  SPECIJALNI LEPTIR ZA MNOZENJE SA J
C
                     DO 30  I = J,N,LEP2
                        IP = I + LEP
                        TR = XR(I) - XR(IP)
                        TI = XI(I) - XI(IP)
                        XR(I) = XR(I) + XR(IP)
                        XI(I) = XI(I) + XI(IP)
                        XR(IP) = TI
                        XI(IP) = -TR
   30                CONTINUE
                  ENDIF
   40          CONTINUE
            ENDIF
   50    CONTINUE
         RETURN
         END
```

Slika 5.22 Realizacija DIF algoritma sa tri leptira.

5.8.1 RADIKS-4 ALGORITMI

Posmatrajmo prvo FFT algoritam sa preuređivanjem ulazne sekvence. Ulazna sekvenca se deli na četiri podsekvence: $x[4n], x[4n+1], x[4n+2]$ i $x[4n+3]$ gde je $n = 0,1,\ldots, N/4$. Onda je:

$$X[k] = \sum_{n=0}^{N/4-1} x[4n]W_N^{4nk} + \sum_{n=0}^{N/4-1} x[4n+1]W_N^{(4n+1)k} + \sum_{n=0}^{N/4-1} x[4n+2]W_N^{(4n+2)k} + \sum_{n=0}^{N/4-1} x[4n+3]W_N^{(4n+3)k} =$$

$$= \sum_{l=0}^{3} W_N^{lk}\left[\sum_{n=0}^{N/4-1} x[4n+l]W_N^{4nk}\right] = \sum_{l=0}^{3} W_N^{lk}\left[\sum_{n=0}^{N/4-1} x_{1l}[n]W_{N/4}^{nk}\right] = \sum_{l=0}^{3} W_N^{lk} X_{1l}[k] \tag{5.51}$$

Takođe je:

$$\begin{aligned} X[k+N/4] &= \sum_{l=0}^{3} W_N^{l(k+N/4)}\left[\sum_{n=0}^{N/4-1} x_{1l}[n]W_{N/4}^{n(k+N/4)}\right] = \\ &= \sum_{l=0}^{3} (-j)^l W_N^{lk}\left[\sum_{n=0}^{N/4-1} x_{1l}[n]W_{N/4}^{nk}\right] = \sum_{l=0}^{3} (-j)^l W_N^{lk} X_{1l}[k] \end{aligned} \tag{5.52}$$

$$X[k+N/2] = \sum_{l=0}^{3} (-1)^l W_N^{lk} X_{1l}[k] \tag{5.53}$$

$$X[k+3N/4] = \sum_{l=0}^{3} (j)^l W_N^{lk} X_{1l}[k] \tag{5.54}$$

ili, uopšteno:

$$X[k+mN/4] = \sum_{l=0}^{3} (-j)^{lm} W_N^{lk} X_{1l}[k] \tag{5.55}$$

gde je:

$$X_{1l}[k] = \sum_{n=0}^{N/4-1} x_{1l}[n]W_{N/4}^{nk} = \sum_{n=0}^{N/4-1} x[4n+l]W_{N/4}^{nk} \tag{5.56}$$

Leptir operacija za radiks-4 DIT algoritam prikazana je na slici 5.23. Pošto je $W_N^0 = 1$, za izvršenje leptir operacije potrebna su samo tri kompleksna, odnosno, 12 realnih množenja. Kako se postupak podele može nastaviti jer je $N = 4^p = 2^{2p}$, ukupno ima p stepeni sa $N/4$ leptirova u svakom stepenu. Dakle, ukupan broj realnih množenja iznosi:

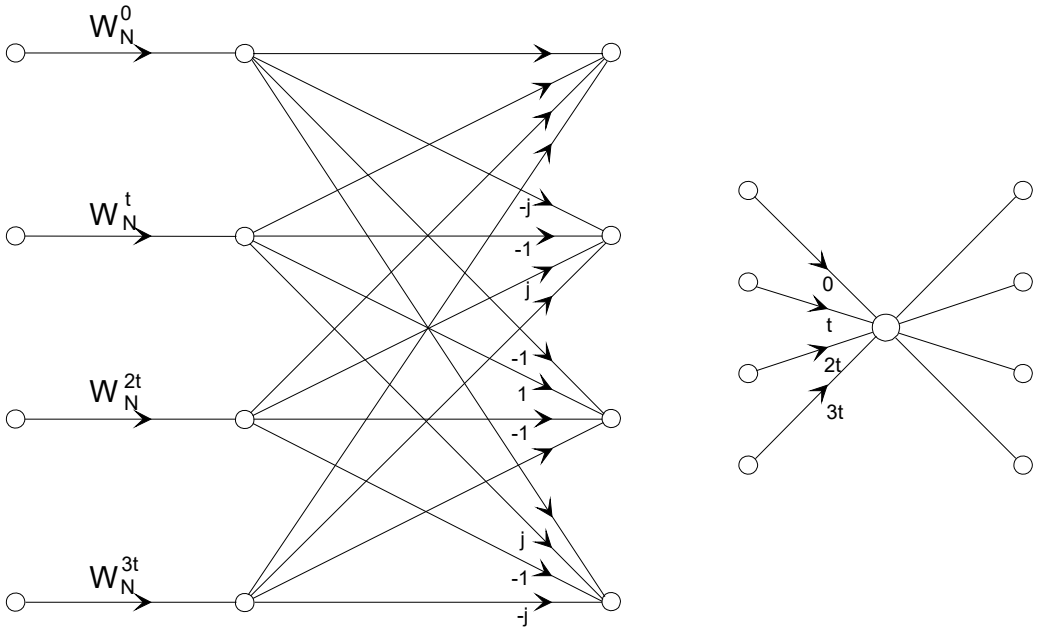
$$N_M = 12p\frac{N}{4} = 1.5N\log_2 N \tag{5.57}$$

što je za 25% manje od odgovarajućeg broja množenja kod osnovnih verzija radiks-2 algoritama.

Broj kompleksnih sabiranja u leptir operaciji sa slike 5.23 iznosi 12. Međutim, grupisanjem sabiranja u leptiru i višestrukim korišćenjem dobijenih međurezultata, broj kompleksnih sabiranja može se smanjiti na osam. Dakle, ukupan broj realnih sabiranja, uključujući i ona koja potiču od množenja kompleksnih brojeva iznosi:

$$N_A = 16p\frac{N}{4} + 6p\frac{N}{4} = 2.75N\log_2 N \tag{5.58}$$

odnosno, smanjen je za oko 8% u odnosu na osnovne verzije radiks-2 algoritama.



(a) **(b)**

Slika 5.23 Leptir operacija za radiks-4 DIT algoritam: (a) potpuni, (b) uprošćeni simbol.

Kao i kod radiks-2 algoritama, broj potrebnih aritmetičkih operacija može se dalje smanjiti uvođenjem specijalnih leptira. Isto tako, za realizaciju kompleksnog množenja može se koristiti 3/3 algoritam.

Preuređivanje ulazne sekvence je neophodno ako se želi izračunavanje u mestu. U ovom slučaju, pošto je u pitanju radiks-4 algoritam, indeksi se preuređuju po digit inverznom redosledu. Preuređivanje se izvodi na sledeći način. Prvo se indeksi iz decimalnog sistema prevedu u brojni sistem sa osnovom 4. Zatim se invertuje redosled cifara koje predstavljaju indekse, i na kraju se invertovane predstave indeksa ponovo vrate u decimalni oblik. Treba primetiti da bit inverzni i digit inverzni redosled nisu isti.

Na sličan način može se izvesti odgovarajući radiks-4 DIF algoritam. U tom slučaju, izlazna sekvenca se deli na četiri podsekvence: $X[4k], X[4k+1], X[4k+2]$ i $X[4k+3]$, gde je $k = 0,1,\ldots, N/4$. Na sličan način kao kod DIT algoritma dobija se:

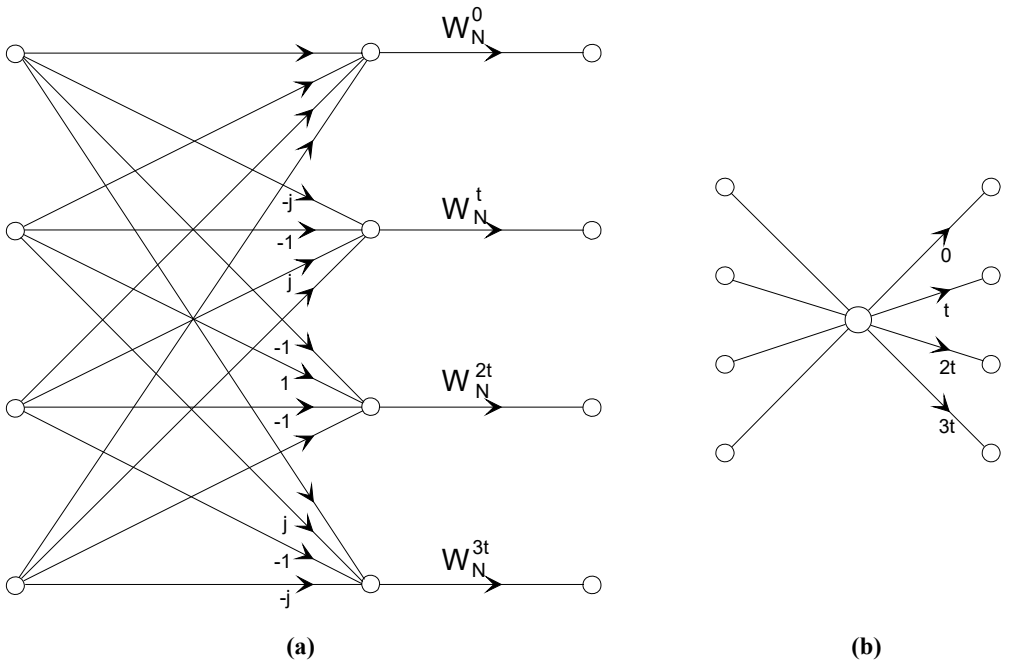
$$X[4k+m] = \sum_{n=0}^{N/4-1} x_{1m}[n] W_{N/4}^{kn} \tag{5.59}$$

gde je:

$$x_{1m}[n] = \sum_{l=0}^{3} (-j)^{lm} W_N^{lk} x[n + l\,N/4] \tag{5.60}$$

Leptir operacija za radiks-4 DIF algoritam prikazana je na slici 5.24.

Nastavljajući proces dekompozicije sekvenci dolazi se do konačne verzije radiks-4 algoritma sa preuređivanjem u frekvencijskom domenu, koja je poznata kao radiks-4 DIF algoritam. Broj aritmetičkih operacija je identičan kao kod DIT algoritma. Jedina razlika između ova dva tipa algoritma je u konstrukciji leptira.



Slika 5.24 Leptir operacija za radiks-4 DIF algoritam: (a) potpuni, (b) uprošćeni simbol.

5.8.2 PROGRAMSKA IMPLEMENTACIJA RADIKS-4 ALGORITAMA

Programi za realizaciju radiks-2 DIT i DIF algoritama koje su prikazani na slikama 5.15 i 5.16 mogu se veoma lako modifikovati za implementaciju radiks-4 DIT i DIF algoritama. Osnovne izmene su u instrukcijama unutrašnje petlje koja realizuje leptir operaciju, kao i u generisanju indeksa vektora u kojima se nalaze sekvence. Kao primer, na slici 5.25 prikazana je implementacija radiks-4 DIF algoritma sa izračunavanjem trigonometrijskih funkcija, dok je program za preuređivanje sekvence u digit inverzni redosled prikazan na slici 5.26. Efikasnije programske realizacije radiks-4 algoritama kod kojih se koristi više leptirova, a trigonometrijske funkcije čitaju iz tabele ili rekurzivno izračunavaju, mogu se lako napisati modifikacijom odgovarajućih programa za radiks-2 algoritme.

5.8.3 RADIKS-8 ALGORITMI

Deljenjem ulazne, odnosno izlazne sekvence na osam podsekvenci, dobijaju se radiks-8 DIT, odnosno DIF algoritmi. Leptiri za radiks-8 algoritme imaju osam ulaznih podataka, pa je njihova struktura komplikovanija. S druge strane broj takvih leptirova je manji. Broj realnih množenja kod radiks-8 algoritama je:

$$N_M = 1.33N \log_2 N \tag{5.61}$$

dok je broj realnih sabiranja:

$$N_A = 2.75N \log_2 N \tag{5.62}$$

Kao što se vidi, broj realnih množenja je manji nego kod radiks-2 algoritama za 33%, dok je broj realnih sabiranja manji za 8%. S obzirom da broj elemenata sekvence mora biti $N = 8^p$ (8, 64, 512, 4096, itd.) radiks-8 algoritmi se znatno ređe primenjuju od radiks-2 i radiks-4 algoritama.

Dalje povećanje broja podsekvenci pri dekompoziciji smanjuje broj množenja ali povećava broj sabiranja, pa nije ni od teorijskog ni praktičnog značaja.

```
C   BRZA FURIJEOVA TRANSFORMACIJA, RADIKS-4 DIF ALGORITAM
C
C   XR    - REALNI DEO VEKTORA PODATAKA.
C   XI    - IMAGINARNI DEO VEKTORA PODATAKA.
C   M     = 0.5*LOG2(N)
C
      SUBROUTINE DIF4FFT(XR,XI,M)
      DIMENSION XR(1), XI(1)
      N=4**M
      DO 30  K = 1,M
         LEP4 = 4**(M+1-K)
         PHI = 6.283185307 / LEP4
         LEP = LEP4 / 4
         ARG1 =  0.
         DO 20  J = 1,LEP
            ARG2 = ARG1 + ARG1
            ARG3 = ARG1 + ARG2
            C1 = COS(ARG1)
            C2 = COS(ARG2)
            C3 = COS(ARG3)
            S1 = SIN(ARG1)
            S2 = SIN(ARG2)
            S3 = SIN(ARG3)
            ARG1 = J * PHI
            DO 10  I = J,N,LEP4
               IP1 = I + LEP
               IP2 = IP1 + LEP
               IP3 = IP2 + LEP
               TR1 = XR(I) + XR(IP2)
               TR3 = XR(I) - XR(IP2)
               TI1 = XI(I) + XI(IP2)
               TI3 = XI(I) - XI(IP2)
               TR2 = XR(IP1) + XR(IP3)
               TR4 = XR(IP1) - XR(IP3)
               TI2 = XI(IP1) + XI(IP3)
               TI4 = XI(IP1) - XI(IP3)
               XR(I) = TR1 + TR2
               TR2 = TR1 - TR2
               TR1 = TR3 - TI4
               TR3 = TR3 + TI4
               XI(I) = TI1 + TI2
               TI2 = TI1 - TI2
               TI1 = TI3 + TR4
               TI3 = TI3 - TR4
               XR(IP1) = C1*TR3 + S1*TI3
               XI(IP1) = C1*TI3 - S1*TR3
               XR(IP2) = C2*TR2 + S2*TI2
               XI(IP2) = C2*TI2 - S2*TR2
               XR(IP3) = C3*TR1 + S3*TI1
               XI(IP3) = C3*TI1 - S3*TR1
   10       CONTINUE
   20    CONTINUE
   30 CONTINUE
      RETURN
      END
```

Slika 5.25 FORTRAN potprogram za radiks-4 DIF algoritam.

5.9 GENERALNA TEORIJA FFT ALGORITAMA

U prethodnom izlaganju dužina sekvence čija se DFT određuje bila je ograničena uslovom $N = R^p$, $(R = 2,4,8)$. Ovakvo ograničenje omogućilo je jednostavnu dekompoziciju i razvoj efikasnih FFT algoritama. Međutim, u praksi postoje slučajevi kada ulazna sekvenca ne zadovoljava ograničenje $N = R^p$ i ne sme se proširivati nulama. U literaturi o izračunavanju DFT

razvijeni su postupci efikasnog izračunavanja koji su primenljivi na mnogo širi krug slučajeva, a mogu dovesti i do efikasnijih rešenja. U osnovi svih metoda opet leži princip dekompozicije sekvence, samo što je taj princip u opštem slučaju zamenjen preslikavanjem jednodimenzionalne sekvence u višedimenzionalnu sekvencu.

```
C   PREUREDJIVANJE VEKTORA PO DIGIT INVERZNOM
C   REDOSLEDU INDEKSA ZA RADIKS-4 FFT ALGORITME.
C
C   XR     - REALNI DEO VEKTORA PODATAKA.
C   XI     - IMAGINARNI DEO VEKTORA PODATAKA.
C   M      = 0.5*LOG2(N)
C
        SUBROUTINE DIGREV (XR,XI,M)
        DIMENSION XR(1), XI(1)
        N = 4**M
        N4 = N / 4
        LAST = N - 1
        J = 1
C   BROJAC PO I BROJI U PRIRODNOM REDOSLEDU
        DO 10  I = 1,LAST
           IF (I.LT.J) THEN
              T = XR(I)
              XR(I) = XR(J)
              XR(J) = T
              T = XI(I)
              XI(I) = XI(J)
              XI(J) = T
           ENDIF
           K = N4
C   BROJAC PO J BROJI U DIGIT INVERZNOM REDOSLEDU
           DOWHILE (K*3.LT.J)
              J = J - K*3
              K = K / 4
           ENDDO
           J = J + K
   10   CONTINUE
        RETURN
        END
```

Slika 5.26 FORTRAN potprogram za preuređivanje sekvenci kod radiks-4 algoritama.

5.9.1 ALGORITMI TIPA RADIKS-*R*

Da bi bila moguća dekompozicija sekvence, njena dužina N mora biti proizvod bar dva faktora, odnosno,

$$N = N_1 N_2 \tag{5.63}$$

Dekompozicija sekvence najlakše se može objasniti pomoću *preslikavanja indeksa* [B-30, B-35]. Neka su indeksi ulazne i izlazne sekvence iz FFT algoritma n i k dati izrazima:

$$n = N_2 n_1 + n_2, \quad 0 \le n_1 \le N_1 - 1, \quad 0 \le n_2 \le N_2 - 1 \tag{5.64}$$

$$k = k_1 + N_1 k_2, \quad 0 \le k_1 \le N_1 - 1, \quad 0 \le k_2 \le N_2 - 1 \tag{5.65}$$

Iz teorije brojeva je poznato da su preslikavanja opisana jednačinama (5.64) i (5.65) jednoznačna, što se lako može i eksperimentalno ustanoviti. To znači da kada n_1 i n_2 (k_1 i k_2) uzimaju redom vrednosti iz dozvoljenog opsega, n (k) dobijaju sve vrednosti od 0 do $N-1$, ne preskačući ili ispuštajući nijednu vrednost. Primenjujući preslikavanja (5.64) i (5.65) na indekse iz jednačine (5.1), dobija se:

$$X[k]=X[k_1+N_1k_2]=\sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1}x[N_2n_1+n_2]W_N^{(k_1+N_1k_2)(N_2n_1+n_2)}=$$

$$=\sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1}x[N_2n_1+n_2]W_N^{N_2k_1n_1}W_N^{k_1n_2}W_N^{N_1k_2n_2}W_N^{N_1N_2k_2n_1} \tag{5.66}$$

S obzirom da je $W_N^{N_2k_1n_1}=W_{N_1}^{k_1n_1}$, $W_N^{N_1k_2n_2}=W_{N_2}^{k_2n_2}$, i $W_N^{N_1N_2k_2n_1}=1$, iz (5.66) se dobija:

$$X[k_1+N_1k_2]=\sum_{n_2=0}^{N_2-1}\left\{\left[\sum_{n_1=0}^{N_1-1}x[N_2n_1+n_2]W_{N_1}^{k_1n_1}\right]W_N^{k_1n_2}\right\}W_{N_2}^{k_2n_2} \tag{5.67}$$

Jednačina (5.67) ima veoma jednostavnu interpretaciju. Ulazna jednodimenziona sekvenca se preslikava u dvodimenzionu sekvencu (matricu), gde n_1 označava redni broj kolone, a n_2 redni broj vrste. Izraz u srednjoj zagradi u (5.67) predstavlja DFT sekvenci dužine N_1 (vrsta matrice) kojih ima N_2, tako da se može definisati pomoćna matrica:

$$G[n_2,k_1]=\sum_{n_1=0}^{N_1-1}x[N_2n_1+n_2]W_{N_1}^{k_1n_1} \tag{5.68}$$

Izračunavanje pomoćne matrice $G[n_2,k_1]$ može se obaviti u mestu jer ulazna matrica nije više potrebna. Zatim se svaka vrsta matrice $G[n_2,k_1]$ množi sa faktorom $W_N^{k_1n_2}$ čime se dobija:

$$\hat{G}[n_2,k_1]=W_{N_1}^{k_1n_2}G[n_2,k_1] \tag{5.69}$$

Ova operacija se takođe može izvršiti u mestu. Faktori $W_N^{k_1n_2}$ nazivaju se *rotacioni faktori* (engl. *twiddle factors*) jer izazivaju samo rotaciju u kompleksnoj ravni zbog toga što imaju jedinični moduo. Kada ne bi bilo množenja rotacionim faktorima, izraz (5.67) predstavljao bi pravu dvodimenzionalnu DFT. Na kraju, suma po n_2 u jednačini (5.67) predstavlja skup DFT sekvenci od N_2 elemenata koje predstavljaju kolone matrice $\hat{G}[n_2,k_1]$:

$$X[k_1+N_1k_2]=\sum_{n_2=0}^{N_2-1}\hat{G}[n_2,k_1]W_{N_2}^{k_2n_2} \tag{5.70}$$

I ova operacija se može obaviti u mestu. Dakle, kompletno izračunavanje DFT može se obaviti u mestu, što je, na drugi način, već pokazano u prethodnim izlaganjima.

Jedan interesantan specijalni slučaj se dobija za $N_1=2$, $N_2=N/2$. Tada izračunavanja prema jednačini (5.67) odgovaraju prvom stepenu algoritma sa preuređivanjem u frekvencijskom domenu. Drugi specijalni slučaj se dobija za $N_1=N/2$, $N_2=2$. Tada izračunavanja prema jednačini (5.67) odgovaraju prvom stepenu algoritma sa preuređivanjem u vremenu.

Preslikavanja indeksa (5.64) i (5.65) nisu jedina jednoznačna preslikavanja iz jedne u dve dimenzije. Na primer, preslikavanje indeksa se može izvršiti i prema jednačinama:

$$n=n_1+N_1n_2,\quad 0\le n_1\le N_1-1,\quad 0\le n_2\le N_2-1 \tag{5.71}$$

$$k=N_2k_1+k_2,\quad 0\le k_1\le N_1-1,\quad 0\le k_2\le N_2-1 \tag{5.72}$$

U ovom slučaju izbor $N_1=2$, $N_2=N/2$ dovodi do prvog stepena algoritma sa preuređivanjem u vremenu, dok izbor $N_1=N/2$, $N_2=2$ dovodi do prvog stepena algoritma sa preuređivanjem u frekvencijskom domenu.

U dosadašnjem razvoju brzih DFT algoritama nije bilo reči o složenosti faktora N_1 i N_2. Jedan od njih, ili oba, mogu se dalje rastaviti na faktore. Najefikasniji algoritmi se dobijaju ako je N složeni broj oblika:

$$N = N_1 N_2 \cdots N_p \tag{5.73}$$

kada se dobija preslikavanje indeksa u više dimenzija. Slučaj $N = 2^p$ predstavlja specijalni slučaj rastavljanja broja N prema (5.73). Interesantno je primetiti da se onda izračunavanje DFT svodi na niz izračunavanja DFT od dva elementa između kojih se rezultati množe rotacionim faktorima. Višedimenziono preslikavanje indeksa dato sa (5.64) i (5.65) se onda svodi na:

$$n = 2^{p-1} n_1 + 2^{p-2} n_2 + \cdots + 2 n_{p-1} + n_p \tag{5.74}$$

$$k = k_1 + 2k_2 + \cdots + 2^{p-2} k_{p-1} + 2^{p-1} k_p \tag{5.75}$$

Pošto indeksi n_i i k_i mogu imati vrednosti 0 ili 1, na osnovu (5.74) i (5.75) vidi se da su, ako je ulazna sekvenca u normalnom redosledu, indeksi izlazne sekvence u bit inverznom redosledu, ako se izračunavanje obavlja u mestu.

Ista analiza se može sprovesti i za druge vrednosti N oblika $N = R^p$ kada se dobija klasa tzv. *radiks-R algoritama*. U literaturi se često takvi DFT algoritmi nazivaju *Kuli-Taki algoritmi* (Cooley-Tukey) po autorima jednog od najznačajnijih radova [C-38] iz oblasti brzog izračunavanja DFT.

Kao i u prethodnim analizama, kompleksnost algoritama može se proceniti ispitivanjem broja aritmetičkih operacija. Međutim, pošto iz jednačine (5.63) sledi da se ulazna sekvenca rastavlja na N_1 parcijalnih sekvenci dužine N_2 koje se mogu svrstati u dvodimenzionu matricu, na čije će se vrste i kolone primenjivati DFT algoritam, potrebno je na neki način utvrditi broj operacija u svakom koraku dekompozicije. Ako se, na primer, sa $M(N)$ označi broj kompleksnih množenja u DFT algoritmu, primenjenom na sekvencu dužine N, onda se u slučaju opisanom jednačinom (5.63) potreban broj množenja dobija sabiranjem tri člana:

1. Broja množenja za izvršavanje DFT po vrstama, $N_2 M(N_1)$,
2. Broja množenja rotacionim faktorima, $N_1 N_2 = N$, i,
3. Broja množenja za izvršavanje DFT po kolonama, $N_1 M(N_2)$.

Dakle, ima se:

$$M(N) = N_2 M(N_1) + N + N_1 M(N_2) = N\left[\frac{M(N_1)}{N_1} + \frac{M(N_2)}{N_2} + 1\right] \tag{5.76}$$

U opštem slučaju, ako je N složen broj dat sa (5.73), jednačina (5.76) se može generalizovati na oblik:

$$M(N) = N\left[\sum_{i=1}^{p} \frac{M(N_i)}{N_i} + p - 1\right] \tag{5.77}$$

odakle se može zaključiti, da se minimalni broj množenja dobija ako je $N = 3^p$, tj $N_i = 3$. Ovakav zaključak bi bio u važnosti kada bi za svako N_i važilo $M(N_i) = N_i^2$, što ne važi za neke vrednosti N_i na primer za $N_i = 2$, ili za $N_i = 4$ kada za izračunavanje DFT nije potrebno nijedno množenje. U posebnom slučaju kada je $N = 2^p$, uzimajući u obzir da je $M(2) = 0$, iz (5.77) sledi:

$$M(N) = N(p-1) = N(\log_2 N - 1) \tag{5.78}$$

što je, na prvi pogled, u kontradikciji sa jednačinom (5.27). Međutim, u jednačini (5.27) figuriše i $N/2$ trivijalnih kompleksnih množenja sa jedan u prvom stepenu izračunavanja (DFT sekvence od dva elementa), koja nisu uračunata u jednačini (5.78). Ako se ova razlika uzme u obzir, obe formule daju isti rezultat.

Prethodna formula daje i objašnjenje zašto je radiks-4 algoritam efikasniji od radiks-2 algoritma u slučajevima kada se mogu primeniti oba algoritma. Kada je $N = 4^p = 2^{2p}$, kod radiks-2 algoritma u dijagramu toka postoji $2p$ stepena sa po $N/2$ radiks-2 leptira, dok kod radiks-4 algoritma u dijagramu toka ima p stepeni sa po $N/4$ radiks-4 leptira. Kako u oba slučaja za izračunavanje leptira nije potrebno vršiti množenja, broj množenja u algoritmu određuje broj rotacionih faktora koji nisu jednaki jedinici, a tu prednost očigledno ima radiks-4 algoritam. Slična analiza može se izvršiti i za određivanje broja operacija sabiranja.

Prethodno razmatranje daje ideju kako se u izvesnim slučajevima može poboljšati efikasnost FFT algoritma. Na primer, ako je $N = 2^p$, onda se N može faktorizovati i na sledeći način:

$$N = 2^p = 2^{p_1} 4^{p_2} 8^{p_3}, \quad p = p_1 + 2p_2 + 3p_3 \tag{5.79}$$

Dakle, pošto je prema prethodnoj analizi, po efikasnosti najbolji radiks-8 algoritam, radiks-4 algoritam je nešto manje efikasan, dok je radiks-2 algoritam najmanje efikasan, pri dekompoziciji treba što je moguće više koristiti radiks-8 stepene i radiks-4 stepene. Tako se dobija *algoritam sa mešovitim radiksom* (engl. *mixed radix*) [S-10] koji je efikasniji od prostih radiks-2 ili radiks-4 algoritama i koji je dat u Prilogu 1.

5.9.2 ALGORITAM SA DVOSTRUKIM RADIKSOM

Drugi primer efikasnog kombinovanja radiks-2 i radiks-4 algoritama predstavlja *algoritam sa dvostrukim radiksom* (engl. *split-radix algorithm*) [D-11, D-13]. Naime, posmatranjem slike 5.10, na kojoj je prikazan prvi korak u razvoju radiks-2 DIF algoritma, može se lako uočiti da se množenja rotacionim faktorima pojavljuju samo u onoj polovini dijagrama toka u kome se izračunavaju elementi izlazne sekvence sa neparnim indeksima. U slučaju radiks-4 DIF algoritma to ne važi, što je iznenađujuća činjenica jer je pokazano da je radiks-4 generalno efikasniji. Na osnovu uočene pojave može se razviti sledeći algoritam. U svakoj dekompoziciji izlazne sekvence, odbirci sa parnim indeksima posebno se izračunavaju koristeći rekurzivno isti algoritam kao za celu sekvencu. Odbirci sa neparnim indeksima izračunavaju se koristeći dve radiks-4 transformacije nad preostalom polovinom ulaznih odbiraka, zbog toga što se na taj način dobija ušteda od oko 25% u broju množenja. Dakle, u prvom koraku dekompozicije na osnovu jednačina (5.35) i (5.36), imamo:

$$X[2k] = \sum_{n=0}^{N/2-1} \big(x[n] + x[n+N/2]\big) W_{N/2}^{nk}, \quad k = 0,1,\cdots, N/2-1 \tag{5.80}$$

$$X[4k+1] = \sum_{n=0}^{N/4-1} \Big\{\big(x[n] - x[n+N/2]\big) + j\big(x[n+N/4] - x[n+3N/4]\big)\Big\} W_N^n W_{N/4}^{kn}, \quad k = 0,1,\ldots, N/4-1 \tag{5.81}$$

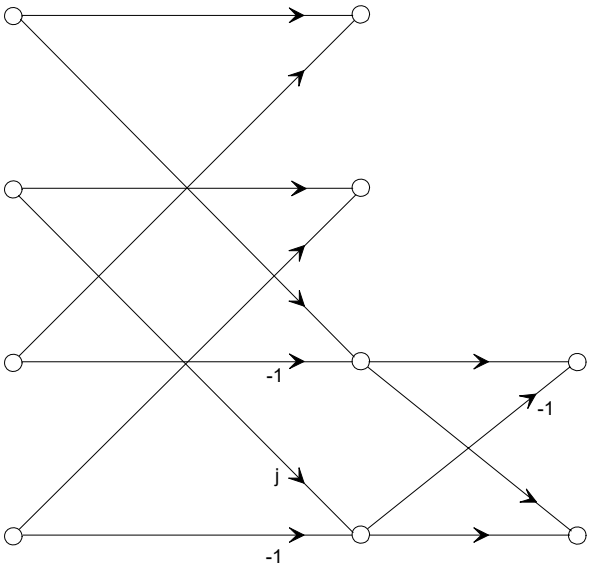
$$X[4k+3] = \sum_{n=0}^{N/4-1} \Big\{\big(x[n] - x[n+N/2]\big) - j\big(x[n+N/4] - x[n+3N/4]\big)\Big\} W_N^{3n} W_{N/4}^{kn}, \quad k = 0,1,\ldots, N/4-1 \tag{5.82}$$

Kao što se vidi, polovina izlaznih podataka se izračunava po radiks-4 a polovina po radiks-2 algoritmu. Zbog toga se ovakav tip dekompozicije naziva *dekompozicija sa dvostrukim radiksom*. Primenjujući isti postupak dekompozicije na parcijalne sekvence, dobija se DIF algoritam sa dvostrukim radiksom. Na sličan način može se izvesti i DIT algoritam sa dvostrukim radiksom. Važnost algoritma sa dvostrukim radiksom je u tome što od svih brzih DFT algoritama primenjenih na sekvencu dužine $N = 2^p$, algoritam sa dvostrukim radiksom ima najmanje netrivijalnih aritmetičkih operacija. Na primer, ako se za izvođenje kompleksnog množenja koristi $3/3$ algoritam, ukupan broj realnih množenja i sabiranja dat je izrazima:

$$N_M = N \log_2 N - 3N + 4 \tag{5.83}$$

$$N_A = 3N \log_2 N - 3N + 4 \tag{5.84}$$

Leptir operacija DIF algoritma sa dvostrukim radiksom prikazana je na slici 5.27. S obzirom na efikasnost algoritama sa dvostrukim radiksom, na slikama 5.28 i 2.29 su prikazane programske realizacije osnovnih verzija DIF i DIT algoritama. Svi postupci povećanja brzine izračunavanja koji su primenjivani kod radiks-2 ili radiks-4 algoritama primenljivi su i kod algoritma sa dvostrukim radiksom: specijalni leptiri, smeštaj trigonometrijskih funkcija u tabele, itd., tako da postoji veliki broj verzija efikasnih programskih realizacija DIF i DIT algoritma sa dvostrukim radiksom.



Slika 5.27 Leptir algoritma sa dvostrukim radiksom.

5.9.3 FFT ALGORITMI SA PROSTIM FAKTORIMA

Ako su faktori na koje se rastavlja dužina sekvence N u (5.63) uzajamno prosti (tj. nemaju zajednički delilac veći od 1), moguće je pomoću preslikavanja:

$$n = \langle An_1 + Bn_2 \rangle_N, \quad 0 \le n_1 \le N_1 - 1, \quad 0 \le n_2 \le N_2 - 1 \tag{5.85}$$

$$k = \langle Ck_1 + Dk_2 \rangle_N, \quad 0 \le k_1 \le N_1 - 1, \quad 0 \le k_2 \le N_2 - 1 \tag{5.86}$$

```
C   BRZA FURIJEOVA TRANSFORMACIJA, DIF ALGORITAM
C   SA DVOSTRUKIM RADIKSOM
C
C   XR    - REALNI DEO VEKTORA PODATAKA.
C   XI    - IMAGINARNI DEO VEKTORA PODATAKA.
C   M     = LOG2(N)
C
      SUBROUTINE DIFFFTSR(XR,XI,M)
      DIMENSION XR(1), XI(1)
      N=2**M
      L2 = 2 * N
      DO 30  K = 1,M-1
         L2 = L2 / 2
         L4 = L2 / 4
         PHI = 6.283185307 / L2
         ARG1 =  0.
         DO 20  J = 1,L4
            ARG3 = 3 * ARG1
            C1 = COS(ARG1)
            C3 = COS(ARG3)
            S1 = SIN(ARG1)
            S3 = SIN(ARG3)
            ARG1 = J * PHI
            ISTART = J
            INC = 2 * L2
            DOWHILE (ISTART.LT.N)
               DO 10  I = ISTART,N-1,INC
                  IP1 = I + L4
                  IP2 = IP1 + L4
                  IP3 = IP2 + L4
                  TR1 = XR(I) - XR(IP2)
                  TR2 = XR(IP1) - XR(IP3)
                  TI1 = XI(I) - XI(IP2)
                  TI2 = XI(IP1) - XI(IP3)
                  XR(I) = XR(I) + XR(IP2)
                  XI(I) = XI(I) + XI(IP2)
                  XR(IP1) = XR(IP1) + XR(IP3)
                  XI(IP1) = XI(IP1) + XI(IP3)
                  TI3 = TR1 - TI2
                  TR1 = TR1 + TI2
                  TI2 = TR2 - TI1
                  TR2 = TR2 + TI1
                  XR(IP2) = C1*TR1 - S1*TI2
                  XI(IP2) = -C1*TI2 - S1*TR1
                  XR(IP3) = C3*TI3 + S3*TR2
                  XI(IP3) = C3*TR2 - S3*TI3
   10          CONTINUE
               ISTART = 2 * INC - L2 + J
               INC = 4 * INC
            ENDDO
   20    CONTINUE
   30 CONTINUE
      ISTART = 1
      INC = 4
      DOWHILE (ISTART.LT.N)
         DO 40  I = ISTART,N,INC
            IP1 = I + 1
            TR1 = XR(I)
            XR(I) = TR1 + XR(IP1)
            XR(IP1) = TR1 - XR(IP1)
            TI1 = XI(I)
            XI(I) = TI1 + XI(IP1)
            XI(IP1) = TI1 - XI(IP1)
   40    CONTINUE
         ISTART = 2 * INC - 1
         INC = 4 * INC
      ENDDO
      RETURN
      END
```

Slika 5.28 Programska realizacija DIF algoritma sa dvostrukim radiksom.

```
C   BRZA FURIJEOVA TRANSFORMACIJA, DIT ALGORITAM
C   SA DVOSTRUKIM RADIKSOM
C
C   XR     - REALNI DEO VEKTORA PODATAKA.
C   XI     - IMAGINARNI DEO VEKTORA PODATAKA.
C   M      = LOG2(N)
C
        SUBROUTINE DITFFTSR(XR,XI,M)
        DIMENSION XR(1), XI(1)
        N=2**M
        ISTART = 1
        INC = 4
        DOWHILE (ISTART.LT.N)
           DO 10  I = ISTART,N,INC
              IP1 = I + 1
              TR1 = XR(I)
              XR(I) = TR1 + XR(IP1)
              XR(IP1) = TR1 - XR(IP1)
              TI1 = XI(I)
              XI(I) = TI1 + XI(IP1)
              XI(IP1) = TI1 - XI(IP1)
    10    CONTINUE
          ISTART = 2 * INC - 1
          INC = 4 * INC
        ENDDO
        L2 = 2
        DO 40  K = 2,M
           L2 = L2 * 2
           L4 = L2 / 4
           PHI = 6.283185307 / L2
           ARG1 =  0.
           DO 30  J = 1,L4
              ARG3 = 3 * ARG1
              C1 = COS(ARG1)
              C3 = COS(ARG3)
              S1 = SIN(ARG1)
              S3 = SIN(ARG3)
              ARG1 = J * PHI
              ISTART = J
              INC = 2 * L2
              DOWHILE (ISTART.LT.N)
                 DO 20  I = ISTART,N-1,INC
                    IP1 = I + L4
                    IP2 = IP1 + L4
                    IP3 = IP2 + L4
                    TR1 = XR(IP2)*C1 + XI(IP2)*S1
                    TI1 = XI(IP2)*C1 - XR(IP2)*S1
                    TR2 = XR(IP3)*C3 + XI(IP3)*S3
                    TI2 = XI(IP3)*C3 - XR(IP3)*S3
                    TR3 = TR1 + TR2
                    TR2 = TR1 - TR2
                    TR1 = TI1 + TI2
                    TI2 = TI1 - TI2
                    XR(IP2) = XR(I) - TR3
                    XR(I) = XR(I) + TR3
                    XR(IP3) = XR(IP1) - TI2
                    XR(IP1) = XR(IP1) + TI2
                    XI(IP2) = XI(I) - TR1
                    XI(I) = XI(I) + TR1
                    XI(IP3) = XI(IP1) + TR2
                    XI(IP1) = XI(IP1) - TR2
    20           CONTINUE
                 ISTART = 2 * INC - L2 + J
                 INC = 4 * INC
              ENDDO
    30    CONTINUE
    40  CONTINUE
        RETURN
        END
```

Slika 5.29 Programska realizacija DIT algoritma sa dvostrukim radiksom.

pogodnim izborom konstanti A, B, C i D potpuno eliminisati množenja sa rotacionim faktorima. Generalizacijom izraza (5.67) tako da uključi konstante A, B, C i D, dobijaju se uslovi koje treba zadovoljiti:

$$\langle AC \rangle_N = N_2, \ \langle BD \rangle_N = N_1, \ \langle AD \rangle_N = \langle BC \rangle_N = 0 \tag{5.87}$$

Određivanje skupa vrednosti konstanti A, B, C i D može se izvršiti primenom *Kineske teoreme ostataka* iz teorije brojeva. Jedno od mogućih rešenja je [B-30, B-35]:

$$A = N_2, \ B = N_1, \ C = N_2 \langle N_2^{-1} \rangle_{N_1}, \ D = N_1 \langle N_1^{-1} \rangle_{N_2} \tag{5.88}$$

gde izraz $< N_2^{-1} >_{N_1}$ označava inverzni element za N_2 u operaciji množenja koji je redukovan po modulu N_1. Na primer, $< 3^{-1} >_4 = 3$, jer je $< 3*3 >_4 = < 9 >_4 = 1$. Korišćenjem preslikavanja (5.85) i (5.86) i vrednosti koeficijenata iz (5.88) izraz za DFT (5.1) se može napisati u obliku:

$$X[k] = X\left[\left\langle N_2 \langle N_2^{-1} \rangle_{N_1} k_1 + N_1 \langle N_1^{-1} \rangle_{N_2} k_2 \right\rangle_N\right] =$$

$$= \sum_{n_2=0}^{N_2-1} \left\{ \sum_{n_1=0}^{N_1-1} x\left[\langle N_2 n_1 + N_1 n_2 \rangle_N\right] W_{N_1}^{k_1 n_1} \right\} W_{N_2}^{k_2 n_2} \tag{5.89}$$

što predstavlja pravu dvodimenzionu DFT bez rotacionih faktora. Redosled transformacija po vrstama i kolonama u (5.89) je proizvoljan.

Postupak dekompozicije se može dalje nastaviti ako je bar jedan od faktora složen broj. To je moguće jer je uslov dekompozicije da faktori budu samo uzajamno prosti brojevi, ali ne i da svaki od njih bude prost broj. Dekompozicija se može vršiti sve dok se broj N ne napiše u obliku (5.73) gde su svi faktori prosti brojevi. Broj množenja je smanjen i, umesto jednačinom (5.77), dat je formulom:

$$M(N) = N \sum_{i=1}^{p} \frac{M(N_i)}{N_i} \tag{5.90}$$

Ovaj tip algoritma prvi put je predložen u [G-14], ali je do njegove šire primene došlo tek posle publikovanja rada [B-30]. Algoritam je dobio naziv *algoritam sa prostim faktorima* (engl. *prime factor algorithm* - *PFA*) zbog toga što faktori na koje se rastavlja dužina sekvence moraju biti uzajamno prosti.

Smanjenje broja aritmetičkih operacija praćeno je i negativnim posledicama koje se ogledaju, pre svega, u povećanoj složenosti programske implementacije algoritma. Naročito su veliki problemi oko preuređivanja sekvenci i obezbeđivanja izračunavanja u mestu. Za razliku od programskih realizacija radiks-R algoritama, u programiranju algoritma sa prostim faktorima formiraju se posebni moduli za neke karakteristične faktore N_i kao što su 2, 3, 4, 5, 7, 8, 9, 11, 13, 16, 17, 19 i 25. Efikasnost realizacije direktno zavisi od efikasnosti korišćenih modula. Naravno, tako veliki broj modula utiče na dužinu programa, pa se u jednostavnijim realizacijama retko koristi više od tri do četiri različita modula, što sa druge strane smanjuje broj mogućih dužina ulaznih sekvenci. Za procenu broja operacija množenja, prema (5.90), potrebno je znati broj operacija u modulu za dužinu N_i, što je prikazano u Tabeli 5.3. U Tabeli 5.4 prikazane su moguće dužine ulazne sekvence i broj aritmetičkih operacija, kada je broj faktora u (5.73) manji ili jednak 4.

Tabela 5.3 Broj realnih množenja i sabiranja u DFT modulima za kompleksne ulazne sekvence dužine *N*.

N	Množenja	Sabiranja
2	0	2
3	2	6
4	0	8
5	5	17
7	8	36
8	2	26
9	10	42
11	20	84
13	20	94
16	10	74
17	35	157
19	38	186
25	66	210

PFA algoritam po efikasnosti spada među najbolje algoritme za izračunavanje DFT na računarima opšte namene. Na primer, broj množenja za $N = 1008$ (= 7*9*16) iznosi 5804, a broj sabiranja 29100. Ove brojeve treba porediti sa 10248 množenja i 25944 sabiranja kod najefikasnijeg radiks-4 algoritma za $N = 1024$, koji ima čak pet različitih leptirova. Programska realizacija PFA algoritma sa faktorima 2, 3, 4 i 5 prikazana je u Prilogu 1.

5.10 PRAKTIČNA PRIMENA DFT ALGORITMA

Zbog svoje izrazite računske efikasnosti, DFT algoritam je našao izuzetno veliku primenu u skoro svim područjima digitalne obrade signala. Posebno je značajna primena DFT algoritma u efikasnom izračunavanju linearne konvolucije o čemu je već bilo reči u odeljku 4.5.1. Međutim, značaj FFT algoritama sigurno ne bi bio tako veliki kada ne bi postojali isto tako efikasni metodi za izračunavanje inverzne DFT kao i efikasni algoritmi za direktnu i inverznu DFT sekvenci sa realnim elementima. Stoga će u ovom odeljku biti analizirani upravo takvi algoritmi kao i njihova aritmetička kompleksnost.

5.10.1 IZRAČUNAVANJE INVERZNE DFT

Posmatrajući izraze za direktnu i inverznu DFT, (4.16) i (4.17), uočavaju se dve razlike. Jedna od njih je postojanje multiplikativnog faktora $1/N$ u izrazu za inverznu DFT, dok je druga u znaku eksponenta rotacionih faktora. S obzirom na veliku sličnost izraza, za izračunavanje inverzne DFT mogu se razviti slični efikasni algoritmi kao za direktnu DFT. U literaturi su do sada za izračunavanje inverzne DFT predložena tri načina. U svakom od njih zanemaruje se multiplikativni faktor, koji se može uzeti u obzir u delu algoritma koji koristi rezultate inverzne DFT.

Prvi pristup je da se u program uvede jedna celobrojna promenljiva (kao na primer INV u programu sa slike 5.1) koja ima vrednost 0 za direktnu DFT i 1 za inverznu DFT. Time se malo komplikuje program i malo povećava aritmetička složenost jednostavnijih programa. Međutim, kod najefikasnijih FFT algoritama koji koriste više leptirova ovaj pristup se ne može primeniti.

Tabela 5.4 Broj realnih množenja i sabiranja za PFA algoritme sa kompleksnim ulaznim podacima.

$N = N_1N_2N_3N_4$	Množenja	Sabiranja
6 = 2*3*1*1	8	36
10 = 2*1*1*5	20	88
12 = 4*3*1*1	16	96
14 = 2*1*7*1	32	172
15 = 1*3*1*5	50	162
16 = 16*1*1*1	20	148
18 = 2*9*1*1	40	204
20 = 4*1*1*5	40	216
21 = 1*3*7*1	76	300
24 = 8*3*1*1	44	252
28 = 4*1*7*1	64	400
30 = 2*3*1*5	100	384
35 = 1*1*7*5	150	598
36 = 4*9*1*1	80	480
40 = 8*1*1*5	100	532
42 = 2*3*7*1	152	684
45 = 1*9*1*5	190	726
48 = 16*3*1*1	124	636
56 = 8*1*7*1	156	940
60 = 4*3*1*5	200	888
63 = 1*9*7*1	284	1236
70 = 2*1*7*5	300	1336
72 = 8*9*1*1	196	1140
80 = 16*1*1*5	260	1284
84 = 4*3*7*1	304	1536
90 = 2*9*1*5	380	1632
105 = 1*3*5*7	590	2214
112 = 16*1*7*1	396	2188
120 = 8*3*1*5	460	2076
126 = 2*9*7*1	568	2724
140 = 4*1*7*5	600	2952
144 = 16*9*1*1	500	2676
168 = 8*3*7*1	692	3492
180 = 4*9*1*5	760	3624
210 = 2*3*7*5	1180	4848
240 = 16*3*1*5	1100	4812
252 = 4*9*7*1	1136	5952
280 = 8*1*7*5	1340	6604
315 = 1*9*7*5	2050	8322
336 = 16*3*7*1	1636	7908
360 = 8*9*1*5	1700	8148
420 = 4*3*7*5	2360	10536
504 = 8*9*7*1	2524	13164
560 = 16*1*7*5	3100	14748
630 = 2*9*7*5	4100	17904
720 = 16*9*1*5	3940	18726
840 = 8*3*7*5	5140	23172
1008 = 16*9*7*1	5804	29100
1260 = 4*9*7*5	8200	38328
1680 = 16*3*7*5	11540	50964
2520 = 8*9*7*5	17660	82956
5040 = 16*9*7*5	39100	179772

Drugi pristup je zasnovan na relaciji:

$$x^*[n] = \left[\frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn} \right]^* = \frac{1}{N} \sum_{k=0}^{N-1} X^*[k] W_N^{kn} \tag{5.91}$$

odnosno, za izračunavanje inverzne DFT može se iskoristiti isti program kao za direktnu DFT, samo što se kao ulazna sekvenca koristi $X^*[k]$ umesto $X[k]$ i što se za izlaznu sekvencu dobija $x^*[n]$ umesto $x[n]$. To praktično znači da se elementima imaginarnog dela sekvence $X[k]$ promeni znak i da u dobijenoj izlaznoj sekvenci, $x^*[n]$, treba promeniti znak elementima imaginarnog dela sekvence. Operacije promene znaka vrlo malo povećavaju složenost algoritma za izračunavanje inverzne DFT.

Treći, najnoviji, pristup izračunavanju inverzne DFT nedavno je opisan u [D-14]. Naime, iz (5.91) se dobija:

$$jx^*[n] = \frac{1}{N} \sum_{k=0}^{N-1} jX^*[k] W_N^{kn} \tag{5.92}$$

odakle sledi:

$$x[n] = j \left[\frac{1}{N} \sum_{k=0}^{N-1} jX^*[k] W_N^{kn} \right]^* \tag{5.93}$$

Pošto je:

$$jX^*[k] = \mathrm{Im}\big(X[k]\big) + j\,\mathrm{Re}\big(X[k]\big) \tag{5.94}$$

za izračunavanje inverzne DFT može se koristiti bilo koji FFT potprogram, ali pri njegovom pozivu treba zameniti mesta realnom i imaginarnom delu sekvence $X[k]$. Time se po završetku rada programa dobijaju korektni vektori realnog i imaginarnog dela sekvence $x[n]$, ali takođe u izmenjenom redosledu. Ovaj pristup, kao što se vidi, ne zahteva nikakve dodatne operacije. Jedini zahtev je da se realni i imaginarni delovi korišćenih sekvenci smeštaju u dva realna vektora, a ne u jedan kompleksni vektor.

5.10.2 IZRAČUNAVANJE DFT DVE REALNE SEKVENCE

Svi do sada opisani algoritmi za efikasno izračunavanje DFT podrazumevaju da elementi ulazne i izlazne sekvence predstavljaju kompleksne brojeve. Međutim, u većini praktičnih slučajeva ulazna sekvenca predstavlja niz realnih brojeva jer se dobija digitalizacijom realne kontinualne funkcije. U takvim slučajevima, zbog osobina opisanih jednačinama (4.26) - (4.30), potrebno je izračunati samo polovinu izlaznih odbiraka čime se broj potrebnih operacija smanjuje za 50%. Veće uštede u broju operacija moguće je postići korišćenjem metoda opisanih u ovom i narednom odeljku.

Ukoliko se efikasnim FFT algoritmom izračunava DFT realne sekvence, u procesu izračunavanja pojaviće se veliki broj množenja i sabiranja u kojima je jedan od argumenata jednak nuli. To je posledica činjenice da je imaginarni deo ulazne sekvence jednak nuli. Pretpostavimo da imamo dve realne sekvence iste dužine N, $x_1[n]$ i $x_2[n]$. Od njih se može formirati kompleksna sekvenca $x[n]$ na sledeći način:

$$x[n] = x_1[n] + jx_2[n], \quad n = 0,1,\ldots,N-1 \tag{5.95}$$

Zbog osobine linearnosti DFT, iz (4.23) i (5.95) se dobija:

$$X[k] = X_1[k] + jX_2[k], \quad k = 0,1,\ldots,N-1 \tag{5.96}$$

Iz (5.95) se takođe dobija:

$$x_1[n] = \frac{x[n] + x^*[n]}{2} \tag{5.97}$$

$$x_2[n] = \frac{x[n] - x^*[n]}{2j} \tag{5.98}$$

odakle se korišćenjem osobine linearnosti DFT i relacije (4.25) prema kojoj je DFT od $x^*[n]$ jednaka $X^*[N-k]$, konačno dobija:

$$X_1[k] = \frac{X[k] + X^*[N-k]}{2} \tag{5.99}$$

$$X_2[k] = \frac{X[k] - X^*[N-k]}{2j} \tag{5.100}$$

Dakle, formiranjem kompleksne sekvence kombinovanjem dve realne sekvence, mogu se izračunati DFT obe sekvence sa brojem operacija koji je potreban za izračunavanje DFT jedne sekvence uvećanim za 2*N* kompleksnih sabiranja. Ovaj metod je primenljiv uvek kada treba naći DFT više sekvenci iste dužine.

5.10.3 IZRAČUNAVANJE DFT REALNE SEKVENCE DUŽINE 2*N*

Metod opisan u prethodnom odeljku nije primenljiv kada treba odrediti DFT samo jedne realne sekvence. Međutim i tada je moguće ostvariti velike uštede u broju izračunavanja ako je broj elemenata u sekvenci paran, tj. dužina sekvence je 2*N*. To je naravno čest slučaj u praksi jer je dužina sekvence najčešće neki stepen broja 2. U takvom slučaju mogu se definisati parcijalne sekvence $x_1[n]$ i $x_2[n]$ dužine *N* na sledeći način:

$$x_1[n] = f[2n], \quad n = 0,1,\ldots,N-1 \tag{5.101}$$

$$x_2[n] = f[2n+1], \quad n = 0,1,\ldots,N-1 \tag{5.102}$$

gde je $f[n]$ originalna sekvenca dužine 2*N*. Od parcijalnih sekvenci $x_1[n]$ i $x_2[n]$ može se formirati nova kompleksna sekvenca prema (5.95) i primeniti isti postupak za određivanje $X_1[k]$ i $X_2[k]$ kao u (5.99) i (5.100). Na kraju, istim postupkom kao u izvođenju DIT algoritma, dobija se:

$$F[k] = \sum_{n=0}^{N-1} f[2n] W_{2N}^{2nk} + \sum_{n=0}^{N-1} f[2n+1] W_{2N}^{(2n+1)k} =$$

$$= \sum_{n=0}^{N-1} x_1[n] W_N^{nk} + W_{2N}^k \sum_{n=0}^{N-1} x_2[n] W_N^{nk}, \quad k = 0,1\ldots,N-1 \tag{5.103}$$

odnosno:

$$F[k] = X_1[k] + W_{2N}^k X_2[k], \quad k = 0,1,\ldots,N-1 \tag{5.104}$$

$$F[k+N] = X_1[k] - W_{2N}^k X_2[k], \quad k = 0,1,\ldots,N-1 \tag{5.105}$$

Dakle, broj operacija za DFT realne sekvence od $2N$ elemenata jednak je broju operacija koji je potreban za jednu DFT sekvence od N elemenata uvećanom za N kompleksnih množenja i $2N$ kompleksnih sabiranja potrebnih za dodatna izračunavanja u (5.104) i (5.105).

5.10.4 EFIKASNO IZRAČUNAVANJE LINEARNE KONVOLUCIJE POMOĆU DFT

U odeljku 4.5.1 su objašnjeni principi izračunavanja linearne konvolucije korišćenjem DFT. U ovom odeljku će biti analizirana aritmetička složenost takvog metoda izračunavanja i izvršiće se poređenje sa direktnim metodom izračunavanja na osnovu jednačine (1.30).

Neka su poznati ulazna sekvenca $x[n]$, dužine N_x, i impulsni odziv sistema $h[n]$, dužine N_h. Izlazna sekvenca data je linearnom konvolucijom sekvenci $x[n]$ i $h[n]$, tj. $y[n] = x[n] * h[n]$. Dužina izlazne sekvence je $N_y = N_x + N_h - 1$. Ako se, kao u odeljku 4.5.1 sekvence $x[n]$ i $h[n]$ dopune potrebnim brojem nula da bi se izjednačile vrednosti linearne i cirkularne konvolucije, onda sve tri sekvence imaju istu dužinu $N \geq N_y$. Međutim, za broj elemenata N pogodno je izabrati neku vrednost koja omogućava izračunavanje DFT prema nekom efikasnom FFT algoritmu, na primer $N = 2^p$. Dakle ukupan broj operacija koji je potreban za izračunavanje linearne konvolucije pomoću DFT sastoji se iz tri dela:

1. Broj operacija za izračunavanje $X[k]$ i $H[k]$ iz $x[n]$ i $h[n]$,
2. Broj operacija za izračunavanje $Y[k] = X[k]H[k], \quad k = 0,1,\ldots,N-1$,
3. Broj operacija za izračunavanje $y[n]$ iz $Y[k]$.

Ukoliko se za izračunavanje direktne i inverzne DFT koristi osnovna verzija DIT ili DIF FFT radiks-2 algoritma, potreban broj aritmetičkih operacija u tačkama 1 i 3 dat je izrazima (5.27) do (5.29). Za izvođenje tačke 2, potrebno je N kompleksnih množenja, odnosno $4N$ realnih množenja i $2N$ realnih sabiranja. Dakle, imamo:

$$N_M = 3(2N\log_2 N) + 4N = 6N\log_2 N + 4N \tag{5.106}$$

$$N_A = 3(3N\log_2 N) + 2N = 9N\log_2 N + 2N \tag{5.107}$$

Ove vrednosti treba uporediti sa brojem operacija koji je potreban za izračunavanje izraza (1.30). Ako se koristi najefikasniji metod za direktno izračunavanje izraza (1.30) u kome su eliminisana sva množenja sa nulom, onda je potreban broj realnih množenja i sabiranja:

$$N'_M = N_x N_h \tag{5.108}$$

$$N'_A = (N_x - 1)(N_h - 1) \tag{5.109}$$

Uzmimo, kao primer slučaj, $N_x = 512$, $N_h = 512$, $N = 1024$. Onda je prema (5.106) i (5.107) $N_M = 65536$ i $N_A = 94208$, dok je prema (5.108) i (5.109) $N'_M = 262144$ i $N'_A = 261121$. Dakle, ako se izračunavanje linearne konvolucije u ovom slučaju izvodi korišćenjem DFT, potrebno je svega 25% množenja i 36% sabiranja u odnosu na broj operacija kod direktne metode. Ukoliko se primeni neki efikasniji FFT algoritam ušteda može biti i veća.

Međutim, u praksi se prilično često pojavljuje slučaj kada su dužine sekvenci $x[n]$ i $h[n]$ jako različite. Takav je slučaj, recimo, kod obrade govornih signala kada je ulazna sekvenca $x[n]$ vrlo dugačka a impulsni odziv sistema $h[n]$ znatno kraći, tj. $N_h << N_x$. Tada se jedna od sekvenci,

a ponekad i obe, produžava velikim brojem nula, da bi se dobila pogodna vrednost za N. Time se znatno povećava broj operacija u sva tri koraka izračunavanja i znatno smanjuje efikasnost izračunavanja konvolucije.

Efikasno izračunavanje linearne konvolucije dve sekvence izrazito različitih dužina može se ostvariti ako se duža sekvenca podeli na segmente, pa se onda formiraju parcijalne konvolucije između segmenata duže sekvence i kraće sekvence. Takav metod poznat je pod nazivom *blok konvolucija*. Postoje dve varijante blok konvolucije. U jednoj varijanti segmenti ulazne sekvence se ne preklapaju dok se kod druge varijante preklapaju. U oba slučaja se uočava veoma dobra osobina blok konvolucije da izračunavanje konvolucije može započeti odmah pošto je prvi segment ulazne sekvence na raspolaganju, tj. da cela sekvenca ne mora biti na raspolaganju. Ovo je od ogromne važnosti u obradi signala u realnom vremenu kada su velika kašnjenja nedopustiva.

5.10.4.1 Blok konvolucija kada se segmenti ulazne sekvence ne preklapaju

Neka se ulazna sekvenca $x[n]$ podeli na S parcijalnih sekvenci dužine L prema izrazu:

$$x[n] = \sum_{i=0}^{S-1} x_i[n], \quad n = 0,1,\ldots,SL-1 \tag{5.110}$$

gde je:

$$x_i[n] = \begin{cases} x[n], & iL \le n \le (i+1)L-1 \\ 0, & \text{za druge vrednosti } n \end{cases} \tag{5.111}$$

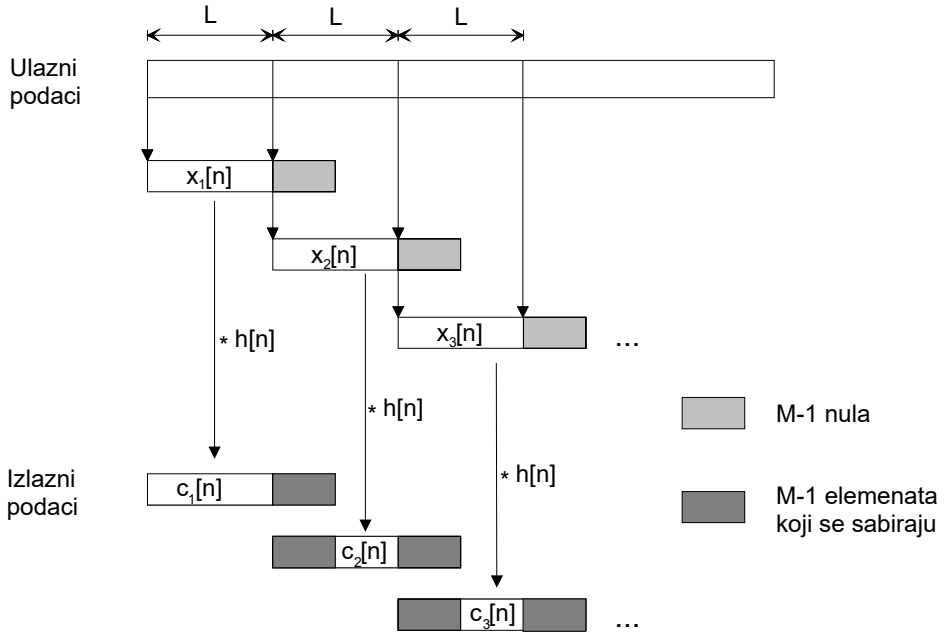
Onda se izraz za linearnu konvoluciju svodi na:

$$y[n] = \sum_{m=0}^{N_h-1} \sum_{i=0}^{S-1} x_i[n-m]h[m] = \sum_{i=0}^{S-1} \sum_{m=0}^{N_h-1} x_i[n-m]h[m] = \sum_{i=0}^{S-1} c_i[n] \tag{5.112}$$

gde $c_i[n]$ predstavljaju linearne konvolucije segmenata ulazne sekvence $x[n]$ i sekvence $h[n]$. Svaka parcijalna konvolucija ima $L+N_h-1$ elemenata koji su različiti od nule, tj. parcijalne konvolucije $c_i[n]$ su veće dužine od segmenata ulazne sekvence. Iz jednačine (5.112) se vidi da se elementi parcijalnih konvolucija sabiraju kada imaju isti indeks. Međutim neki elementi izlazne sekvence dobijaju se direktno iz onih delova parcijalnih konvolucija koji se ne preklapaju sa drugim parcijalnim konvolucijama. Zbog toga je ovaj metod blok konvolucije poznat i pod nazivom *preklopi i saberi* (engl. *overlap-and-add*). Princip formiranja izlazne sekvence kod blok konvolucije sa segmentiranjem ulazne sekvence bez preklapanja prikazan je na slici 5.30.

Procena složenosti izračunavanja algoritama za izračunavanje konvolucije obično se izvodi određivanjem broja aritmetičkih operacija potrebnih za dobijanje jednog izlaznog odbirka. Kod opisanog postupka formiranje parcijalnih linearnih konvolucija može se izvršiti korišćenjem FFT algoritama na način koji je opisan u prethodnom izlaganju. Da bi se izjednačili rezultati cirkularne i linearne konvolucije kao i da bi se mogao primeniti neki efikasni FFT algoritam sekvenca $h[n]$, kao i parcijalne sekvence $x_i[n]$, moraju biti iste dužine $N \ge L+N_h-1$, koja je pogodna za primenu izabranog FFT algoritma. Prvo se izračunava DFT sekvence $h[n]$, proširene na kraju sa elementima nulte vrednosti. Zatim se S puta izračunava DFT parcijalnih sekvenci $x_i[n]$ koje su takođe proširene nultim odbircima na kraju do dužine N. Posle množenja $H[k]$ sa $X_i[k]$ član po član dobija se $C_i[k]$, a iz $C_i[k]$ se inverznom DFT dobijaju parcijalne konvolucije $c_i[n]$, koje treba sabrati radi određivanja izlazne sekvence. Broj operacija po jednom izlaznom odbirku dobija se

korišćenjem izraza (5.27) do (5.29), pri čemu se obično ne uzima u obzir broj operacija za određivanje $H[k]$, jer se ta operacija izvodi samo jednom.

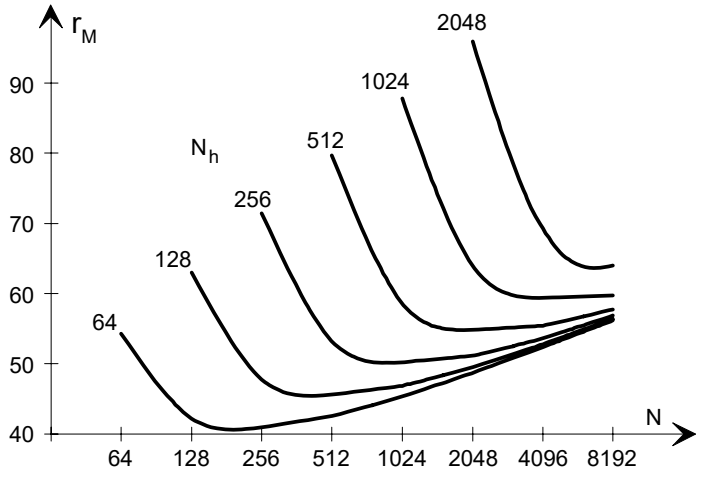


Slika 5.30 Formiranje izlazne sekvence kod blok konvolucije bez preklapanja segmenata ulazne sekvence.

Broj množenja po jednom odbirku za blok konvoluciju po metodu preklopi i saberi iznosi:

$$r_M(N) = \frac{4N\log_2 N + 4N}{L} = \frac{4N(\log_2 N + 1)}{N - N_h + 1} \approx \frac{2^{p+2}(p+1)}{2^p - N_h} \tag{5.113}$$

odakle se vidi da za dato N_h postoji optimalna vrednost N (odnosno L) za koju je potreban minimalni broj množenja. Na slici 5.31 prikazana je zavisnost $r_M(N)$ sa dužinom impulsnog odziva N_h kao parametrom. Sa slike se vidi da se kriva u oblasti minimuma malo menja i da je optimalna vrednost $N \approx 5N_h$.



Slika 5.31 Broj množenja po izlaznom odbirku u funkciji dužine proširenog bloka *N*.

Interesantno je uporediti i broj elemenata po izlaznom odbirku sa slučajem kada se konvolucija izračunava direktno, bez primene DFT. Tada je, prema (5.108), prosečan broj množenja po izlaznom odbirku jednak N_h, pri čemu su zanemarene male uštede koje se mogu

ostvariti pri izračunavanju početnih i krajnjih izlaznih odbiraka. Korišćenjem slike 5.31, može se dobiti i ova važna informacija. Sa slike 5.31 lako se uočava da je $r_M(N) < N_h$ za $N_h > 64$, što znači da za $N_h < 64$ treba koristiti direktni metod izračunavanja konvolucije.

5.10.4.2 Blok konvolucija kada se segmenti ulazne sekvence preklapaju

Alternativni postupak za izračunavanje blok konvolucije zasnovan je na takvoj podeli ulazne sekvence koja omogućava da se izlazna sekvenca formira direktno, uzimajući samo one elemente parcijalnih konvolucija koji predstavljaju korektan rezultat. Kao što je poznato, prelazni režim pri konvoluciji ulazne sekvence sa impulsnim odzivom $h[n]$ dužine $N_h - 1$ traje $N_h - 1$ odbiraka. Zbog toga se segmentacija ulazne sekvence vrši tako da se segmenti preklapaju za $N_h - 1$ elemenata. U svakom segmentu postoji L novih elemenata, tako da je dužina svakog segmenta $L + N_h - 1$. Prvi segment se proširuje sa $N_h - 1$ nula sa leve strane. Dakle, ima se:

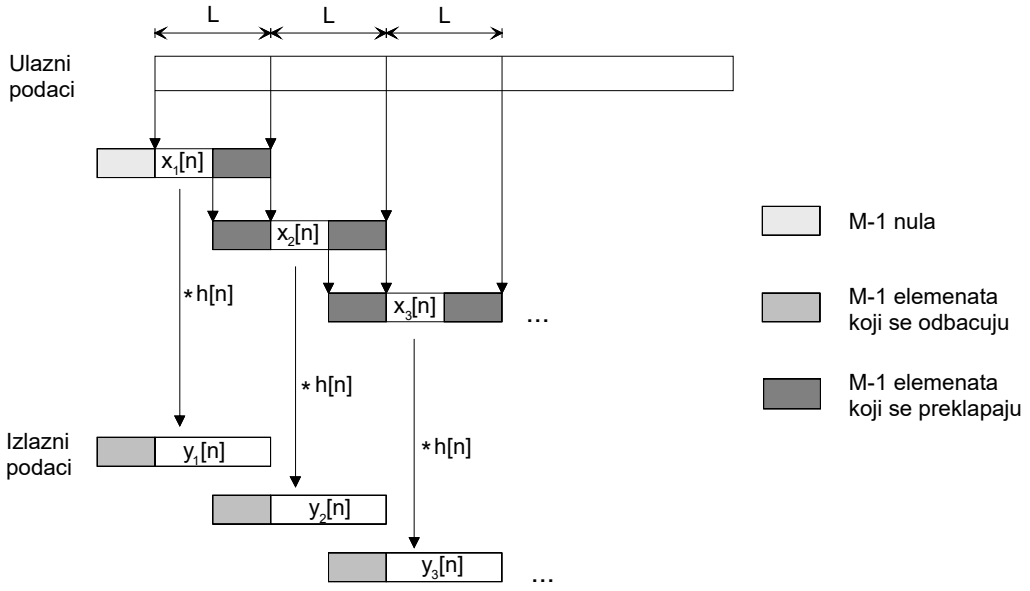
$$x_i[n] = \begin{cases} x[n], & iL - N_h + 1 \le n \le (i+1)L - 1 \\ 0, & \text{za druge vrednosti } n \end{cases} \tag{5.114}$$

Onda je:

$$y[n] = \sum_{i=0}^{S-1} y_i[n] \tag{5.115}$$

gde su $y_i[n]$ rezultati parcijalnih cirkularnih konvolucija iz kojih je izbačeno prvih $N_h - 1$ elemenata koji bi zbog preklapanja izazvali grešku u konačnom rezultatu. Blok konvolucija kada se ulazni segmenti preklapaju poznata je i pod imenom *selektuj i sačuvaj* (engl. *select-save* ili *overlap-save*), a grafički je prikazana na slici 5.32.

Opisani metod blok konvolucije takođe je veoma efikasan. Parcijalne konvolucije se izračunavaju nekim od efikasnih FFT algoritama. Iako za formiranje izlazne sekvence nisu potrebne dodatne operacije, metod nije efikasniji od metoda preklopi i saberi jer su zbog preklapanja ulazni segmenti nešto duži. Rezultati poređenja metoda blok konvolucije zavise od karakteristika FFT algoritma koji se koristi u izračunavanju parcijalnih konvolucija, kao i od vrednosti L i N_h.



Slika 5.32 Formiranje izlazne sekvence kod blok konvolucije sa preklapanjem segmenata ulazne sekvence.

5.11 IZRAČUNAVANJE DFT POMOĆU KONVOLUCIJE

U prethodnom odeljku opsežno su razmatrani metodi indirektnog izračunavanja linearne konvolucije korišćenjem DFT sa ciljem da se poveća efikasnost izračunavanja. Međutim, pokazano je da postoje i slučajevi kada je efikasnije direktno izračunavanje konvolucije. Šta više, postoje i slučajevi kada je efikasnije indirektno izračunati DFT koristeći konvolucionu relaciju. Ova problematika, koja je na prvi pogled u kontradikciji sa zaključcima iz prethodnog odeljka, biće predmet narednog izlaganja.

Posmatrajmo niz $x[n]$ dužine N i njegovu Furijeovu transformaciju $X(e^{j\Omega})$. Posmatrajmo proces izračunavanja M odbiraka kontinualne funkcije $X(e^{j\Omega})$, pri čemu se tačke izračunavanja nalaze na jediničnom krugu sa jednakim ugaonim rastojanjima, tj.

$$\Omega_n = \Omega_0 + n\Delta\Omega, \quad n = 0,1,...,M-1 \tag{5.116}$$

gde su, za razliku od DFT, broj tačaka izračunavanja M, početna učestanost Ω_0 i njen inkrement $\Delta\Omega$ proizvoljni. U stvari, DFT predstavlja specijalni slučaj ovakvog procesa izračunavanja gde je $M = N$, $\omega_0 = 0$ i $\Delta\omega = 2\pi/N$. Iz definicione relacije za Furijeovu transformaciju date jednačinom (2.5), dobija se:

$$X(e^{j\Omega_n}) = \sum_{k=0}^{N-1} x[k]e^{-j\Omega_n k}, \quad n = 0,1,...,M-1 \tag{5.117}$$

Ako se uvede pomoćna promenljiva:

$$W = e^{-j\Delta\Omega} \tag{5.118}$$

onda se iz jednačine (5.117) dobija:

$$X(e^{j\Omega_n}) = \sum_{k=0}^{N-1} x[k]e^{-j\Omega_0 k}W^{nk}, \quad n = 0,1,...,M-1 \tag{5.119}$$

Korišćenjem identiteta:

$$nk = \tfrac{1}{2}[n^2 + k^2 - (n-k)^2] \tag{5.120}$$

izraz (5.119) se može prevesti u oblik:

$$X(e^{j\Omega_n}) = \sum_{k=0}^{N-1} x[k]e^{-j\Omega_0 k}W^{k^2/2}W^{n^2/2}W^{-(n-k)^2/2}, \quad n = 0,1,...,M-1 \tag{5.121}$$

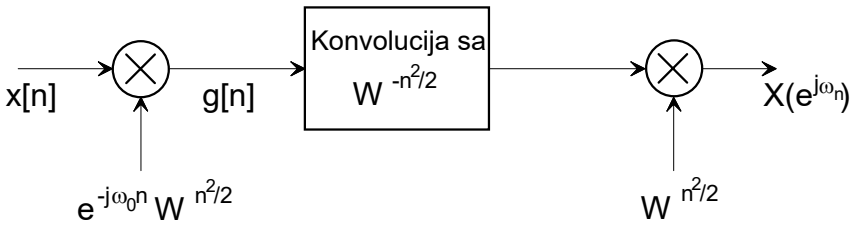
Uvođenjem pomoćne funkcije:

$$g[k] = x[k]e^{-j\Omega_0 k}W^{k^2/2} \tag{5.122}$$

izraz (5.121) se konačno dovodi na oblik:

$$X(e^{j\Omega_n}) = W^{n^2/2}\sum_{k=0}^{N-1} g[k]W^{-(n-k)^2/2}, \quad n = 0,1,...,M-1 \tag{5.123}$$

koji očigledno predstavlja konvoluciju sekvence $g[n]$ sa sekvencom $W^{-n^2/2}$, posle čega se rezultat konvolucije množi sekvencom $W^{n^2/2}$. Kao rezultat konvolucije i množenja dobijaju se traženi odbirci u frekvencijskom domenu $X(e^{j\Omega_n})$. Proces određivanja odbiraka u frekvencijskom domenu pomoću konvolucije prikazan je na slici 5.33.



Slika 5.33 Blok dijagram chirp transform algoritma.

Opisani algoritam se u engleskoj literaturi naziva *chirp transform algorithm* - *CTA* (engl. chirp = cvrkut). U čisto softverskoj implementaciji CTA nema nikakve prednosti nad ranije opisanim FFT algoritmima u pogledu aritmetičke složenosti. Šta više, za izračunavanje konvolucije u okviru CT algoritma obično se koristi DFT. Međutim, velika prednost CT algoritma nad DFT je u tome što je mnogo fleksibilniji. Prvo, CTA omogućava izračunavanje spektralnih odbiraka i kada je $M \neq N$. Druga prednost CT algoritma je što M i N ne moraju biti složeni brojevi što je, kao što je već pokazano, neophodan uslov za efikasno izračunavanje DFT. Treće, početna učestanost ω_0 je sasvim proizvoljna. U slučajevima kada je na raspolaganju i hardverska podrška za efikasno izračunavanje konvolucije, CTA često predstavlja bolju alternativu za izračunavanje spektra. Takav je na primer slučaj ako se za procesiranje diskretnih signala koriste CCD (engl. charge coupled devices) ili SAW (engl. surface acoustic wave) naprave. Osim toga, u poslednje vreme pojavljuju se i digitalni integrisani procesori pogodni za izračunavanje konvolucije, pa CTA sve više dobija na značaju.