Primerjava in napoved časa izvajanja rekurzivnih sekvenčnih algoritmov za izračun Fourierjeve transformacije

Žiga Zorman, Tomaž Dobravec

Fakulteta za računalništvo in informatiko
Univerza v Ljubljani, Večna pot 113, 1000 Ljubljana, Slovenija
E-pošta: ziga.zorman@gmail.com, tomaz.dobravec@fri.uni-lj.si

Comparison and estimation of execution time of recursive sequential Fourier transform algorithms

The Discrete Fourier Transform is one of the most important tools in digital signal processing. A very large number of articles have been written about these algorithms, however, only a few of them are comparing their speed and efficiency.
The aim of this paper is to compare the time complexity of various sequential Fast Fourier Transform algorithms [1], implemented in recursive fashion. The paper also presents the formula to estimate the execution time of these algorithms, which is based on the number of required recursive calls.

1 Uvod

V tem članku se bomo osredotočili na primerjavo rekurzivnih sekvenčnih algoritmov za izračun hitre Fourierjeve transformacije [1]. Primerjali bomo čas izvajanja ter predstavili napoved časa izvajanja algoritma v odvisnosti od števila rekurzivnih klicev. Zanimajo nas predvsem trije algoritmi po metodi Cooley-Tukey [2], to so:

- algoritem z osnovo 2,
- algoritem z osnovo 4,
- algoritem z deljeno osnovo (2 in 4).

Za vse zgoraj naštete algoritme smo napisali dve različici, prva uporablja decimacijo v časovnem prostoru, druga pa decimacijo v frekvenčnem prostoru. Za primerjavo smo analizirali tudi izvajanje navadne diskretne Fourierjeve transformacije (DFT) po definiciji.

2 Algoritmi

V tem razdelku bomo na kratko predstavili implementirane algoritme za izračun Fourierjeve transformacije (FT) in število kompleksnih seštevanj in množenj, ki jih potrebujejo. Pri vsakem od algoritmov bomo med seboj primerjali izvedbi z decimacijo v časovnem prostoru (DIT) in decimacijo v frekvenčnem prostoru (DIF).
Z uporabo Eulerjeve formule smo po spodnji enačbi (1) ob vsaki transformaciji izračunali potence N-tega primitivnega korena enote.

$$\omega_N^k = e^{-i\frac{2\pi}{N}k} = \cos(\frac{2\pi}{N}k) - i\sin(\frac{2\pi}{N}k);\ k, N \in \mathbb{N}_0 \tag{1}$$

Teh vrednosti nismo izračunali že v naprej in jih tudi nismo shranili v kakršnokoli vrsto tabele za kasnejšo ponovno uporabo.
Izvorno kodo implementiranih algoritmov in celoten ALGator projekt smo shranili tudi v javno dosegljiv GitHub repozitorij [6].

2.1 Diskretna Fourierjeva transformacija

Definicija 2.1 *Diskretna Fourierjeva transformacija (DFT)* $\overline{X}_k$ *vhodnega zaporedja* x_k *dolžine* N *je podana z naslednjo enačbo [1]:*

$$\overline{X}_k = \sum_{r=0}^{N-1} x_r \omega_N^{rk};\ k = 0, 1, \ \ldots,\ N-1. \tag{2}$$

Pri tem je ω_N *N-ti primitivni koren enote.*

Ta algoritem smo implementirali striktno po zgornji enačbi (2) zato, da bomo lahko meritve ostalih algoritmov primerjali z definicijo DFT.

2.2 Osnova 2, osnova 4 in deljena osnova

Algoritem Cooley-Tukey (1965), ki sta ga ponovno odkrila James William Cooley in John Tukey, razdeli problem velikosti N na več podproblemov. V primeru algoritma z osnovo dva na dva podproblema, z osnovo 4 pa na štiri podprobleme enakih velikosti. Algoritem z deljeno osnovo pa na 3 podprobleme, enega velikosti $\frac{N}{2}$ ter dva velikosti $\frac{N}{4}$, kjer je N dolžina vhodnega problema. Tako da v primeru algoritma z osnovo 2 velja: $N = N_1 \cdot N_2$, kjer sta N_1 število podproblemov in N_2 velikost podproblemov. To nam omogoči, da izračun ene DFT razbijemo na več manjših DFT in zaradi posebnih lastnosti N-tega primitivnega korena enote [4], zmanjšamo čas izračuna. V našem primeru na $O(N \cdot \log N)$.
Glavno idejo omenjenih algoritmov smo razdelili na tri korake [2]:

1. korak: razdelimo problem na več manjših podproblemov.

2. korak: rešimo vsak podproblem z enakim algoritmom.

3. korak: sestavimo rešitev začetnega problema iz rešitev vseh podproblemov.

Predpostavimo, da želimo izračunati DFT $\overline{X}_k$ vhodnega zaporedja x_k dolžine N, kjer velja $N = 2^t; t \in \mathbb{N}$. Tedaj lahko izberemo za $N_1 = 2$ in $N_2 = 2^{t-1}$, kar pomeni, da smo razdelili vhodno množico podatkov na dve množici velikosti $\frac{N}{2}$. Ta način razdelitve vhodnih podatkov na dve podmnožici imenujemo algoritem z osnovo 2 (angl. radix-2). Če izberemo $N_1 = 4$ in $N_2 = 2^{t-2}$, tako vhodni problem razdelimo na 4 manjše podprobleme velikosti $\frac{N}{4}$ in dobimo algoritem z osnovo 4. Poleg metode razdelitve vhodnega problema se algoritmi z različnimi osnovami, zaradi lastnosti N-tega primitivnega korena enote, razlikujejo tudi v številu kompleksnih množenj.
Obravnavali bomo dve vrsti vseh treh naštetih algoritmov in sicer decimacijo v časovnem prostoru ter decimacijo v frekvenčnem prostoru. Razlikujeta se po dodeljevanju elementov vhodnega zaporedja x_k dolžine N v podmnožice. Na primer algoritem z osnovo 2: DIT razdeli vhodno zaporedje na sodi (x_{2m}) in lihi (x_{2m+1}) del, DIF pa na prvo (x_k) in drugo ($x_{k+\frac{N}{2}}$) polovico.
Podrobnejšo razlago o prostorih decimacije in algoritmih najdete v diplomskem delu [5].

3 Orodje za primerjavo algoritmov in izvajalno okolje

Za testiranje in analizo algoritmov smo uporabili sistem ALGator [3], ki so ga razvili na Fakulteti za računalništvo in informatiko v Ljubljani. ALGator je implementiran v programskem jeziku Java. Razvijalcu omogoča enostavno integracijo ter izvajanje algoritmov na izbranih testnih podatkih in analiziranje rezultatov izvajanj.
Sistem omogoča dodajanje in upravljanje s poljubnim številom projektov. V okviru enega projekta je definiran problem, testne množice vhodnih podatkov ter način reševanja nalog tega problema. Projekt lahko vsebuje poljubno število algoritmov, ki naloge rešujejo na predpisan način. Med seboj lahko na enostaven način primerjamo algoritme istega projekta.
Vse meritve smo izvedli na sistemu z naslednjimi specifikacijami:

- procesor Inter Core i5 2.6 GHz z dvema jedroma,
- 10 GB delovnega pomnilnika,
- operacijski sistem Microsoft Windows 10 Proffesional.

4 Meritve

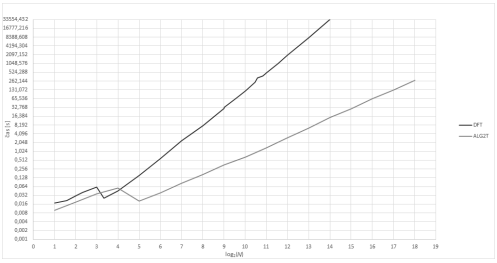
4.1 Testni primeri

Testni primeri so sestavljeni iz vhodnega zaporedja realnih števil in že izračunane transformacije - rešitve. Na tak način smo najenostavneje preverili, če je izračunan rezultat posameznega algoritma pravilen. Seveda pri izvajanju algoritma časa preverjanja rezultata nismo upoštevali.

Zaradi časovne zahtevnosti smo algoritme testirali na dolžini vhodnih podatkov od 2^2 pa do 2^{18}, izjema je bila navadna DFT po definiciji, pri kateri je bila maksimalna dolžina vhodnih podatkov 2^{14}.

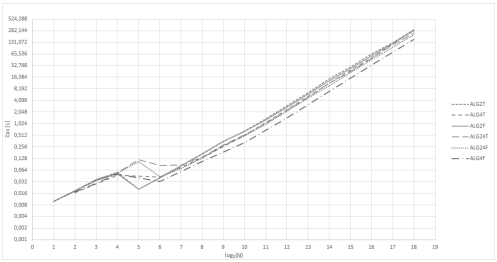
4.2 Izmerjen čas izvajanja

Ker vemo, da sistem ALGator na posameznem testnem primeru meri čas izvajanja T-krat (kolikokrat je podano v konfiguracijski datoteki testne množice) in da prihaja do razlik med prvim izvajanjem algoritma in nadaljnjimi, so nas zanimale tudi razlike med maksimalnim, minimalnim, povprečnim in srednjo vrednostjo časa izvajanja algoritma. Z grafa 10, ki prikazuje zgoraj navedene izmerjene čase izvajanja algoritma z osnovo 2 (decimacija v časovnem prostoru), je razvidno, da je maksimalni čas izvajanja za velike N malo manj kot trikrat večji od srednje vrednosti časa izvajanja algoritma. Zaradi vrednosti, ki so prikazane na omenjenem grafu, smo se odločili, da se bomo pri meritvah časa osredotočili na srednjo vrednost izmerjenega časa. To pomeni, da so vsi prikazani časi na grafih v tem članku srednje vrednosti izmerjenih časov.
Rekurzivni algoritem z osnovo 2 z decimacijo v časovnem prostoru je najpočasnejši med algoritmi, ki smo jih implementirali po metodi Cooley-Tukey. Vendar je vseeno veliko hitrejši kot navadna DFT, kar jasno pokaže graf 1.



Slika 1: Srednja vrednost časa izvajanja DFT in algoritma z osnovo 2 z decimacijo v časovnem prostoru.

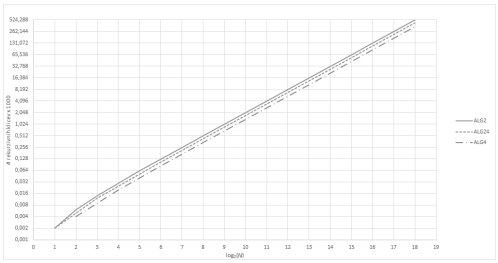
Zanimiva je ugotovitev z grafa 2, da so algoritmi (implementirani na rekurzivni način), ki uporabljajo decimacijo v frekvenčnem prostoru, hitrejši od tistih z decimacijo v časovnem prostoru, kljub enakemu številu operacij. Prav tako tudi, da je algoritem z osnovo 4 hitrejši od algoritma z deljeno osnovo, saj za izračun potrebuje večje število kompleksnih množenj.



Slika 2: Grafični prikaz srednje vrednosti časa izvajanja v odvisnosti od dolžine vhodnih podatkov algoritmov, ki uporabljajo metodo Cooley-Tukey.

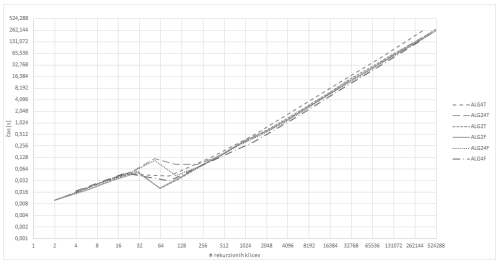
4.3 Rekurzivni klici

Zanimalo nas je, kako število rekurzivnih klicev vpliva na čas izvajanja algoritma, kar je prikazano na grafu 3. Vidimo, da je število klicev pri osnovi 2 največje, pri algoritmu z osnovo 4 pa najmanjše, kar je posledica načina delitve vhodnega problema na manjše podprobleme. Število rekurzivnih klicev je neodvisno od prostora decimacije (časovni ali frekvenčni).



Slika 3: Grafični prikaz števila rekurzivnih klicev v odvisnosti od dolžine vhodnih podatkov.

Spodnji graf 4 prikazuje srednjo vrednost časa izvajanja algoritmov v odvisnosti od števila rekurzivnih klicev. Vidimo, da čas narašča premosorazmerno s številom klicev. Sklepamo lahko, da poleg števila klicev na čas izvajanja vpliva še neka druga lastnost, saj imajo algoritmi z enako osnovo različne čase izvajanja (čas je očitno odvisen tudi od prostora decimacije).



Slika 4: Grafični prikaz srednje vrednosti časa izvajanja algoritma v odvisnosti od števila rekurzivnih klicev.

4.4 Napoved časa izvajanja algoritmov glede na število rekurzivnih klicev

Graf 4 iz prejšnjega razdelka prikazuje linearitmično - približno $a + b * nlog(n)$ - odvisnost časa izvajanja algoritma od števila rekurzivnih klicev. Zaradi te lepe lastnosti in ker je čas izvajanja algoritma težko (pošteno) izmeriti, smo se odločili, da ga bomo poskušali napovedati s pomočjo števila rekurzivnih klicev. S pomočjo sistema ALGator smo izmerili število rekurzivnih klicev ter na podlagi rezultatov razvili naslednje formule:

- **algoritma z osnovo 2**:

$$\#C_2(N) = 2\frac{1-N}{1-2} \tag{3}$$

- **algoritma z osnovo 4**:

$$\#C_4(N) = 4\frac{1-N}{1-4} \tag{4}$$

- **algoritma z deljeno osnovo**:

$$\#C_{24}(N) = \begin{cases} 2 & N = 2 \\ 5 & N = 4 \\ 2 \cdot \#C_{24}(\frac{N}{4}) + \#C_{24}(\frac{N}{2}) + 3 & N \geq 5 \end{cases} \tag{5}$$

kjer je N ustrezna dolžina vhodnih podatkov, ki jih sprejme algoritem. Iz tabele na sliki 5 vidimo, da se število izračunanih rekurzivnih klicev popolnoma ujema z izmerjenim številom. Z metodo najmanjših kvadratov smo potem iz podatkov z grafa 4 za vsak algoritem izračunali konstanti *a* in *b* po formuli:

$$b = \frac{r\sum_{i=1}^{r}(y_i \ln(x_i)) - \sum_{i=1}^{r} y_i \sum_{i=1}^{r} \ln(x_i)}{r\sum_{i=1}^{r} \ln(x_i)^2 - (\sum_{i=1}^{r} \ln(x_i))^2}, \tag{6}$$

$$a = \frac{\sum_{i=1}^{r} y_i - b\sum_{i=1}^{r} \ln(x_i)}{r}. \tag{7}$$

Pri tem pa uporabili prvih *r* izmerjenih točk in sicer *r* = 9 (oziroma 6 in 8) za osnovo 2 (oziroma 4 in deljeno osnovo). Dejanske izračunane vrednosti konstant za posamezen algoritem so prikazane v tabeli na sliki 6.

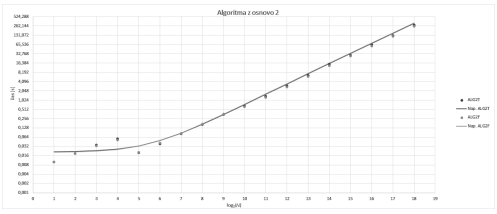
	ALG2		ALG4		ALG24	
	DIT	*DIF*	*DIT*	*DIF*	*DIT*	*DIF*
a	21,7128	20,9316	28,3583	32,2245	46,8604	38,4760
b	0,0462	0,0474	0,0491	0,0303	0,0388	0,0355

Slika 6: Izračunane vrednosti konstant *a* in *b*.

Nato smo z logaritmičnim ujemanjem po spodnji formuli izračunali napovedan čas izvajanja za vsak algoritem:

$$T(N) = a + b \cdot C_q(N) \cdot \ln(C_q(N)). \tag{8}$$

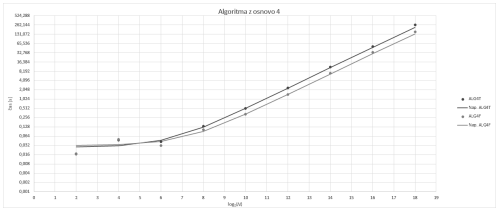
S spodnjih grafov 7, 8, 9 razberemo, da se napovedan čas dobro ujema z izmerjenim.



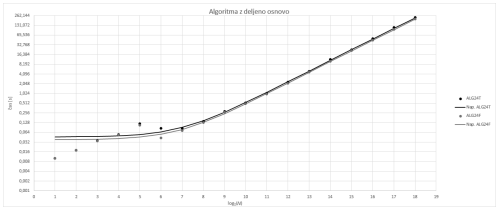
Slika 7: Izmerjen in napovedan (izračunan) čas izvajanja za algoritma z osnovo 2.

	ALG2T		ALG2F		ALG4T		ALG4F		ALG24T		ALG24F	
N	*Izmerjen #CALL*	*Izračunan #CALL*	*Izmerjen #CALL*	*Izračunan #CALL*	*Izmerjen #CALL*	*Izračunan #CALL*	*Izmerjen #CALL*	*Izračunan #CALL*	*Izmerjen #CALL*	*Izračunan #CALL*	*Izmerjen #CALL*	*Izračunan #CALL*
2	2	2	2	2	/	/	/	/	2	2	2	2
4	6	6	6	6	4	4	4	4	5	5	5	5
8	14	14	14	14	/	/	/	/	12	12	12	12
16	30	30	30	30	20	20	20	20	25	25	25	25
32	62	62	62	62	/	/	/	/	52	52	52	52
64	126	126	126	126	84	84	84	84	105	105	105	105
128	254	254	254	254	/	/	/	/	212	212	212	212
256	510	510	510	510	340	340	340	340	425	425	425	425
512	1022	1022	1022	1022	/	/	/	/	852	852	852	852
1024	2046	2046	2046	2046	1364	1364	1364	1364	1705	1705	1705	1705
2048	4094	4094	4094	4094	/	/	/	/	3412	3412	3412	3412
4096	8190	8190	8190	8190	5460	5460	5460	5460	6825	6825	6825	6825
8192	16382	16382	16382	16382	/	/	/	/	13652	13652	13652	13652
16384	32766	32766	32766	32766	21844	21844	21844	21844	27305	27305	27305	27305
32768	65534	65534	65534	65534	/	/	/	/	54612	54612	54612	54612
65536	131070	131070	131070	131070	87380	87380	87380	87380	109225	109225	109225	109225
131072	262142	262142	262142	262142	/	/	/	/	218452	218452	218452	218452
262144	524286	524286	524286	524286	349524	349524	349524	349524	436905	436905	436905	436905

Slika 5: Število izmerjenih in izračunanih rekurzivnih klicev za posamezen algoritem.

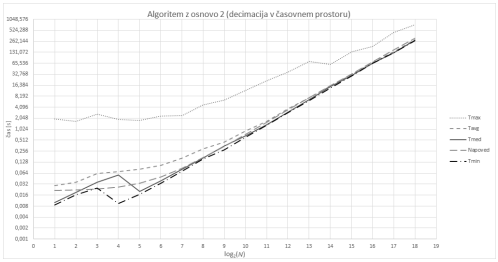


Slika 8: Izmerjen in napovedan (izračunan) čas izvajanja za algoritma z osnovo 4.



Slika 9: Izmerjen in napovedan (izračunan) čas izvajanja za algoritma z deljeno osnovo.

Za primerjavo z ostalimi izmerjenimi časi algoritma z osnovo 2 DIT smo vključili še graf 10. Razlika pri ostalih algoritmih je prav tako zelo podobna.



Slika 10: Grafični prikaz izmerjenih časov in napovedanega (izračunanega) za algoritem z osnovo 2 (DIT).

5 Zaključek

S pomočjo sistema ALGator smo na vhodnih podatkih dolžine od 2^2 pa do 2^{18} testirali 4 različne algoritme za izračun Fourierjeve transformacije in njenega inverza. Po pričakovanjih se je najslabše odrezal algoritem, ki smo ga implementirali striktno po definiciji DFT. Med algoritmi, ki smo jih implementirali z rekurzijo, se je najbolje odrezal algoritem z osnovo 4 z decimacijo v frekvenčnem prostoru, kljub temu, da algoritem z deljeno osnovo potrebuje manj množenj.
Rezultati pri izračunu časa izvajanja v odvisnosti od števila rekurzivnih klicev algoritmov kažejo zelo dobro ujemanje napovedanih (izračunanih) in izmerjenih vrednosti. Imajo enak trend kot izmerjen povprečni čas izvajanja. Čas izvajanja smo merili v 18 točkah, krivuljo za napoved pa smo izračunali iz manj kot prve polovice izmerjenih točk, kar kaže na dobro metodo napovedi.

Literatura

[1] Henri J. Nussbaumer, “Fast Fourier Transform and Convolution Algorithms: Second Corrected and Updated Edition”, 2. izd., *Springer Series in Information Science*, vol. 2, str. 81–94, Berlin [etc.]: Springer, 1982.

[2] E. Chu, A. George “Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms”, *Computational Mathematics Series*, Boca Raton: CRC Press LLC, 2000.

[3] T. Dobravec, “ALGator - izvajanje in analiza algoritmov”. [Online].
Dosegljivo:
https://github.com/ALGatorDevel/Algator. [Dostopano 18. 2. 2016].

[4] P. Duhamel, Henk D. L. Hollmann. “Split radix FFT algorithm”. Dosegljivo: goo.gl/tYmWcG. [februar 2016].

[5] Ž. Zorman “Primerjava algoritmov za izračun Fourierjeve transformacije s pomočjo sistema ALGator : diplomsko delo”. V Ljubljani: [Ž. Zorman], 2017. [COBISS-ID 1537378755].

[6] Ž. Zorman “Izvorna koda implementiranih algoritmov”. Dosegljivo: https://github.com/Flinch010/bachelor-s-degree. [februar 2017].