SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2128

Adaptivna valićna transformacija ostvarena na CUDA arhitekturi

Matija Osrečki

Zagreb, lipanj 2011.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`*.*

Zvonku, Janji, Miletu, Ani i ostalima na putu.

Sadržaj

1. Uvod **1**

2. Valićna transformacija **2**

2.1. Fourierova transformacija na vremenskom otvoru 2

2.2. Kontinuirana valićna transformacija 3

2.3. Diskretna valićna transformacija 3

2.4. Višerezolucijska analiza i funkcija skale 4

2.5. Filtarski slogovi 6

2.6. Koraci podizanja i ostvareni sustav 7

3. Odabir adaptive veličine vremenskog prozora **9**

3.1. Linearna regresija minimizacijom *L2* norme 9

3.2. *L2* linearna regresija nad vremenskim prozor 10

3.3. Efikasno računanje varijabilnog parametra 10

3.4. Odabir parametra intervalom dozvole 12

3.4.1. Opis algoritma 13

3.5. Uporaba metoda odšumljivanja 14

3.5.1. Općenito o RICI metodi 15

3.5.2. Primjena RICI metode za odabir adaptivnog parametra 17

3.6. Usporedba dviju metoda 18

4. Paralelizacija **20**

4.1. Općenito o CUDA-i i parelizaciji 20

4.2. Logička i memorijska organizacija CUDA arhitekture 21

4.3. Paralelizacija RICI metode 23

4.4. Rezultati 26

5. Zaključak **29**

Literatura **30**

1. Uvod

U suvremeno doba okruženi smo mnoštvom informacija. Mi želimo da te informacije budu što bolje kvalitete, i kad je žurba želimo ih brzo. Zahvaljujući razvoju modernih tehnologija situacija je sve bolja. Obrada multimedijalnih sadržaja poput videa, slike ili zvuka su samo jedno od mnogih područja na kojem se taj razvoj ostvaruje.

Kad govorimo o obradi takvih tipova informacija, zapravo se radi o obradi digitalnih signala. Cilj ovog rada jest prikazati suvremene metode kojima se signali mogu analizirati i obrađivati, ali također i metode kojima se postiže optimizacija vremenskih performansi takve obrade signala. Sama analiza signala je implementirana uporabom adaptivne valićne transformacije, metode slične Fourierovoj transformaciji. Valićna transformacija sadrži fiksni dio i adaptivni dio koji može poboljšati ta svojstva. U nastavku rada su opisane metode koje vrši odabir adaptivnih parametara u nadi postizanja boljih svojstava analize.

S druge strane, optimizacija vremenskih performansi se postiže paralelizacijom na grafičkom procesoru. U tu svrhu je odabrana CUDA tehnologija razvijena od strane Nvidie. CUDA je dobra za jednostavne proračune nad gomilom podataka, kao što je i za očekivati s obzirom da se radi o grafičkom procesoru. Pokazat će se kako je jedna od adaptivnih metoda upravo pogodna za takvu paralelizaciju i pretpostavka je da će CUDA zaista značajno ubrzati taj dio.

U nastavku rada je najprije objašnjena sama valićna transformacija i odabir adaptivnog dijela, dok pred kraj slijedi opis CUDA arhitekture i rezultati implementacije.

2. Valićna transformacija

Poput Fourierove transformacije, valićna transformacija je matematički alat kojim neki signal možemo promatrati u nekoj drugoj domeni u kojoj do izražaja dolaze neka svojstva koja nas zanimaju. Prisjetimo se da u Fourierovoj transformaciji početni signal dobijemo zbrajanjem sinusoida različitih frekvencija, pomnoženih s pripadajućim koeficijentom. Ti koeficijenti se zovu Fourierovi koeficijenti i oni čine frekvencijsku domenu.

Fourierova transformacija ima jedno ograničenje – u frekvencijskoj domeni ne postoji vremenska lokalizacija signala. To znači da ako mijenjamo signal na jednom dijelu, mijenjat će se čitava frekvencijska domena, što je neprikladno za signale koji se mijenjaju (nestacionarne signale). Kako bi to izbjegli, inženjeri iz različitih struka su neovisno došli do svojih rješenja za koje se kasnije ispostavilo da su varijante valićne transformacije. Jedino je preostajalo da matematičari to točno formuliraju.

U nastavku je ukratko objašnjena matematička pozadina valićne transformacije i sustav koji ju implementira. Detaljnije u [7].

2.1. Fourierova transformacija na vremenskom otvoru

Kako bi se riješio problem vremenske lokalizacije, najprije je osmišljena tzv. **Fourierova transformacija na vremenskom otvoru** ili STFT (eng. *short-time Fourier transform*). Ideja STFT-a je uzeti dio po dio signala i nad svakim dijelom provesti Fourierovu transformaciju. Za signal $x(t)$ i funkciju prozora $g(t)$, STFT je definirana na sljedeći način:

$$STFT(f, s) = \int_{-\infty}^{\infty} x(t)g(t - s)e^{-j2\pi ft}dt. \tag{2.1}$$

Treba primijetiti da za razliku od CTFT, spektar STFT je funkcija dvije varijable – frekvencije f i pomak s. Za neki pomak s signal se množi sa pomaknutom funkcijom prozora $g(t-s)$ i računa se CTFT za taj dio signala. Možemo zamisliti kako duž signala

mičemo prozor definiran funkcijom $g(t)$. $g(t)$ pritom može biti bilo koja funkcija koja filtrira dio signala. Pravokutni otvor $box(t)$ je najjednostavniji primjer takve funkcije:

$$box(t) = \begin{cases} 1 & \text{za } t \leq |\frac{1}{2}| \\ 0 & \text{inače.} \end{cases} \tag{2.2}$$

2.2. Kontinuirana valićna transformacija

Glavni problem STFT-a je što ima fiksnu rezoluciju, odnosno širinu otvora, odnosno ne može dobro istovremeno razriješiti i frekvencije i vremenske događaja. Odabir većeg otvora rezultira boljom frekvencijskom rezolucijom, a lošijom vremenskom rezolucijom, dok suprotno vrijedi za odabir manjeg otvora. Upravo zato je razvijena valićna transformacija, koja omogućuje istovremenu analizu signala u više rezolucija. Osim toga, brza valićna transformacija, čija je vremena složenost $O(n)$ je računarski efikasnija od brze Fourierove transformacije (FFT), vremenske složenosti $O(n \log n)$.

Kontinuirana valićna transformacija ili CWT (eng. *continuous wavelet transform*) je definirana na sljedeći način:

$$C(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t) \psi \left(\frac{t-b}{a} \right) dt \tag{2.3}$$

Analogno STFT-u, a predstavlja skalu (obrnuto proporcionalno frekvenciji), dok b predstavlja vremenski pomak. $C(a,b)$ je skalarni produkt signala, odnosno korelacija $x(t)$ i skalirane i posmaknute valićne funkcije $\psi\left(\frac{t-b}{a}\right)$, koja prikazuje sličnost dobivenog valića i signala. Valićna funkcija $\psi(t)$ je obično mali valić, integralne sume 0 i konačne energije. Najjednostavniji primjer je Haarov valić:

$$\psi(t) = \begin{cases} 1 & 0 \leq t < \frac{1}{2} \\ -1 & \frac{1}{2} \leq t < 1 \\ 0 & \text{inače.} \end{cases} \tag{2.4}$$

2.3. Diskretna valićna transformacija

CWT ima jedan problem – redundancija. Pošto se koeficijenti računaju za sve moguće pomake valića $\psi(i)$, očito dolazi do preklapanja posmaknutih valića. Kako bi se uklonila redundancija, potrebno je zadati skup signala koji predstavlja tzv. **ortogonalnu bazu**. Taj koncept nije samo isključiv za analizu signala. U 3-dimenzionalnom vektorskom prostoru svaki vektor se $\vec{a}$ može prikazati kao linearna kombinacija vektora $\vec{i}$, $\vec{j}$ i

$\vec{k}$, koji pritom čine ortogonalnu bazu[1]. Uostalom sami signali se mogu promatrati kao beskonačno dimenzionalni vektori.

U Fourierovoj transformaciji ortogonalnu bazu čine sinusoide proizvoljnih frekvencija i faza. U STFT, ortogonalna baza se postiže odabirom jednako udaljenih frekvencija i vremenskih pomaka – $STFT(n/T, mT)$. T upravo mora biti širina prozora $g(t)$, jer inače ovo ne bi predstavljalo ortogonalnu bazu.

U nekim slučajevima, ortogonalna baza za CWT se postiže odabirom skala koje su potencija broja 2, a vremenski pomaci višekratnici vrijednosti skale:

$$C(1/2^j, k/2^j) = 2^{j/2} \int_{-\infty}^{\infty} x(t)\psi(2^j t - k)dt. \tag{2.5}$$

Prema tome, skalirani i pomaknuti valić se definira na sljedeći način:

$$\psi_{j,k} = 2^{j/2}\psi(2^j t - k). \tag{2.6}$$

Postoji velika klasa valića $\psi(t)$ za koje $\psi_{j,k}$ predstavlja ortogonalnu bazu. U tom slučaju se ta transformacija zove **diskretna valićna transformacija** ili DWT (eng. *discrete wavelet transform*).

$$DWT : c_{j,k} = \int_{-\infty}^{\infty} x(t)\psi_{j,k}(t)dt \tag{2.7}$$

Rekonstrukcija signala se dobije pomoću **inverzne diskretne valićne transformacije** ili IDWT (eng. *inverse discrete wavelet transform*).

$$IDWT : x(t) = \sum_j \sum_k c_{j,k}\psi j_k \tag{2.8}$$

Unatoč tome što DWT nema redundancije, CWT je u nekim slučajevima bolja jer je manje osjetljivija na šum. Također, DWT nije vremenski stalna, dok CWT je, što znači da DWT vremenski posmaknutog signala neće rezultirati vremenski posmaknutim spektrom originalnog signala.

2.4. Višerezolucijska analiza i funkcija skale

Kako bi se omogućila efikasna primjena DWT, sljedeći problem koji treba riješiti je beskonačan broj različitih frekvencija. Zbog toga valićnu transformaciju treba sagledati na drugi način.

[1]Geometrijska interpretacija ortogonalnosti jest okomitost

Neka V_j bude skup svih signala $x(t)$ koji se može dobiti pomoću valova $\psi_{i,k}$ takvih da vrijedi $i < j$, za svaki k.

$$x(t) = \sum_{i=-\infty}^{j-1} \sum_{k} c_{i,k} \psi_{i,k}(t) \tag{2.9}$$

Prostori V_j su ugniježđeni, pritom za $j = \infty$, V_j predstavlja prostor $L2^2$, a za $j = -\infty$ samo signal nule $x(t) = 0$. V_{j+1} se jednostavno izračuna iz V_j:

$$V_{j+1} = V_j + \sum_{k} c_{j,k} \psi_{j,k}(t) = V_j + W_j. \tag{2.10}$$

Gdje je W_j skup svih signala koji se dobije pomoću valića $\psi_{j,k}$, za svaki k. Neki prostor V_i se dalje može razložiti:

$$\begin{aligned} V_i &= V_{i-1} + W_{i-1} \\ &= V_{i-2} + W_{i-2} + W_{i-1} \\ &= V_{i-3} + W_{i-3} + W_{i-2} + W_{i-1} \end{aligned} \tag{2.11}$$

Dobiveni signal se tako može razlagat od većih prema manjim rezolucijama (eng. *"From finer scale to coarser scale"*), što se obično označava na sljedeći način:

$$\begin{aligned} x(t) &= A_1 + D_1 \\ &= A_2 + D_2 + D_1 \\ &= A_3 + D_3 + D_2 + D_1 \end{aligned} \tag{2.12}$$

Ovdje D_i označava detalj u i-toj razini, dok A_i označava aproksimaciju u i-toj razini. U svakoj razini se računa novi detalj, sve niže frekvencije. U jednom trenutku treba stati i izračunati aproksimaciju signala, što se postiže uporabom funkcije skale $\phi(t)$. Svaki signal $x(t)$ iz skupa V_j dobijemo uporabom funkcije skale.

$$x(t) = \sum_{k} \phi_{j,k}(t) \tag{2.13}$$

Funkcija skale $\phi_{j,k}(t)$ je definirana slično valićnoj funkciji $\psi_{j,k}(t)$.

$$\phi_{j,k} = 2^{j/2} \phi(2^j t - k). \tag{2.14}$$

Funkcija skale za Haarov valić je definirana na sljedeći način:

$$\phi(t) = \begin{cases} 1 & 0 \leq t < 1 \\ 0 & \text{inače.} \end{cases} \tag{2.15}$$

[2] *L2* je Hilbertov prostor, tj. prostor signala konačne energije

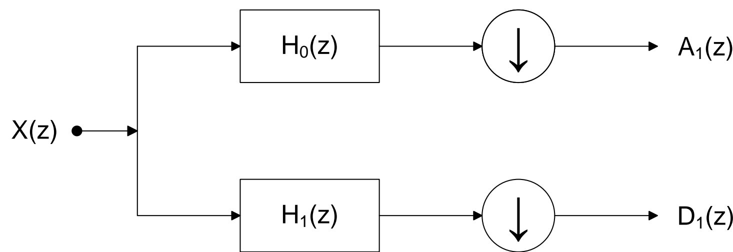
Prema tome, svaki signal $x(t)$ pomoću DWT-a se može prikazati pomoću funkcije skale i valićne funkcije:

$$x(t) = \overbrace{\sum_{k=-\infty}^{\infty} a_{0,k}\phi_{0,k}(t)}^{aproksimacija} + \overbrace{\sum_{j=0}^{\infty}\sum_{k=-\infty}^{\infty} d_{j,k}\psi_{j,k}(t)}^{detalj}. \tag{2.16}$$

2.5. Filtarski slogovi

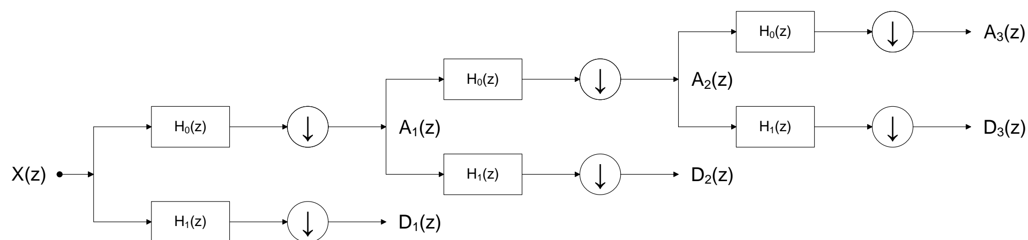
U prethodnom poglavlju je prikazano kako se neki signal $x(t)$ može rastaviti od redom od većih prema manjim rezolucijama. Pri tome se dobiveni signal viših rezolucija zove detalj, a nižih rezolucija aproksimacija.

U nastavku se razmatra otipkani signal $X(nT)$, odnosno $X(z)$. Otipkavanjem signala je određen najveći prostor signala V_P u kojem se signal nalazi. Prema tome u jednoj razini analize signala, dobije se aproksimacija $A_1(z)$ koja je u prostoru V_{P-1} i detalj $D_1(z)$ prostora W_{P-1}. Na slici 2.1 je prikazan sustav koji vrši tu analizu.



Slika 2.1: Jedna razina analize signala filtarskim slogom

Početni signal $x(z)$ paralelno prolazi kroz dva filtra $H_0(z)$ i $H_1(z)$ i vrši se decimacija, odnosno uzima se svaki drugi član signala. Filtar $H_0(z)$ je niskopropusni filtar dok je $H_1(z)$ visokopropusni filtar. Ako je potrebno više koraka analize, u tom slučaju se aproksimacija $A_1(z)$ šalje dalje kroz isti ovaj filtar, kao što je prikazano na slici 2.2.

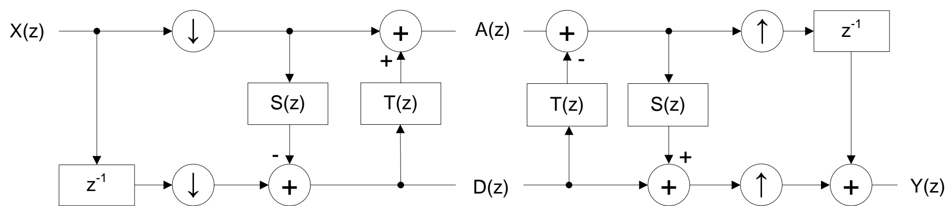


Slika 2.2: Više razina analize signala filtarskim slogom

Filtri $H_0(z)$ i $H_1(z)$ su specifični za odabir valićne funkcije $\psi(t)$. Rekonstrukcija se vrši upravo obrnuto od analize, uporabom filtera $F_0(z)$ i $F_1(z)$ također specifičnih za neku valićnu funkciju, i interpolacijom umjesto decimacije.

2.6. Koraci podizanja i ostvareni sustav

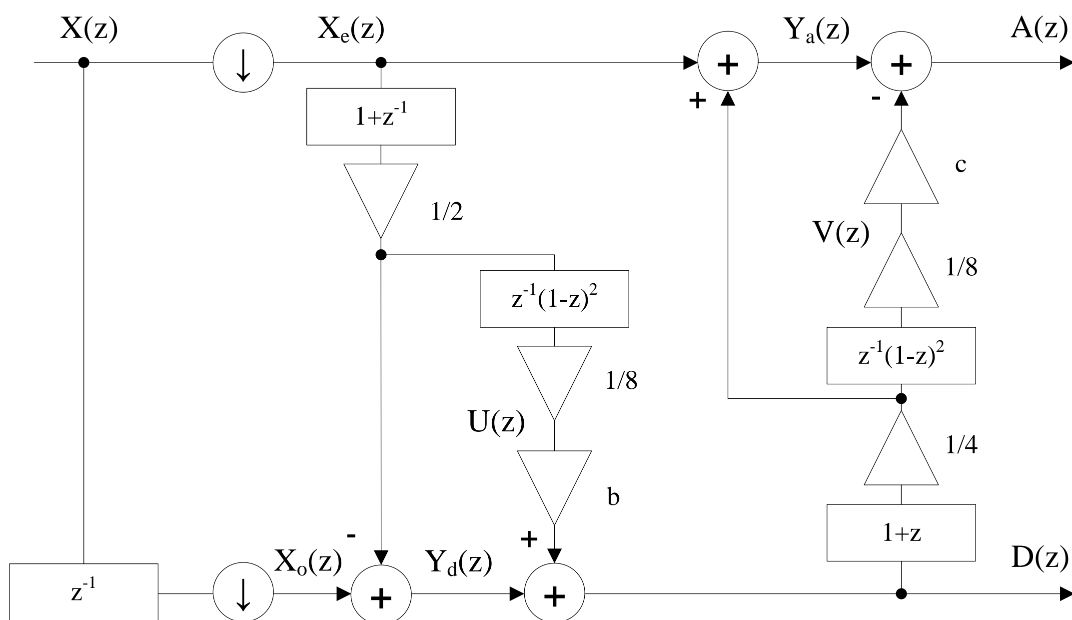
Pokazano je da ako filtri $H_0(z)$ i $H_1(z)$ zadovoljavaju određene uvjete, da se koraci podizanja mogu implementirati i uporabom **koraka podizanja** (eng. *lifting scheme*), kao što je prikazano na slici 2.3.



Slika 2.3: Analiza uporabom koraka podizanja

Osim analize, na slici 2.3 je prikazana i rekonstrukcija, koja je upravo obrnuta samoj analizi. $T(z)$ i $S(z)$ su filtri koji se mogu dobit iz $H_0(z)$ i $H_1(z)$.

U ovom radu je implementirana valićna transformacija uporabom koraka podizanja, a sam sustav je prikazan na slici 2.4. U nastavku je objašnjena metoda optimalnog odabira parametara b ili c, za potrebe kompresije, uklanjanja šuma ili drugih oblika obrade signala.



Slika 2.4: Konačni sustav za analizu

3. Odabir adaptive veličine vremenskog prozora

3.1. Linearna regresija minimizacijom *L2* norme

Linearna regresija je metoda određivanja odnosa između varijable Y i jedne ili više varijabli X, gdje se Y modelira linearnom funkcijom na temelju podataka i nepoznatih parametara te linearne funkcije.

Za zadani skup podataka $\{y_i, x_{i1}, \ldots, x_{ip}\}_{i=1}^n$, gdje je n broj uzoraka, linearna regresija pretpostavlja linearan odnos između varijable y_i i vrijednosti p-dimenzionalnog vektora regresije X_i. Osim toga, u model je uključena slučajna varijabla ϵ_i, koja predstavlja grešku.

$$y_i = \beta_i x_{i1} + \cdots + \beta_p x_{ip} + \epsilon_i = x_i^\intercal \beta + \epsilon_i, \quad i = 1, \ldots, n, \tag{3.1}$$

$$y = X\beta + \epsilon, \quad i = 1, \ldots, n \tag{3.2}$$

Izraz 3.1 predstavlja skalarni produkt vektora x_i i β. Pritom se y_i se zove regresant, x_i regresor, dok β je vektor parametara, a ϵ_i vektor greške. Pritom su y i ϵ n-dimenzionalni vektori, a X je $n \times p$ matrica.

Najjednostavnija metoda određivanja nepoznatih parametara β_p linearne regresije jest minimizacijom $L2$ norme pogreške ϵ_i [1].

$$S(b) = \sum_{i=1}^{n} \epsilon_i^2 = (y - Xb)^\intercal (y - Xb) \tag{3.3}$$

$$\beta = \arg\min_{b \in \mathbb{R}^n} S(b) = (X^\intercal X)^{-1} X^\intercal y \tag{3.4}$$

Iz izraza 3.4 se vidi kako se β može jednostavno i egzaktno izračunati uporabom formule u zatvorenoj formi. U nastavku rada je prikazano kako je linearna regresija minimizacijom $L2$ norme uz neke tehnike iskorištena za obrađivanje signala u okviru valićne transformacije i filtarskih slogova [2].

3.2. *L2* linearna regresija nad vremenskim prozor

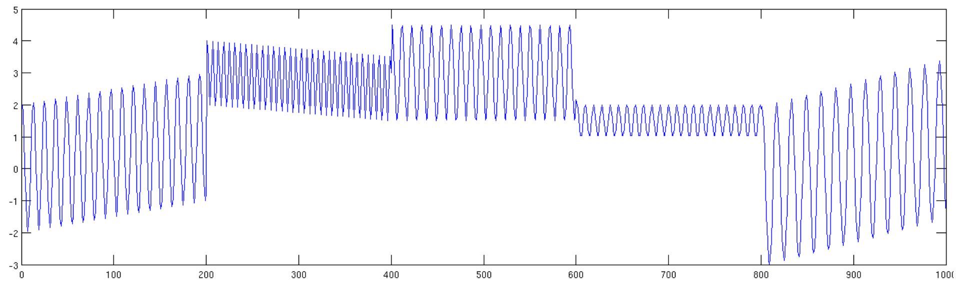
Adaptivna valićna transformacija uporabom koraka podizanja osim svog fiksnog dijela, ima i adaptivni dio. Filtri $S(z)$ i $T(z)$ sadrže varijabilne parametre b, odnosno c. Parametar b je posebno zanimljiv pošto se njegovom manipulacijom može postići da što manje informacija završi u detalju, odnosno da je što bliži nuli, što je korisno za potrebe kompresije. Također se može postići da samo šum ode u detalj i na taj način se može eliminirati. Određivanje traženog parametra b se vrši upravo linearnom regresijom na temelju signala $Y_d(z)$ (regresant) i $U(z)$ (regresor), i to u ovom slučaju minimizacijom $L2$ norme.

Pošto signali mogu biti jako promjenjive prirode, bilo bi neprikladno odabrati jedinstveni parametar na temelju čitavog područja signala. Ako se prisjetimo, ovo je analogno problemu Fourierove transformacije - nema vremenske lokalizacije, odnosno spektar je jedan jedini za cijeli signal. Tako je i došlo do STFT-a, gdje se Fourierova transformacija provodila na malim dijelovima signala, ali je spektar poprimio dodatnu dimenziju. Ista ideja se može primijeniti i ovdje – umjesto da se linearna regresija provede na cijelom signalu, za svaki element signala se provodi na malom dijelu oko tog elementa. Nažalost, ova ideja pati od istog problema kao i STFT – odabir veličine vremenskog prozora je značajno za rezultate. Ukoliko je prozor manji – utjecaj šuma je veći, a ako je prozor veći – prijelazna područja različitih frekvencije će biti veća. U tu svrhu se bira različita širina prozora za svaki element signala, o čemu će biti govora u nastavku.

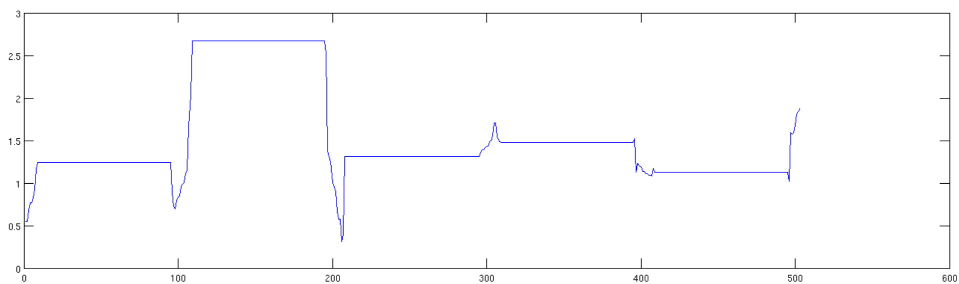
Slika 3.1 prikazuje ulazni signal – kombinacija sinusoida bez utjecaja šuma. Parametri b za veličinu prozora 11 su prikazani na slici 3.2. Može se vidjeti kako svakoj sinusoidi pripada jedno pravokutno područje vrijednosti proporcionalno frekvenciji. Istom signalu je dodan šum i rezultat je prikazan na slici 3.3. Na slici 3.4 je prikazano kako odabir veličine prozora utječe na parametar b. Glavni cilj ovog rada je upravo određivanja parametra b za svaki element tako da se što više potisne utjecaj šuma, odnosno da odabir parametra što više nalikuje onom na slici 3.2. Ako se u tome uspije, šum će završiti u detalju, a vrijednosti parametra b se može komprimirati. Detalj se može postaviti na nulu kako bi se uklonio šum ili komprimirala slika.

3.3. Efikasno računanje varijabilnog parametra

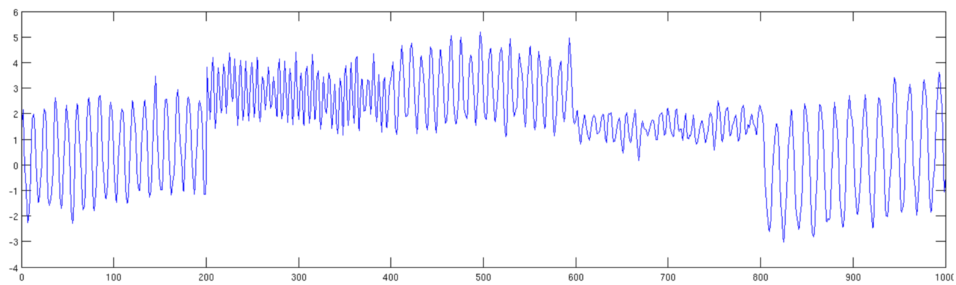
U prethodnom poglavlju je prikazano kako se odabir adaptivnog parametra za neki element signala vrši linearnom regresijom nad prozorom određene veličine oko tog



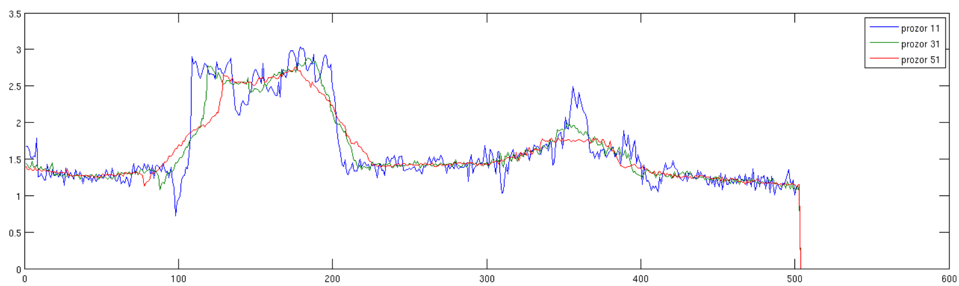
Slika 3.1: Signal $x(z)$ duljine 1000 bez šuma



Slika 3.2: Parametar b dobiven linearnom regresijom nad prozorom veličine 11 za signal $x(z)$



Slika 3.3: Signal $y(z)$ dobiven dodavanjem šuma na signal $x(z)$



Slika 3.4: Parametar b dobiven linearnom regresijom za signal $y(z)$

elementa. Metode koje su u nastavku rada objašnjene koriste vrijednosti parametara različitih elementa (pozicija prozora) signala i različitih širina prozora. Broj parametara je u kvadratnoj ovisnosti o duljini signala i zato se vrijednosti svih parametara ne računaju unaprijed, nego samo po potrebi. Računanje vrijednosti parametra za različite pozicije i veličine prozora je moguće u vremenskoj složenosti $O(1)$, ako se prije toga obave neki proračuni, koji su vremenske i memorijske složenosti $O(n)$.

Neka su zadani kauzalni signali $y(n)$ (regresant) i $x(n)$ (regresor) duljine n, nad kojima se vrši linearna regresija minimizacijom $L2$ za različite pozicije i veličine prozora. Neka je za primjer prozor za koji radimo proračun na poziciji p i veličine $2k+1$. Parametar b se tada može računski dobiti formulom koja slijedi.

$$b_{p,2k+1} = \frac{\sum_{i=p-k}^{p+k} x(i)y(i)}{\sum_{i=p-k}^{p+k} x^2(i)} \tag{3.5}$$

Iz formule 3.5, koje je malo drugačiji oblik formule 3.4, vidi se da je računanje suma umnožaka $x^2(i)$ i $x(i)y(i)$ ono što predstavlja računarsku složenost. U tu svrhu se sume tih umnožaka trebaju preračunati prije složenijih algoritama, kako izračun parametara ne bi povećavao vremensku složenost.

$$xx(n) = \sum_{i=0}^{n} x^2(i), \quad xy(n) = \sum_{i=0}^{n} x(i)y(i) \tag{3.6}$$

Iz čega slijedi:

$$b_{p,2k+1} = \frac{xy(p+k) - xy(p-k-1)}{xx(p+k) - xx(p-k-1)} \tag{3.7}$$

Pritom su ovdje iz konvencije veličine prozora ograničene samo na neparne vrijednosti, ali ovako se mogu računati za parne veličine prozora.

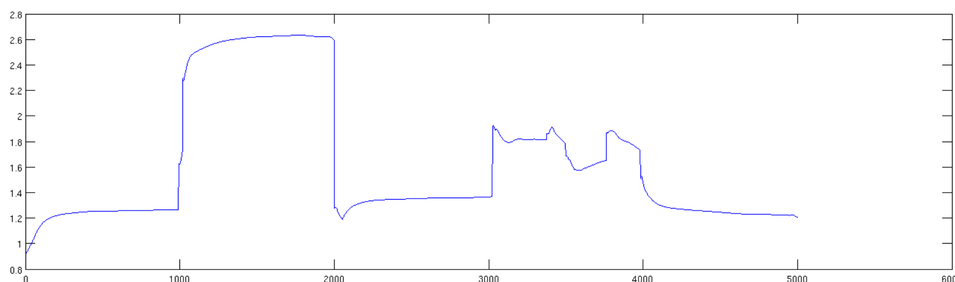
3.4. Odabir parametra intervalom dozvole

Kako bi se shvatila ideja iza prve metode za dobivanje treba obratiti pozornost na sliku 3.2. Sa nje se vidi da se idealno rješenje sastoji od područja konstantne vrijednosti parametra b. U usporedbi s idealnim rješenjem, odabir manjeg prozora rezultira bržim prijelazom iz jednog područja u drugo, ali je utjecaj šuma prevelik. S druge strane veći prozori rezultiraju glađim vrijednostima prema sredini pojedinog područja, ali je prijelaz veći. Najveći problem ove metode je određivanje kada se dogodi prijelaz.

3.4.1. Opis algoritma

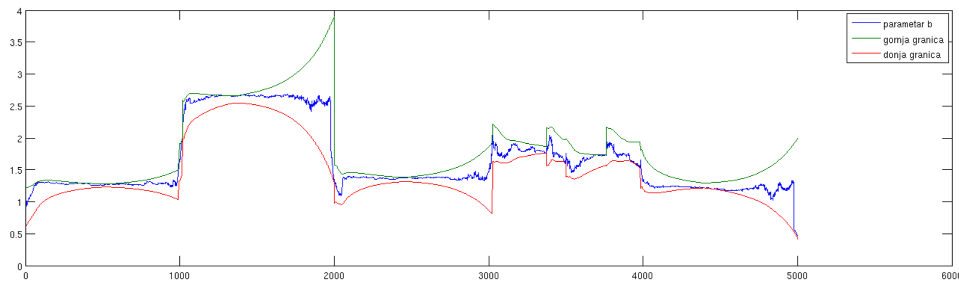
Algoritam kreće od početka signala i prema tome je na početku nekog od područja. Ideja je krenuti od manjih prozora i krećući se po pozicijama signala prema desno, pomalo povećavati prozor dok se ne otkrije da se više ne može povećavati. U tom trenutku bi trebali biti na sredini područja i algoritam počinje smanjivati veličinu prozora dok se ne dođe do kraja područja. U svakom koraku se na određenu poziciju kao rezultat piše vrijednost parametra na toj poziciji za trenutnu vrijednost prozora.

Ali kako odrediti kada je došlo do sredine i kraja područja? U tu svrhu se određuje interval (**interval dozvole**) unutar kojeg vrijednost parametra za trenutnu poziciju mora biti kako se ne bi prekinula trenutna faza algoritma. To znači da ako trenutno povećavamo veličinu prozora i vrijednost izađe iz tog intervala, prelazi se u drugu fazu gdje se smanjuje veličina prozora. Ako je algoritam u drugoj fazi i vrijednost izađe, znači da je to kraj područja i algoritam kreće isponova za sljedeće područje. Interval je područje oko srednje vrijednosti dodanih vrijednosti parametara, čija veličina pada prema sredini područja i opet raste prema kraju područja. Na početku se srednja vrijednost aproksimira kao medijan ili srednja vrijednost nekoliko početnih vrijednosti manjih veličina prozora.

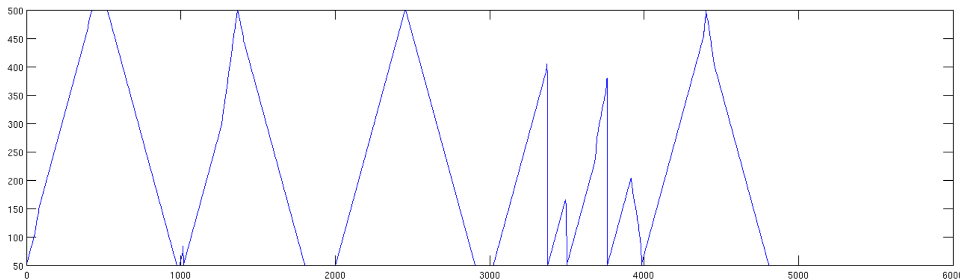


Slika 3.5: Rezultat prve metode odabira parametra b

Na slici 3.5 je prikazan rezultat gore opisanog algoritma. Kao rezultat se za pojedini element uzima srednja vrijednost umjesto vrijednosti samog parametra, što uzrokuje izglađenost. Ta ideja je preuzeta od algoritma za uklanjanje šuma koji je u nastavku rada opisan. Na slici 3.6 je prikazano kako se interval koji određuje područje mijenja i sama vrijednost parametra za trenutnu veličinu prozora. Ta širina prozora je prikazana na slici 3.7 i sa nje se vidi kako veličina prozora raste prema sredini područja, što je i bila ideja. Jedino u 4. području algoritam zapinje, što je rezultat veće količine šuma.



Slika 3.6: Vrijednost parametra b i intervala dozvole



Slika 3.7: Veličina prozora za pojedini element

Algoritam dobro radi na primjeru, ali je previše ovisan o parametrima samog intervala, odnosno nedostaje mu robusnosti. Dobra stvar je linearna vremenska složenost. U nastavku je opisana vremenski složenija, ali robusnija metoda.

3.5. Uporaba metoda odšumljivanja

Drugi način aproksimacije parametra b je uporaba RICI (eng. *relative intersection of confidence intervals*) metode uklanjanja šuma. Ideja je izračunati parametre za svaki element, za neku manju veličinu prozora. Dobiveni parametri će biti zašumljeni, ali se neće osjetiti prijelazi između područja. Pritom treba ukloniti šum RICI metodom, koja je primjerena upravo za ovakve pravokutne signale. U nastavku slijedi opis RICI metode, način implementacije za odabir parametra b s obzirom na različite veličine prozora, i rezultati implementacije. Više o RICI metodi se može naći u [4].

3.5.1. Općenito o RICI metodi

RICI metoda je algoritam primjeren za uklanjanje šuma nad pravokutnim signalima. Zadan je signal $y(n)$ duljine N koji sadrži bijeli šum $\eta(n) \sim \mathcal{N}(0, \sigma_\eta^2)$

$$y(n) = x(n) + \eta(n), \tag{3.8}$$

gdje je $x(n)$ signal bez šuma.

Algoritam pokušava odrediti aproksimaciju $\tilde{x}(n)$ usrednjivanjem k elemenata signala $y(n)$:

$$\tilde{x}_k(n) = \frac{\sum_{j=1}^{k} y(n+j-1)}{k}. \tag{3.9}$$

Za svaki k, određuju se intervali pouzdanosti $D_k(n)$:

$$D_k(n) = [L_k(n), U_k(n)], \quad k \geq 1, \tag{3.10}$$

gdje su gornja i donja granica definirane kao

$$L_k(n) = \tilde{x}_k(n) - \Gamma\sigma_k(n) \tag{3.11}$$

$$U_k(n) = \tilde{x}_k(n) + \Gamma\sigma_k(n). \tag{3.12}$$

Γ je početna veličina prozora, $\sigma_k(n)$ standardna devijacija k uzorka $\tilde{x}_k(n)$. ICI algoritam prati najveću donju granicu i najmanju gornju granicu (ICI interval)

$$\overline{L}_k(n) = \max_{i=1,\ldots,k} L_i(n) \tag{3.13}$$

$$\underline{U}_k(n) = \min_{i=1,\ldots,k} U_i(n). \tag{3.14}$$

Bira se k^+, odnosno najveći k takav da vrijedi

$$\overline{L}_k(n) \leq \underline{U}_k(n), \tag{3.15}$$

i $x_{k^+}(n)$ se bira za procjenu $x(n)$.

Ovo je tek uvjet ICI metode (eng. *intersection of confidence intervals*), koji nije dovoljno dobar jer premali odabir vrijednosti parametra Γ rezultira prevelikim odabirom k^+ i dobiveni signal će biti previše izglađen. Ako je pak Γ premalen dobiveni signal će biti premalo izglađen.

Kao poboljšanje razvijena je RICI metoda, za koju nije bitan odabir parametra Γ. Predložena metoda se temelji na omjeru ICI intervala i trenutnog intervala pouzdanosti

$$R_k(n) = \frac{\underline{U}_k(n) - \overline{L}_k(n)}{U_k(n) - L_k(n)} = \frac{\underline{U}_k(n) - \overline{L}_k(n)}{2\Gamma\sigma_k(n)}. \tag{3.16}$$

Sada se bira najveći k koji zadovoljava i ICI uvjet i uvjet

$$R_k(n) \geq R_c, \tag{3.17}$$

gdje je R_c vrijednost praga, koja se određuje empirijski. RICI metoda predstavlja poboljšanje pred ICI metodom jer puno bolje prepoznaje rubove, i nije toliko ovisna o parametru Γ.

Algorithm 1 RICI metoda

$\Gamma \leftarrow 4.4$
$\sigma \leftarrow 0.2$
$R_c \leftarrow 0.85$
$result \leftarrow []$
for $i = 1 \rightarrow N$ **do**
  $avg \leftarrow x[i]$
  $sum \leftarrow x[i]$
  $k \leftarrow 1$
  $[lower, upper] \leftarrow [+\infty, -\infty]$
  repeat
    $k \leftarrow k + 1$
    $sum \leftarrow sum + x[i + k - 1]$
    $avg \leftarrow \frac{sum}{k}$
    $len \leftarrow \Gamma \frac{\sigma}{\sqrt{k}}$
    $[lower, upper] \leftarrow [avg - len, avg + len]$
    $R_k \leftarrow \frac{upper - lower}{2 * len}$
  until $(lower > upper) \vee (R_k < R_c)$
  $result.dodaj(\frac{sum - x[i+k-1]}{k-1}, k - 1)$
end for
return $result$

Prikazan je pseudokod 1, koji jasno ocrtava programsko ostvarenje algoritma. Za zadani niz $x[]$, program računa novi signal koji sprema u niz $result[]$. Rezultat nije samo dobiveni parametar, nego se pamti i veličina otvora za pojedinu poziciju. Iz pseudokoda se također vide tipične vrijednosti parametara Γ – 4.4, σ – 0.2 i R_c – 0.85 korištene prilikom testiranja. Također je važno primijetiti kako je algoritam kvadratne vremenske složenosti, za razliku od prethodno opisanog algoritma linearne vremenske složenosti.

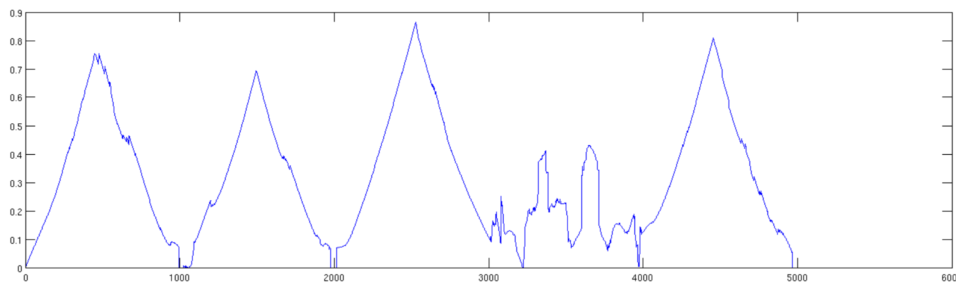
Gore opisani algoritam radi samo slijeva na desno, što znači da će se bolje usrednjiti elementi u lijevom dijelu svakog područja. Zbog toga treba isti algoritam provesti i zdesna na lijevo i usrednjiti dobivene rezultate. Ako su $b_L(n)$ i $b_R(n)$ dobivene vrijednosti parametra prema lijevo i prema desno, a $k_L(n)$ i $k_R(n)$ odgovarajuće širine otvora, konačna vrijednost parametra se dobije težinskom sumom:

$$b(n) = \frac{k_L(n)b_L(n) + k_R(n)b_R(n)}{k_L(n) + k_R(n)}. \tag{3.18}$$

3.5.2. Primjena RICI metode za odabir adaptivnog parametra

RICI metodu je najjednostavnije primijeniti direktno na nizu parametara dobivenih fiksnom veličinom vremenskog prozora, npr. 21 za $N = 1000$. Ako povećamo prozor, prijelazi će doći do izražaja, što je nepoželjno. Treba pripaziti da veličinu prozora prilikom linearne regresije parametra b ne valja miješati sa veličinom lijevog i desnog otvora RICI metode. Zato će se riječ *prozor* koristiti u kontekstu linearne regresije, a *otvor* u kontekstu RICI metode (gdje razlikujemo lijevi i desni otvor).

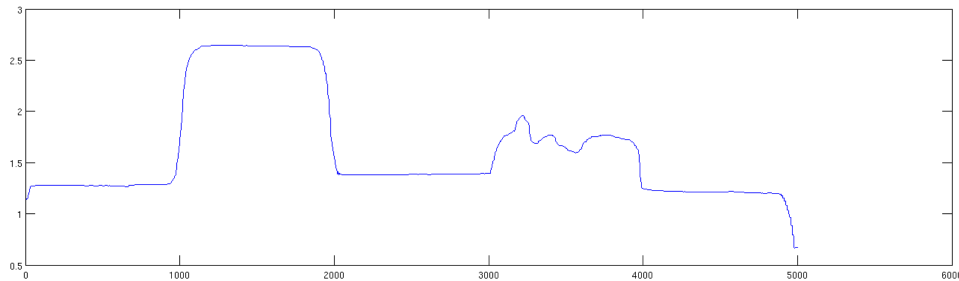
Ako odaberemo samo jednu (malu) širinu prozora, neće biti iskorišteni veći prozori, na koje šum manje utječe. U slučaju previše šuma RICI neće moći sve najbolje izgladiti. Sljedeća metoda koristi i veće prozore, a konačni rezultat se dobije računanjem težinske sume vrijednosti parametra za različite prozore. Najjednostavnije je sve težine postaviti na 1, što je efikasno aritmetička sredina. Kompliciranije ideje uključuju procjenu pozicije unutar područja za svaki element kao vrijednost između 0 i 1 – 0 za rub, a 1 za sredinu područja. Težine se onda određuju tako da veći prozori imaju prednost na sredini područja, a manji na rubu.



Slika 3.8: Procjene pozicija unutar područja za signal duljine 10000

Na slici 3.8 je prikazana procjena pozicije za isti signal kao i prije samo 10 puta veći. Slika 3.9 prikazuje rezultat gore navedenog algoritma na istom primjeru kao i

prva opisana metoda. Parametara je otprilike 5000 i korišteno je 10 različitih veličina prozora vrijednosti između 50 i 500.

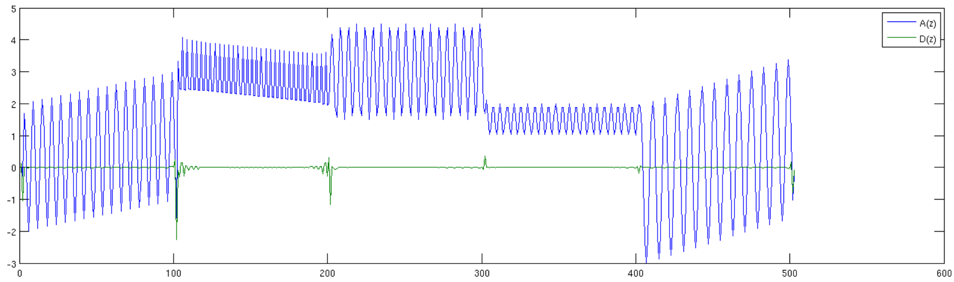


Slika 3.9: Rezultati primjene RICI metode za odabir parametra b

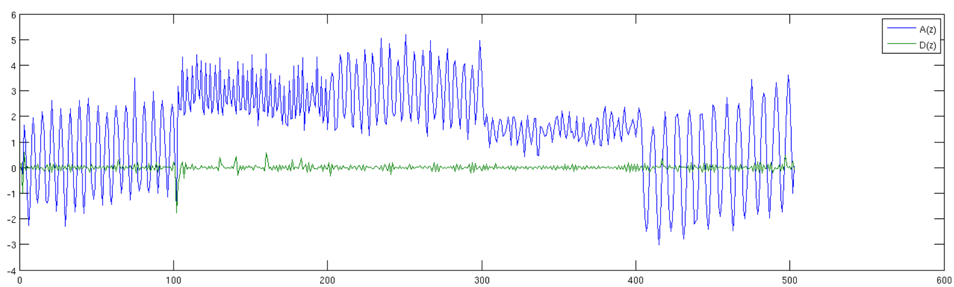
3.6. Usporedba dviju metoda

Obje su metode pokazale dobre rezultate iako svaka ima svoje prednosti i mane. Dok je prva metoda vremenski manje zahtjevna (složenost $O(n)$), jako je ovisna o izboru parametara. S druge strane uporaba RICI metode na bilo koji način daje stabilne rezultate. Pošto je vremenska složenost same RICI metode $O(n^2)$ i priroda samog algoritma takva da se podaci računaju neovisno, RICI metodu je jednostavno paralelizirati, što slijedi u nastavku rada.

Sljedeće slike prikazuju signale $A(z)$ i $D(z)$ dobivene sustavom 2.4. Pritom je parametar b odabran pomoću RICI metode. Ako se koristi metoda intervala dozvole, dobije se gotovo identičan rezultat. Slika 3.10 je rezultat analize čistog signala sa slike 3.1, dok je na 3.11 prikazan rezultat istog signala sa dodanim šumom 3.3. Može se vidjeti kako je za čisti signal detalj praktički nula na cijelom području signala osim na prijelazima, dok šumoviti signal ima nešto lošije rezultate.



Slika 3.10: Aproksimacija i signal dobiveni za čisti signal $x(z)$



Slika 3.11: Aproksimacija i signal dobiveni za šumoviti signal $y(z)$

4. Paralelizacija

Paralelizacija je programska paradigma u kojoj se više procesa, odnosno dretvi, izvodi u potpunosti istovremeno. I dok se to na prvi pogled čini potpuno prirodno i trivijalno, većina procesa na računalu se izvodi tek naizgled paralelno. Velika brzina procesora omogućava da se različiti procesi izvode naizmjence kako bi se postigao privid paralelizma. Kako u zadnje vrijeme fizikalna ograničenja tranzistora sprječavaju rast frekvencije procesora, tendencije vode na drugu stranu, prema uvođenju višejezgrenih procesora. Takvi procesori zaista mogu paralelno izvoditi više procesa istovremeno.

Sukladno s time razvijani su grafički sustavi (GPU – eng. *Graphical Processing Unit*), koji zahtijevaju velik broj operacija u stvarnom vremenu kako bi se vjerno prikazale animacije visoke rezolucije. U stvarnosti to znači obradu velike količine podataka, ali jednostavnim operacijama i bez kompleksnog programskog toka, kakav je tipičan za programe pisane za CPU (eng. *Central Processing Unit*). To je dovelo do razvoja masivno paralelnih višejezgrenih sustava koji su prikladniji za takve poslove. CUDA (eng. *Compute Unified Device Architecture*) je jedan primjer takve arhitekture razvijene od strane Nvidie, koja omogućava programerima da pišu programe u jezicima poput C-a i C++-a.

U nastavku je prikazan kratki opis CUDA-e, primjena na RICI metodu i usporedba s implementacijom na CPU. Više o CUDA arhitekturi se može pročitati iz [3] i [5].

4.1. Općenito o CUDA-i i parelizaciji

Kao što je spomenuto u uvodu, CUDA omogućava programerima uporabu viših programskih jezika kako bi izvršavali kod na grafičkim procesorima razvijenim od strane Nvidie. SDK (eng. *Software Development Kit*) s kojim CUDA dolazi u paketu tako omogućava programeru da piše kod u C-u ili C++-u, a u zadnje vrijeme postoje vanjske biblioteke za rad s Javom, Pythonom, Perlom i drugim programskim jezicima. Krenimo od toka izvođenja tipičnog CUDA programa pa će kasnije biti objašnjeni detalji arhitekture.

Iako se radi o programiranju grafičkog procesora, život programa počinje na CPU. I kad bi se se sva računanja trebala obaviti na GPU, podatke bi zaista najprije trebalo nekako dobaviti do centralne memorije, kako bi se onda ti podaci kopirali u memoriju GPU. Slijedi poziv jedne ili više funkcija pod nazivom *kerneli* koji obavljaju sva potrebna računanja. Kad je to gotovo, slijedi ponovno kopiranje podataka na glavnu memoriju računala, oslobađanje memorije na GPU ako se više neće upotrebljavati i nastavak rada s tim podacima.

To je otprilike tok programa, no treba se zapitati kad se uopće isplati paralelizacija? Još k tome treba uzeti u obzir da memorijska kopiranja između glavne memorije i memorije GPU iziskuju neko vrijeme zbog latencije sabirnice. Npr. treba napisati program koji učitava dva niza jednake duljine n i ispisuje sumu po elementima. Pretpostavimo da učitavanje ili ispisivanje niza košta točno n operacija, kao i sama operacija zbrajanja. Prema tome na CPU, program košta $4n$ operacija. Ako i pretpostavimo da će CUDA beskonačno ubrzati svoj posao (zbrajanje nizova), ubrzanje je tek za faktor 1.33, jer preostaje učitavanje 2 niza i ispisivanje rezultata. To ograničenje se formalno zove Amdahlov zakon:

$$S(N) = \frac{1}{(1 - P) + \frac{P}{N}} \tag{4.1}$$

Gornji izraz predstavlja maksimalno ubrzanje koje se može postići s N procesora ako je udio posla koji se može paralelizirati upravo P.

4.2. Logička i memorijska organizacija CUDA arhitekture

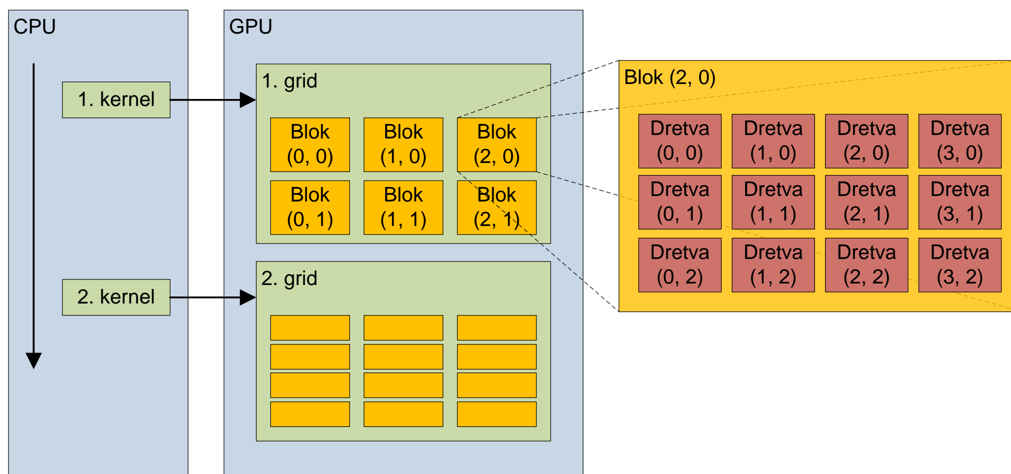
Spomenuto je kako nakon kopiranja podataka na memoriju grafičkog procesora poziv kernela. Jedno od značenja engleske riječi kernel jest klip kukuruza, koji ga simbolično predstavlja. To je tako jer se pozivom kernela stvori veliki broj dretvi (analogno zrnima) koje su organizirane na specifičan način. Pozivom kernela stvara se jedan *grid*, koji je na vrhu te hijerarhije, i predstavlja 2-dimenzionalno polje *blokova*. Blokovi su na sredini hijerarhije i ispod njih se nalaze same dretve, smještene u 3-dimenzionalno polje. Prilikom pozivanja kernelu, moraju mu se predati dimenzije grida i blokova u njima. Svaka dretva tijekom izvođenja ima svoju poziciju u tom prostoru[1] i na temelju toga mora odrediti koje podatke obrađuje. Neka je za primjer zadana matrica dimenzija 200×200 i svaku dretvu treba mapirati na jedan element matrice. Ako odredimo

[1]Malo čudno, ali se to sasvim legitimno može promatrati kao 5-dimenzionalni prostor

da su blokovi veličina $16 \times 16 \times 1$, treba postaviti grid na dimenzije 13×13 kako bi se pokrila cijela matrica. Kojem retku i stupcu pripada dretva se odredi na sljedeći način:

$$\begin{aligned} redak &= blockIdx.y * gridDim.y + threadIdx.y * blockDim.y \\ stupac &= blockIdx.x * gridDim.x + threadIdx.x * blockDim.x \end{aligned} \tag{4.2}$$

`blockIdx` i `threadIdx` su koordinate dretve, a `gridDim` i `blockDim` dimenzije grida i blokova. `threadIdx` i `blockDim` također sadrže i koordinatu `z`, no ona ovdje nema koristi.



Slika 4.1: Logička organizacija CUDA arhitekture

Sa slike 4.1 se vidi gore opisana logička organizacija. Osim logičke organizacije, ključna je hijerarhije memorija, od kojih su najbitnije:

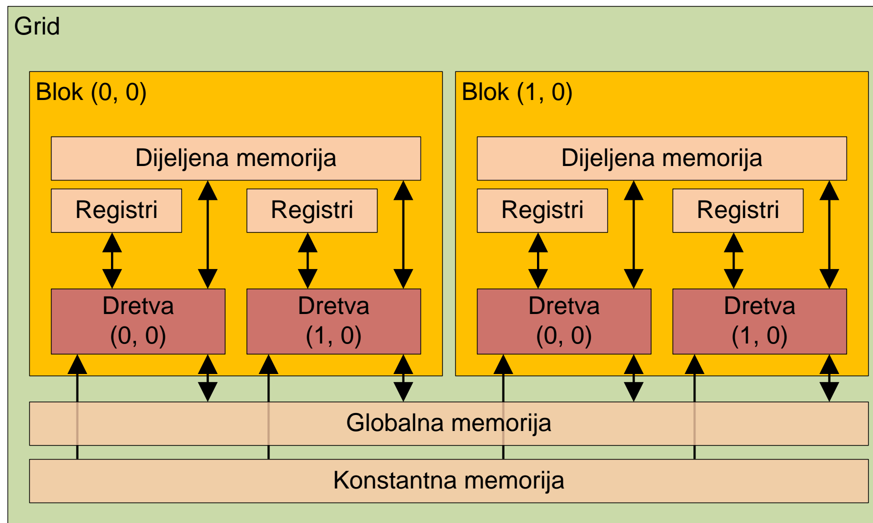
Globalna memorija
Memorija s najvećim kapacitetom (i preko 1 GB), ali najmanjom brzinom (oko 600 ciklusa po operaciji).

Konstantna memorija
Mali dio (64 KB) globalne memorije koji samo omogućava čitanje i koristi *cacheiranje* kako bi značajno ubrzao pristup.

Dijeljena memorija
Memorija još manjeg kapacitet (48 KB po multiprocesoru), koja pripada individualnim blokovima. Brzog je pristupa (nekoliko ciklusa) i ključna je za optimizaciju.

Registri

Ukupno 32 KB kapaciteta najveće brzine (1 ciklus) koji se jednoliko raspodjeljuju dretvama. Koristi se za lokalne varijable.



Slika 4.2: Memorijska hijerarhija CUDA arhitekture

Kako bi se postigla maksimalna optimizacija, potrebno je što bolje iskoristiti dijeljenu i konstantnu memoriju ako je to moguće. Osim toga je jako bitno na koji način se pristupa samoj memoriji, naročito globalnoj. Unutar bloka su dretve podijeljene u tzv. *warpove*, grupe od 32 uzastopne dretve, koje izvode isti strojni kod. Ako jedan warp pristupa uzastopnom bloku memorije, to će se puno brže izvest nego da se memoriji raspršeno adresira. Posljednja stvar koju je nužno spomenuti je sam tok programa. Pošto warpovi dijele strojni kod, treba osigurati da dretve unutar warpa prate ide isti tok, ako je to moguće. To znači da treba minimizirati grananja i naredbe kontrole toka, kako se ne bi dogodilo da dio dretvi unutar warpa slijedi jedan tok, a dio dretvi slijedi drugi.

4.3. Paralelizacija RICI metode

Prisjetimo se kako RICI algoritam radi. Za svaki element niza, vrti se petlja u kojoj se postupno povećava broj elemenata niza, počevši od tog elementa. Kad se postigne određen uvjet, petlja prekida i za dobivene elemente (otvor) se računa srednja vrijednost. Dobivena srednja vrijednost je rezultat za taj element. To je jako pogodno za paralelizaciju pošto se svaki element rezultata može neovisno izračunati. Rješenje

koje je implementirano iskorištava isključivo globalnu memoriju i oslanja se na činjenicu da CUDA ima interni *cache* koji uvelike smanjuje vrijeme dohvata iz memorije [6]. U nastavku je opisano kako su podaci predstavljeni u memoriji CUDA-e, kako se posao raspodjeljuje dretvama i što dretve rade. Kad su sve dretve gotove, preostaje samo rezultate kopirati u glavnu memoriju i usrednjiti lijeve i desne obilaske nizova na temelju RICI otvora.

Kako bi se smanjio broj memorijskih kopiranja između glavne memorije i GPU memorije, ali i broj poziva kernela, jedan kernel je napisan kako bi odjednom obradio više signala. U tu svrhu se svi signali organiziraju u jednu veliku matricu $2n \times m$, gdje je n broj signala, a m duljina signala. To je pogodno ako su signali podjednake duljine, kao što je slučaj prilikom odabira parametra b, a ujedno je i najjednostavnije rješenje. Kad to ne bi bilo tako trebalo bi se naći drugačije rješenje, npr. sve signale spremiti u jedan veliki niz uz pamćenje polja pokazivača na početke odgovarajućih signala. Matrica ima $2n$ redova jer se ulazni signali također obrnuto kopiraju i na taj način obrađuju u lijevo. To zahtjeva nešto više memorije, ali je zato kernel jednostavniji. Nakon što je alociran memorijski prostor za podatke i podaci su kopirani, treba alocirati dvije matrice za rezultate – matrica za odšumljivanje signale i matrica za veličinu RICI otvora.

Elementi ulazne matrice se dretvama dodjeljuju na sljedeći način. Dretve su organizirane u blokove dimenzija 16×16. Ulazna matrica se onda doslovno "poploči" takvim blokovima. Indeks signala i pozicija elementa se onda određuju izrazima 4.2. Sljedećim izrazima se određuju dimenzije grida.

$$grid_width = \left\lceil \frac{m}{16} \right\rceil$$

$$grid_height = \left\lceil \frac{n}{16} \right\rceil$$

Sad kad su dretve dobile svoj posao, naprosto svaka od njih izvrši unutarnju petlju RICI algoritma, gotovo identičnu onoj u pseudokodu i zapišu svoj rezultat. Matrice se potom kopiraju u glavnu memoriju i obrade, kako bi se dobila konačni rezultati uklanjanja šuma. U nastavku je prikazan središnji dio algoritma - sam kernel.

RICI kernel

```
1  typedef float Decimal;
2
3  template <class T> __device__ T min(T &a, T &b) { return a < b ? a : b; }
4  template <class T> __device__ T max(T &a, T &b) { return a > b ? a : b; }
5
6  __global__ void denoise_kernel(const Decimal *signals,  // ulazni signali
7                                 Decimal *res_param,      // rezultati - odsumljeni signali
```

```
8                               int *res_wsize,       // rezultati - velicine prozora
9                               const int n,          // broj signala
10                              const int m,          // duljina signala
11                              const Denoise::CUDAICIDenoiser denoiser // postavke
12                              ) {
13    // 1. odredi pocetne odgovorosti dretve
14    // signal_pos - gdje smo u signalu; signal_index - o kojem signalu se radi
15    int signal_pos = blockIdx.x * blockDim.x * denoiser.thread_jobs + threadIdx.x;
16    int signal_index = blockIdx.y * blockDim.y + threadIdx.y;
17
18    if(signal_index >= 2 * n) return;
19
20    // pomocne varijable za RICI metodu
21    Decimal sum, avg, tavg, curr_sigma, rk, maxlb, minub;
22    int wsize;
23
24    for(int job = 0; job < denoiser.thread_jobs; ++job, signal_pos += blockDim.x) {
25      if(signal_pos >= m) return;
26
27      // inicijaliziraj pomocne varijable za RICI metodu
28      sum = avg = signals[signal_index * m + signal_pos];
29      maxlb = -1e7; minub = +1e7;
30      wsize = -1;
31
32      // povecavaj velicinu prozora do prekida
33      for(wsize = 2; signal_pos + wsize - 1 < m; ++wsize) {
34        // izracunaj trenutnu sumu, prosjek i devijaciju
35        sum += signals[signal_index * m + signal_pos + wsize - 1];
36        tavg = sum / wsize;
37        curr_sigma = denoiser.sigma / sqrtf(wsize);
38
39        // izracunaj novu max. donju i min. gornju granicu i RICI paramear Rk
40        minub = min(minub, (Decimal)(tavg + denoiser.gamma * curr_sigma));
41        maxlb = max(maxlb, (Decimal)(tavg - denoiser.gamma * curr_sigma));
42        rk = (minub - maxlb) / (2 * denoiser.gama * curr_sigma);
43
44        // prekini petlju ako je uvjet zadovoljen
45        if(minub < maxlb || rk < denoiser.rc) break;
46        avg = tavg;
47      }
48
49      // zapisi rezultat
50      res_param[signal_index * m + signal_pos] = avg;
51      res_wsize[signal_index * m + signal_pos] = wsize-1;
52    }
53  }
```

Kernel je zapravo obična funkcija koja ima poseban tretman. Ono što je čini posebnom je ključna riječ $__global__$, koja označava da je funkcija kernel, tj. da se pokreće sa CPU-a i izvodi na GPU. Osim takvih postoje i $__device__$ i $__host__$ funkcije, koje dozvoljavaju da pozivanje samo sa GPU, odnosno CPU. U kodu su prikazane pomoćne funkcije $min()$ i $max()$ koje se pozivaju iz kernela i izvode isključivo na GPU.

Kernel kao parametre prima pokazivače na alocirana polja na GPU sa podacima $signals$, ali i za rezultate res_param i res_wsize. Osim toga tu su i početni broj signala n i veličina signala m. Varijabla $denoiser$ je struktura u kojoj se pamte RICI parametri $sigma$, $gamma$ i rc, ali i broj polja koji jedna dretva obradi. U slučaju da je previše podataka, jedna dretva za svako polje polju bi bilo previše zbog čega svaka dretva dobije više pozicija na istom signalu za obraditi (njih $denoiser.thread_jobs$). Najprije se određuje početna pozicija i indeks signala koji dretve obrađuje. Slijedi deklaracija pomoćnih varijabli i konačno, sama obrada. Svaka dretva obrađuje više zadataka, čemu služi vanjska petlja. Kad je gotova sa zadatkom, petlja ne obrađuje sljedeću poziciju, već onu za $blockDim.x$ veću od trenutne, kako bi dretve istog warpa adresirale susjedne pozicije u globalnoj memoriji. Unutarnja petlja predstavlja RICI metodu i gotovo je identična unutarnjoj petlji na prije prikazanom pseudukodu. Kad ona prekine, rezultati se zapisuje u izlazne matrice, a dretva nastavlja dalje s poslom ili prekida kad je svoj posao obavila.

4.4. Rezultati

Mjeren je čitav program za adaptivnu valićnu transformaciju, koji uključuje odabir parametra b primjenom RICI metode sa usrednjavanjem više razina prozora. Generirani su primjeri signala bez šuma kao onaj na slici 3.1, samo sa različitim brojem uzoraka. Osim signala, mijenja se i broj prozora uključen u odabir parametra b i kreće se od 10 do 50.

Programi su mjereni na računalu Sigma na ZESOI-u[2], na Fakultetu elektrotehnike i računarstva. Procesor na kojem su programi mjereni je dvojezgreni Intel Core2Duo 6400 frekvencije 2.13 GHz i 2 MB L2 *cache*-a. Radne memorije ima 4 GB, što je više nego dovoljno za oba programa. Grafički procesor je Nvidia GeForce 570, koja predstavlja vrh na tržištu grafičkih procesora.

Tablice 4.1 i 4.2 prikazuju rezultate mjerenja. U prvom stupcu su pobrojane vrijednosti K – koliko RICI metoda različitih širina prozora odšumljuje prilikom odabira parametra b. Za $K = 10$, parametar b se određuje usrednjavanjem dobivenog parametra za 10 različitih veličina prozora. U prvom redu tablica su prikazane duljine signala N koji ulaze u program. Treba uzeti u obzir da će se signali prije RICI metode decimirati odnosno skratiti za pola, tako da će RICI metoda zapravo obrađivati duplo kraće signale. Sve ostale ćelije predstavljaju vremena (u sekundama) izvođenja programa za

[2] Zavodu za elektroničke sustave i obradbu informacija

Tablica 4.1: Vremena RICI metode na CPU (u sekundama)

	500	1000	5000	10000	20000	50000
10	0.01	0.04	0.96	3.49	13.16	78.32
20	0.02	0.07	1.84	7.00	26.30	156.87
30	0.04	0.13	2.76	10.50	39.63	235.17
40	0.05	0.18	3.70	13.89	52.66	313.95
50	0.07	0.23	4.59	17.36	65.82	392.26

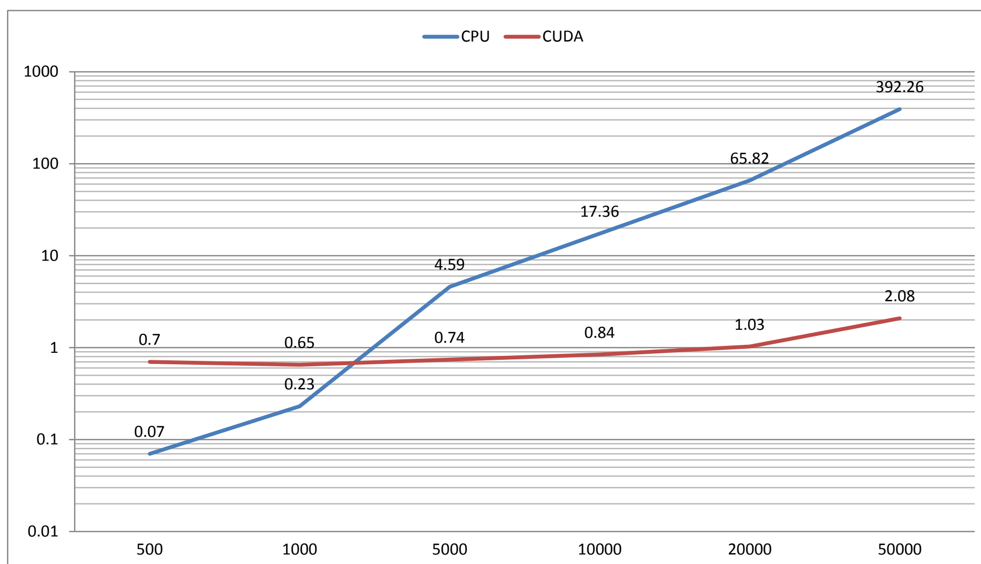
Tablica 4.2: Rezultati paralelizacije RICI metode na CUDA-i (u sekundama)

	500	1000	5000	10000	20000	50000
10	0.67	0.68	0.71	0.69	0.77	1.01
20	0.68	0.69	0.73	0.79	0.89	1.30
30	0.68	0.63	0.72	0.77	0.88	1.53
40	0.68	0.67	0.76	0.81	1.00	2.81
50	0.70	0.65	0.74	0.84	1.03	2.08

određenu kombinaciju duljine signala N i broja veličina prozora K. Pritom tablica 4.1 prikazuje rezultate na jednodretvenom programu na CPU, a 4.2 rezultate paraleliziranog programa na CUDA arhitekturi. Za mjerenja je korišten Unix alat `time`.

Iz navedenog se vidi da je za manje količine podataka prikladniji CPU, jer sama inicijalizacija CUDA-e zahtjeva oko 0.6 sekundi. No s druge strane, za veće ulaze CUDA dominira pošto se vremenska složenost praktički smanji s kvadratne na linearnu ako postoji dovoljno velik broj dretvi[3].

[3] Formalno je možda preciznije reći kako se složenost dijeli sa jako velikom konstantom



Slika 4.3: Vremena u sekundama za $K = 50$, logaritamske skale

5. Zaključak

U radu su uspješno postignuta oba cilja: optimalan odabir adaptivnih parametara i vremenska optimizacija paralelizacijom na CUDA arhitekturi.

Dvije metode koje biraju adaptivni parametar na temelju linearne regresije prozora različitih veličina su uspješno poboljšali svojstva analize signala sa ili bez šuma. I dok je jedna robusnija, tako je druga brža, te kao i mnoge stvari u životu obje metode predstavljaju nekakav kompromis. Linearna regresija je implementirana korištenjem *L2* norme, no ako se koristi *L1* norma prijelazi između područja gotovo i ne postoje [8]. S druge strane, linearna regresija minimizacijom *L1* norme zahtjeva još više računalnih resursa i algoritme koji se u konačnici ni ne mogu paralelizirati.

Paralelizacija RICI metode na CUDA arhitekturi se također pokazala kao pravi pogodak. Iako bi netko mogao pomisliti da bi rijetko bilo potrebe za paralelizacijom signala duljine 25000, to se da osporiti. Ako za primjer uzmemo obradu slike, tipične slike u masovnoj uporabi jesu manjih dimenzija, no i slike su u dvije dimenzije i tu bi se postigla optimizacija ako bi se htio ukloniti šum po redovima. Uostalom možda bi u nečijem projektu baš bilo potrebe za uklanjanjem šuma na signalu duljina više desetaka tisuća i ovo bi bilo više nego korisno.

Što se tiče budućnosti, logičan nastavak ovog rada bio bi iskoristiti sve što je napravljeno u svrhu kompresije i uklanjanja šuma u signalima, te primjenu na konkretnim slikama ili nekom drugom mediju.

LITERATURA

[1] Linear least squares (mathematics). `http://en.wikipedia.org/wiki/Linear_least_squares_(mathematics)`.

[2] Linear regression. `http://en.wikipedia.org/wiki/Linear_regression`.

[3] Wen-mei W. Hwu David B. Kirk. *Programming Massively Parallel Processors*. Morgan Kaufmann, 2010.

[4] J. Lerga, M. Vrankić, i V. Sučić. A Signal Denoising Method Based on the Improved ICI Rule. *IEEE Signal Processing Letters, vol. 15*, 2008.

[5] NVIDIA. *NVIDIA CUDA C Programming Guide*. 2010.

[6] F. Pavetić. CUDA implementacija računanja elektrostatske raspodjele naboja. Fakultet elekrotehnike i računarstva, Sveučilište u Zagrebu, 2011.

[7] Dr. W. J. Phillips. Wavelets and filter banks course notes, 2003.

[8] A. Sović i D. Seršić. Robustly Adaptive Wavelet Filter Bank Using L1 Norm. In Proc. of the IWSSIP, Bosnia and Herzegovina, 2011.

Adaptivna valićna transformacija ostvarena na CUDA arhitekturi

Sažetak

U radu je predstavljena adaptivna diskretna valićna transformacija, u kojoj se adaptivni parametri se računaju linearnom regresijom nad vremenskim prozorom. Ako je prozor prevelik, prijelazna područja signala su preočita, a ako je premalen, utjecaj šuma je prevelik. Predstavljene su dvije metode kojima se adaptivni parametar procjenjuje na temelju većih i manjih veličina prozora. Osim toga, jedna metoda je uspješno paralelizirana na CUDA arhitekturi, što doprinosi velikom ubrzanju za veće ulaze.

Ključne riječi: brza valićna transformacija, linearna regresija, paralelno programiranje, CUDA

Adaptive wavelet transform implemented on CUDA platform

Abstract

This thesis deals with adaptive discrete wavelet transform, which contains adaptive parameters chosen using linear regression over small sections of the given signal. The small sections are calculated using a box function of chosen width. If the width is too big, the transition over some areas of the given signal are too obvious, but if the width is too small, noise affects the result too much. Two methods which try to compromise between the two extremes are presented. Apart from that, results of successfully implementing one of the two methods on CUDA arhitecture are shown.

Keywords: fast wavelet transform, linear regression, parallel programming, CUDA