VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

NÁVRH DIGITÁLNÍCH FILTRŮ NA GPU

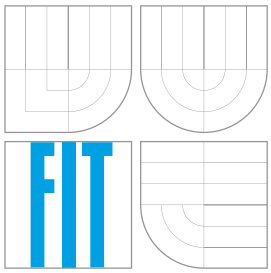
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

FILIP VAVERKA

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

NÁVRH DIGITÁLNÍCH FILTRŮ NA GPU

DIGITAL FILTER DESIGN ON GPU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — FILIP VAVERKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. LUKÁŠ POLOK
SUPERVISOR

BRNO 2013

Abstrakt

Tato práce ukazuje jeden z přístupů k návrhu digitálních filtrů s nekonečnou impulzní odezvou a volitelným řádem. Prezentované řešení je založeno na evolučním genetickém algoritmu a umožňuje tedy přímý návrh filtru dle jeho specifikací. Hlavním přínosem práce je paralelní implementace genetického algorimu, která je akcelerovaná pomocí GPU. Filtry jsou navrhovány v kaskádové reprezentaci. Řešení také umožňuje specifikovat požadovanou jak frekvenční, tak fázovou charakteristiku filtru.

Abstract

This thesis shows one of the approaches to the design of ditigal filters with infinite impulse response and specified order. The proposed solution is based on an evolutionary genetic algorithm and therefore allows for direct filter design from its specification. Its main contribution to this subject is that the implementation is parallel and it is acceleraded by GPU. The filters are designed in cascade representation. It also allows to specify both, the desired frequency and phase characteristics of filters.

Klíčová slova

Návrh digitálního filtru, zpracování signálu, GPU, OpenCL, genetický algoritmus, návrh kaskádového filtru, přímý návrh IIR filtru

Keywords

Digital filter design, Signal processing, GPU, OpenCL, Genetic algorithm, Cascade filter design, Direct IIR filter design

Citace

Filip Vaverka: Návrh digitálních filtrů na GPU, bakalářská práce, Brno, FIT VUT v Brně, 2013

Návrh digitálních filtrů na GPU

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Lukáše Poloka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Filip Vaverka
9. května 2013

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu práce Ing. Lukáši Polokovi za jeho přínos.

© Filip Vaverka, 2013.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **2**

2 Zpracování a analýza signálů **3**
2.1 Vlastnosti systémů s diskrétním časem 4
2.2 Digitální filtry 4
2.2.1 Reprezentace digitálního filtru 5
2.2.2 Analýza charakteristiky digitálního filtru 6
2.2.3 Tradiční metody návrhu 8
2.2.4 Stabilita digitálního filtru 9

3 Genetické algoritmy **10**
3.1 Evoluční genetické algoritmy 10

4 Platforma OpenCL **12**
4.1 Struktura platformy OpenCL 12
4.2 Cílový hardware 13

5 Návrh rozhraní knihovny **15**

6 Návrh filtrů pomocí genetických algoritmů **17**
6.1 Genetický algoritmus 17
6.2 Reprezentace filtrů 17
6.3 Genetické operátory a jejich implementace 18
6.3.1 Operátor selekce 18
6.3.2 Operátor křížení 19
6.3.3 Operátor mutace 20
6.4 Generování náhodných filtrů 22
6.4.1 Generátory pseudonáhodných čísel 22
6.4.2 Generování náhodných filtrů 23
6.5 Vyhodnocení kvality filtrů 24

7 Vlastnosti knihovny **26**
7.1 Výkonnostní charakteristiky 27
7.2 Konvergence algoritmu 28

8 Závěr **30**

Seznam příloh **32**

Kapitola 1

Úvod

Tato práce se zabývá návrhem a implementací knihovny pro návrh digitálních filtrů pomocí evolučního genetického algoritmu. Použití digitální filtrace se díky svým vlastnostem rozšiřuje do mnoha oblastí zpracování signálů různého typu. Pro návrh digitálních filtrů se tradičně používají metody založené na transformaci výchozího analogového filtru na digitální filtr s požadovanými vlastnostmi. V této práci je však popisováno méně rozšířené řešení vycházející z optimalizačních algoritmů. Toto řešení umožňuje návrh kaskádových filtrů s nekonečnou impulzní odezvou, které jsou specifikovány svým řádem a požadovanou frekvenční a fázovou charakteristikou. Jde tedy o přímé hledání takových koeficientů filtru, aby výsledný filtr co nejlépe odpovídal specifikaci a zároveň byl stabilní. Stabilita filtrů v populaci je zajištěna zvolenými genetickými operátory a není ji nutné v průběhu výpočtu ověřovat. Koeficienty filtrů jsou reprezentovány reálnými hodnotami a genetický algoritmus je tedy spojitý.

Podobný přístup je prezentován například již v pracích [8] a [1]. Jedním z přínosů této práce, oproti zmíněným, je však to, že popisované řešení využívá přirozeného paralelizmu genetického algoritmu a k jeho akceleraci využívá platformu OpenCL, která je blíže rozebírána v kapitole 4. Výhodou zvolené platformy je především unifikovaný přístup k výpočetním zdrojům cílové hardwarové platformy, za kterou byl zvolen grafický akcelerátor. Vlastnosti této platformy do značné míry ovlivňují zvolené algoritmy a jejich implementaci. Tyto vlastnosti jsou proto diskutovány v kapitolách 4.1 a 4.2.

Vlastnosti výsledné implementace jsou zhodnoceny v kapitole 7. V několika prvních oddílech je shrnuta teorie zpracování signálů (kapitola 2) a teorie genetických algoritmů (kapitola 3). Teorie rozebíraná v těchto oddílech je vykládána se zaměřením na oblasti potřebné k řešení daného problému. Konkrétní řešení pak popisují kapitoly 5 a 6.

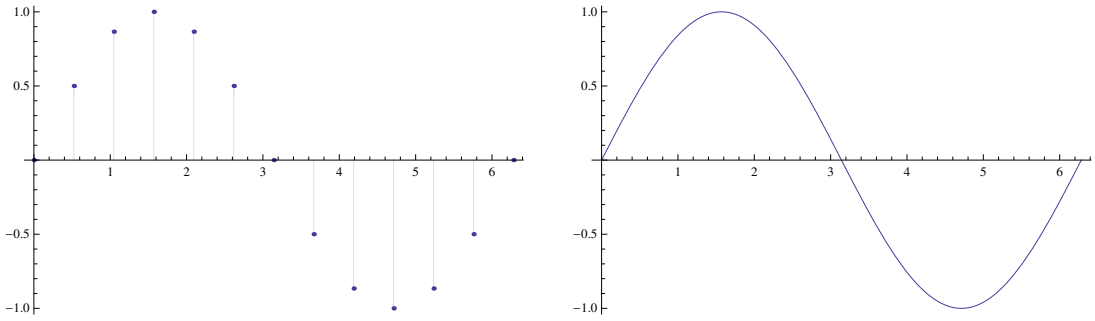
Kapitola 2

Zpracování a analýza signálů

V tomto oddíle budou mimo jiné rozebrány základní pojmy z oblasti zpracování signálů a popsány vlastnosti systémů s diskrétním časem. Následně se zaměřím na podmnožinu těchto systémů, které nazýváme filtry a na některé tradiční metody návrhu digitálních filtrů.

Protože jde o zpracování signálů, prvním pojmem, který je nutné definovat, je signál. Dle [6] je signál definován jako: *„Veličina nebo proměnná, která nese nebo obsahuje nějaký typ informace, která může být přenesena, zobrazena nebo modifikována.“*

Pro tuto práci je zajímavá především oblast, kde signál tvoří nějaká elektrická veličina (v případě analogových systémů) nebo proměnná (digitální systémy). Analogovými systémy jsou pak myšleny systémy, které pracují ve spojitém čase, se spojitými veličinami a jsou založeny na analogových elektrických obvodech. Do této skupiny můžeme zařadit například jednoduchý obvod typu dolní propust. Digitální systémy jsou naopak systémy, které pracují se signálem diskrétním, jehož hodnoty tvoří proměnné s omezeným počtem hodnot. Jde o číslicové zpracování signálu například pomocí digitálního signálového procesoru (DSP), nebo jiného číslicového obvodu. Spojitý signál bude značen jako $x(t)$ nebo $y(t)$, diskrétní pak jako $x[n]$ nebo $y[n]$. Signál je často funkcí času, ale není to nezbytně nutné. Například při filtrování obrazu může být funkcí prostoru.



(a) Diskrétní signál (b) Spojitý signál

Obrázek 2.1: Příklad obou reprezentací signálů

Na signál je možné nahlížet dvěma základními způsoby: buď v časové doméně, kde ho tvoří průběh hodnoty veličiny (proměnné) v čase, nebo v doméně frekvenční, kde můžeme zkoumat zastoupení jednotlivých frekvenčních složek v signálu a jejich fázový posun. Na rozdíl od časové domény, kde bývá signál tvořen skalárními hodnotami, je ve frekvenční

doméně reprezentován komplexními hodnotami. Pro transformaci signálu z časové domény do frekvenční existuje řada metod jak pro spojité, tak i diskrétní systémy (signály). Pro spojité systémy jsou to především Fourierova řada (rozklad periodických signálů) a Fourierova transformace (pro signály tvořené integrovatelnou funkcí). V případě diskrétních systémů lze využít například z-transformaci (rovnice 2.1) nebo diskrétní Fourierovu transformaci (DFT). Zvláštní význam bude mít v této práci právě z-transformace, kterou lze využít k výpočtu frekvenční (a fázové) charakteristiky digitálních filtrů.

$$X(z) = \mathcal{Z}\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n]z^{-n} \tag{2.1}$$

2.1 Vlastnosti systémů s diskrétním časem

V tomto oddíle budou popsány základní vlastnosti diskrétních systémů a jejich význam při návrhu digitálních filtrů. Většinu těchto základních vlastností lze aplikovat i na systémy se spojitým časem.

První takovouto vlastností je kauzalita systému. Ta říká, že systém je kauzální v případě, že jeho výstup závisí pouze na současném a případně také minulém vstupu. Pokud budeme uvažovat diskrétní systém na jehož vstupu je právě vzorek N, pak, jestliže výstup systému závisí pouze na tomto vzorku a na vzorcích předcházejících, je systém kauzální. Filtry a digitální filtry proto spadají (jako většina systémů, které mohou signál zpracovávat proudově) do skupiny kauzálních systémů.

Druhou vlastností je časová invariantnost systému. Ta omezuje vlastnosti systému tak, že systém nesmí měnit své chování v čase. To znamená, že pokud bude vstupem systému signál $x(t)$ a výstupem $y(t)$, pak musí platit, že pro vstup $x(t - t_0)$ je výstupem systému signál $y(t - t_0)$. Filtry splňují i tuto podmínku.

Třetí vlastnost, která se obvykle uvádí, je linearita systému. Aby ji systém splňoval, musí splňovat homogenitu a aditivitu. Mějme systémy s následujícími vstupy a výstupy: $x_1(t) \implies y_1(t)$ a $x_2(t) \implies y_2(t)$. Pro homogenitu pak platí, že $ax_1(t) \implies ay_1(t)$ a pro aditivitu platí, že $x_1(t) + x_2(t) \implies y_1(t) + y_2(t)$. Jednotně můžeme podmínku linearity systému zapsat výrazem 2.2.

$$ax_1(t) + bx_2(t) \implies ay_1(t) + by_2(t) \tag{2.2}$$

Poslední a velice důležitou vlastností systémů je jejich stabilita. V literatuře bývá tato vlastnost označována zkratkou BIBO (bounded input, bounded output), což vyjadřuje i její podstatu, která říká, že výstupem systému s omezeným vstupem bude omezený výstup. Označením „omezený“ pak myslíme, že lze nalézt dvě kladná reálná čísla $B, C < \infty$, pro která platí: $|x(t)| < B \implies |y(t)| < C$. Tato vlastnost má z předchozích největší vliv na řešení problému, z toho důvodu že filtry (a také digitální filtry) mohou být jak stabilní tak i nestabilní. Blíže se problému ověření a zajištění stability digitálních filtrů věnuje kapitola 2.2.

2.2 Digitální filtry

Digitální filtry (dále jen filtry) jsou podmnožinou systémů pro zpracování signálů s diskrétním časem (za předpokladu že signál je diskrétní funkcí času). Jsou to lineární časově invariantní (tzv. LTI) systémy, proto musí splňovat první tři z výše uvedených podmínek.

Častým účelem jejich použití je potlačení určitých frekvenčních složek nebo rozsahů frekvenčních složek v signálu, například odstranění šumu a podobně. Filtry se obvykle dělí do dvou kategorií: filtry s konečnou impulzní odezvou (finite impulse response – FIR) a filtry s nekonečnou impulzní odezvou (infinite impulse response – IIR). Impulzní odezva filtru (nebo jiného systému) je jeho reakce (výstupní signál) na vstupní signál v podobě jednotkového impulzu. Jednotkový impulz je u systémů s diskrétním časem tvořen posloupností vzorků, kde první má hodnotu jedna a ostatní následující nula. V případě systémů se spojitým časem je tvořen Diracovým impulzem. Pomocí diskrétní Fourierovy transformace pak lze z této impulzní odezvy diskrétního systému zjistit jeho frekvenční charakteristiku. V případě digitálních filtrů je však k dispozici výhodnější metoda založená na z-transformaci (rovnice 2.1), která využívá přímo koeficientů filtru a nevyžaduje výpočet jeho impulzní odezvy.

2.2.1 Reprezentace digitálního filtru

Digitální filtr lze reprezentovat různými způsoby, z nichž použití každého je vhodné v jiné situaci. Za základní reprezentaci je možné považovat blokové schéma, kde se vyskytují bloky jako: suma, jednotkové zpoždění (zpoždění o jeden vzorek) a násobení konstantou. Je zřejmé, že tato reprezentace je vhodná pouze pro názornost. Pro algoritmické zpracování a tedy i pro použití v této práci je vhodnější reprezentace pomocí přenosové funkce. Rovnice 2.3 ukazuje přenosovou funkci filtru druhého řádu, který je použit jako základní stavební jednotka filtrů vyšších řádů, navrhovaných zde prezentovaným algoritmem.

$$H(z) = \frac{1 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{2.3}$$

Kde a_1,a_2 a b_1,b_2 jsou koeficienty filtru.

Druhou zde použitou reprezentací digitálního filtru je geometrická reprezentace pomocí diagramu nul a pólů. Jde o jednoduché zobrazení nul a pólů filtru v komplexní rovině. Tato reprezentace je zde využívána při generování nových náhodných filtrů, přičemž důvod použití tohoto způsobu je dále popsán v kapitole 6.

Pro vlastní filtrování signálu digitálním filtrem se často využívá reprezentace filtru pomocí diferenční funkce. Rovnice 2.4 je diferenční rovnicí filtru druhého řádu, která vznikne inverzní z-transformací rovnice 2.3.

$$y[n] = -a_1 y[n-1] - a_2 y[n-2] + x[n] + b_1 x[n-1] + b_2 x[n-2] \tag{2.4}$$

Kde $y[n]$ a $x[n]$ jsou výstupní a vstupní vzorky signálu. Koeficienty filtru jsou opět značeny jako a_1,a_2 a b_1,b_2.

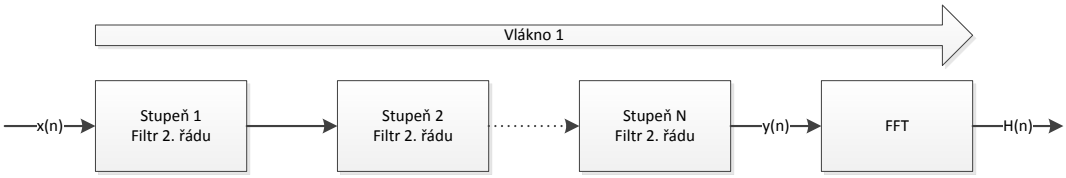
Jak bylo zmíněno výše, filtry vyšších řádů se často tvoří kaskádovým (nebo jiným) spojením několika filtrů nižších řádů. Toto je využito i v této práci a filtry vyšších řádů jsou tvořeny N-ticemi filtrů druhého řádu (dále stupeň). Výstup takového kaskádového filtru pak lze získat například sériovým spojením jednotlivých stupňů, jak znázorňuje obrázek 2.2.

Obrázek 2.2: Struktura kaskádového filtru

2.2.2 Analýza charakteristiky digitálního filtru

Pro zjištění komplexní (tedy frekvenční i fázové) charakteristiky LTI systému a tedy i digitálních filtrů lze využít několik různých postupů. Jelikož během provádění genetického algoritmu dochází k vyhodnocování charakteristik velkého počtu filtrů, je nutné zvolit co možná nejoptimálnější a výpočetně nejméně náročný způsob. Z tohoto důvodu jsou v této kapitole popsána dvě možná řešení. Výhoda prvního řešení je možnost analýzy i neznámého LTI systému, což ovšem pro použití v této práci není nutné a lze ho postupně upravit na řešení druhé.

První řešení je založeno na Fourierově transformaci impulzní odezvy analyzovaného systému. Obecně by se pak tento způsob dal implementovat tak, že je na vstup systému přiveden jednotkový impulz. Na výstupu systému pak dostáváme jeho impulzní odezvu, nad kterou provedeme Fourierovu transformaci a dostáváme charakteristiku daného systému v komplexní rovině. Ve zde popisovaném případě, kdy systém tvoří kaskádový filtr, je nutné nechat signál projít všemi jeho stupni a pak teprve provést transformaci. Toto řešení zachycuje schéma 2.3. Nevýhodou tohoto řešení je malá možnost paralelizmu, způsobená tím, že signál prochází sériově jednotlivými filtry. Tento problém by se dal minimalizovat využitím jedné z vlastností konvoluce 2.5, která říká, že výstup LTI systému lze získat konvolucí jeho vstupu s jeho impulzní odezvou 2.6.



Obrázek 2.3: Výpočet odezvy kaskádového filtru

$$(f * g)[n] \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f[m]\, g[n-m] \tag{2.5}$$

Kde f, g jsou sekvence a $(f * g)[n]$ je sekvence, která vznikla jejich konvolucí.

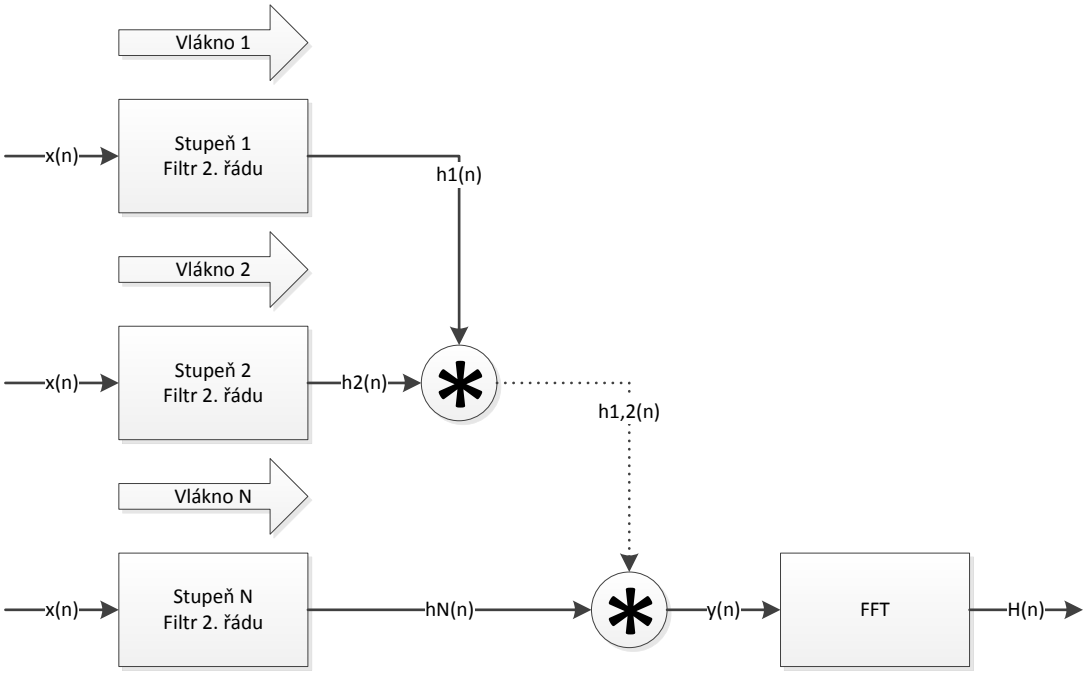
$$y[n] = \sum_{m=0}^{n} h(n-m)x(m) \tag{2.6}$$

Kde $y[n]$ je výstupní signál, $h(n)$ je impulzní odezva LTI systému a $x(n)$ je vstupní signál.

Pokud jsou tedy k dispozici impulzní odezvy všech stupňů, lze postupně získat celkovou impulzní odezvu. Tento proces znázorňuje schéma 2.4. Takto ovšem dochází ke značnému zvýšení výpočetních nároků a především také nároků na datovou propustnost paměťového subsystému, který často bývá úzkým hrdlem masivně paralelních systémů, jakými GPU jsou. Zde je možné využití konvolučního teorému, který říká, že konvoluce signálů v časové doméně přejde v doméně frekvenční v násobení 2.7. Je-li tedy na impulzní odezvu každého stupně filtru nejdříve aplikována Fourierova transformace, pak může být celková charakteristika filtru vypočtena pomocí vynásobení těchto částečných charakteristik jednotlivých stupňů. Tím je tedy operace konvoluce nahrazena výpočtem Fourierovy transformace.

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\} \tag{2.7}$$

Kde f, g jsou funkce a $\mathcal{F}\{f\}$, $\mathcal{F}\{g\}$ jsou jejich Fourierovy transformace.



Obrázek 2.4: Výpočet charakteristiky filtru s využitím konvoluce

Pro úplné odstranění výpočtu Fourierovy transformace je možné využít toho, že se analyzovaný systém skládá pouze ze stupňů tvořených filtry druhého řádu se známými koeficienty. Díky tomu, že jsou koeficienty těchto stupňů známé, lze jejich charakteristiky vypočítat vyčíslením jejich přenosové funkce v požadovaných bodech na jednotkové kružnici v komplexní rovině. Tohoto je možno dosáhnout, jak ukazuje rovnice 2.8, dosazením výrazu $e^{j\omega}$ jako parametru přenosové funkce stupně. Takto je tedy tvořeno ono druhé řešení, které nevyžaduje výpočet konvoluce ani Fourierovy transformace. Jeho výhodou je také možnost výpočtu všech vzorků výsledné charakteristiky filtru paralelně. Kompletní výpočet komplexní charakteristiky filtru by pak bylo možné zapsat funkcí 2.9.

$$hz[n] = \frac{1 + b_1(e^{j\omega})^{-1} + b_2(e^{j\omega})^{-2}}{1 + a_1(e^{j\omega})^{-1} + a_2(e^{j\omega})^{-2}} \tag{2.8}$$

Kde $hz[n]$ je komplexní charakteristika stupně filtru a úhel ω je definován jako $\omega = \pi(n/N)$. N je celkový počet vzorků charakteristiky.

$$H(z) = \prod_{i=1}^{M} \left(\frac{1 + b_{i1}z^{-1} + b_{i2}z^{-2}}{1 + a_{i1}z^{-1} + a_{i2}z^{-2}} \right) \tag{2.9}$$

Kde za z je dosazen výraz $e^{j\omega}$, i představuje číslo stupně daného filtru a M je celkový počet stupňů. a_{i1},a_{i2} a b_{i1},b_{i2} jsou pak koeficienty stupně i. Vyčíslení je možné provést, obdobně jako ve výrazu 2.8, dosazením $\pi(n/N)$ za ω. Není nutné vyčíslovat celou periodu 2π, jelikož pro její vzorky by pak platilo: $H[k] = H^*[N - k]$.

2.2.3 Tradiční metody návrhu

Pro návrh digitálních filtrů se tradičně využívá řada různých metod, které je možné rozdělit do několika skupin. Za základní hledisko může být považováno to, zda je metoda využívána při návrhu filtru typu FIR nebo IIR. Rozšířené jsou metody algebraické, které bývají často založeny na transformaci analogového filtru na digitální. Ale je možné se setkat i s metodami geometrickými, mezi které lze zařadit například metodu umisťování nul a pólů [7]. Některé metody lze použít pouze k návrhu určitých druhů filtrů. Například metody aproximace derivací (Approximation of Derivatives) a impulzní invariance (Impulse Invariance) lze použít pouze pro filtry typu dolní nebo pásmová propust.

Jednou se základních metod návrhu digitálních filtrů typu IIR je bilineární z-transformace (BZT). Tato metoda je založena na transformaci analogového filtru na digitální, mapováním s-roviny na z-rovinu pomocí z-transformace, a je možné ji využít pro návrh filtrů různých typů. Použití a odvození této metody je uvedeno například v [9], kde jsou také uvedeny předchozí zmíněné metody.

2.2.4 Stabilita digitálního filtru

Základem kaskádového filtru navrhovaného popisovaným algoritmem je stupeň tvořený filtrem s nekonečnou impulzní odezvou, jehož přenosovou funkci lze zapsat rovnicí 2.10.

$$H(z) = \frac{1 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{2.10}$$

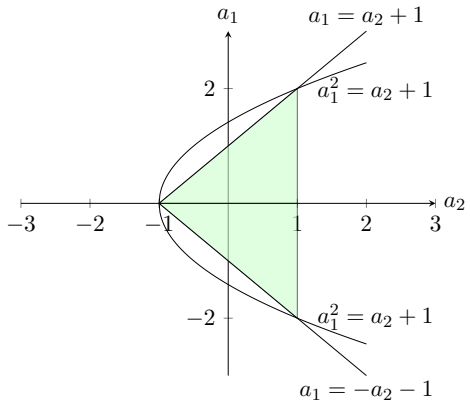
Kde $a_{1,2}$ a $b_{1,2}$ jsou koeficienty filtru a $H(z)$ je jeho přenosová funkce.

Pro zajištění stability celého filtru je nutné aby byly stabilní všechny jeho stupně. Toho lze dosáhnout pouze tím, že se jejich póly (definované rovnicí 2.11) budou nacházet uvnitř jednotkové kružnice v komplexní rovině (z-plane), proto musí platit 2.12.

$$p_1, p_2 = \frac{-a_1 \pm \sqrt{a_1^2 - 4a_2}}{2} \tag{2.11}$$

$$-2 < -a_1 \pm \sqrt{a_1^2 - 4a_2} < 2 \tag{2.12}$$

Koeficient a_2 je součinem pólů, pro zajištění stability tedy pro něj musí platit $|a_2| < 1$. V případě, kdy má polynom dva reálné kořeny musí navíc platit že $|a_1| < |a_2|$. Pokud jde o dvojitý reálný kořen, musí platit $|a_1| < 2$. Pro komplexní kořeny je třeba zajistit, aby $a_1^2 < 2a_2 + 2$. Výslednou, takto omezenou oblast znázorňuje obrázek 2.5. Jde o oblast, ve které se obvykle nachází koeficienty a_1 a a_2 stabilních filtrů.



Obrázek 2.5: Oblast stability koeficientů

Kapitola 3

Genetické algoritmy

Genetické algoritmy (dále GA) jsou rodina nedeterministických algoritmů vhodných k řešení složitých optimalizačních úloh s větším počtem proměnných. Se značnou výhodou je lze také použít pro řešení úloh, kde hrozí uváznutí na lokálně dobrém (ale ne nutně nejlepším) řešení (tzv. uváznutí v lokálním minimu). Naopak v případech, kdy cenovou funkci tvoří například konvexní analytická funkce a počet parametrů je nízký, může být výhodnější zvolit tradiční optimalizační metody. Zde řešený problém (návrh digitálních filtrů) však spadá jak počtem parametrů, tak vlastnostmi cenové funkce do první kategorie a vhodným řešením jsou právě genetické algoritmy.

3.1 Evoluční genetické algoritmy

Evoluční genetický algoritmus je iterativní algoritmus simulující evoluci, jak je obecně známá. V následujícím textu bude vysvětlena činnost evolučních GA a zároveň budou vysvětleny důležité pojmy. Evoluční GA pracují nad množinou kandidátních řešení, tato množina bývá označována jako populace a jedno kandidátní řešení jako jedinec nebo chromozom. Chromozom je tvořen datovou strukturou konečné délky, která přestavuje jedno kandidátní řešení. GA pak provádí nad populací několik operací, pomocí kterých simuluje evoluci a zajišťuje přechod populace do další generace. Tyto operace jsou selekce, mutace a křížení (crossover).

Selekce simuluje přirozený výběr, jinak řečeno vybere určitou podmnožinu jedinců z populace, kteří budou jednak zastoupeni i v následující generaci a také jsou z nich pomocí křížení tvořeni další jedinci. Selekce probíhá na základě hodnoty funkce, která je spojena s každým jedincem v populaci a udává jeho kvalitu. Tato funkce bývá označována jako cena, někdy je výhodnější pracovat s její převrácenou hodnotou, která je označována jako kvalita (případně fitness neboli vhodnost). Jedinci ke křížení jsou pak obvykle vybíráni náhodně na základě určité pravděpodobnosti. Touto dvojící operací je realizován přenos nejlepší genetické informace do další generace. Je zřejmé, že může dojít k situaci, kdy dobré řešení nemůže vzniknout pouze na základě genetické informace obsažené v počáteční populaci a je vhodné simulovat také vliv okolního prostředí na genetickou informaci jedinců v populaci. Tento vliv simuluje operátor mutace, který zajišťuje přísun nové genetické informace a to tak, že projde chromozomy v populaci a s danou pravděpodobností některé z nich náhodně modifikuje. S rostoucí pravděpodobností mutace dochází ke zvětšování prohledávaného prostoru, což může vést k nalezení lepších řešení, ale také ke snižování rychlosti konvergence algoritmu.

Cílem tohoto algoritmu je, aby opakováním tohoto cyklu (generace) vznikala postupně kvalitnější řešení. Algoritmus končí po nalezení jedince s dostatečnou kvalitou. Případně může být ukončen po specifikovaném počtu generací, pak není nalezeno plně vyhovující řešení, ale přesto mohou být dostačující řešení kandidátní (z poslední generace).

Jedna iterace evolučního GA by se dala zjednodušeně popsat takto:

1. Vytvoření základní populace kandidátních řešení.
2. Vyhodnocení kvality řešení v populaci na základě požadavků na řešení a výběr elitních jedinců s nejvyšší kvalitou (podmnožina kandidátních řešení, o dané velikosti, které nejlépe odpovídají požadavkům na řešení).
3. Pokud je v elitní množině kandidátní řešení, které plně odpovídá požadavkům, pak je toto řešení označeno jako konečné a algoritmus je ukončen. Jinak pokračuje bodem 4.
4. Provedení křížení mezi dvojicemi jedinců v elitní podmnožině, čímž je doplněna populace do původního počtu. Poté je také provedena mutace náhodně zvolených jedinců v nové populaci. Algoritmus pokračuje bodem 2.

Tento základní algoritmus bývá dále upravován tak, aby reprezentace chromozomů a genetické operátory co nejlépe vyhovovaly řešenému problému. Chromozom může být tvořen řetězcem bitů (binární GA), kde je například mutace realizována invertováním jednotlivých bitů, nebo vektorem reálných číselných hodnot (spojité GA).

Kapitola 4

Platforma OpenCL

Platforma OpenCL je heterogenní výpočetní platforma zaměřená na unifikaci výpočetních zdrojů. Nejrozšířenější využití nachází jako platforma umožňující využití grafických akcelerátorů pro obecné výpočty (GPGPU – General Purpose GPU). Ovšem implementace nejsou omezeny pouze na GPU, rozšířené jsou také implementace pro CPU a existují i experimentální implementace v FPGA. Platforma OpenCL je tvořena jednak samotnou knihovnou a stejnojmenným programovacím jazykem, který vychází z jazyka C99. Její využití je možné jak v případě akcelerace datově paralelních programovacích modelů, tak i úkolově paralelních. Avšak při akceleraci genetického algoritmu zde popisovaného řešení návrhu filtrů je využíván pouze datově paralelní model. To je způsobeno především povahou problému, ale také tím, že cílový hardware tvoří grafický akcelerátor, jehož doménou jsou spíše úlohy tohoto typu.

Tato kapitola je věnována letmému pohledu na knihovnu OpenCL, která je využívána pro akceleraci zde diskutovaného řešení návrhu digitálních filtrů. Pozornost je věnována především těm vlastnostem OpenCL, které nejvíce ovlivňují použité algoritmy a jejich výslednou implementaci. Ze stejných důvodů jsou v této kapitole zmíněny i některé vlastnosti cílového hardwaru.

4.1 Struktura platformy OpenCL

Architekturu platformy OpenCL je možné popsat několika modely, kde každý z nich zachycuje část jejích vlastností. V této podkapitole budou postupně tyto modely popsány, spolu s jejich vlivem na popisované řešení.

Základním pohledem je model platformy (nebo strukturální model). Ten se skládá z hosta (zde klasické CPU), ke kterému může být připojeno jedno nebo více OpenCL zařízení (OpenCL device). Zařízení se dále dělí na výpočetní jednotky (Compute units – CUs) a tyto jsou dále děleny na výkonné elementy (Processing elments – PEs). Tyto pak vykonávají samotný výpočet nad daty. Výkonné elementy, tvořící jednu výpočetní jednotku pak mohou vykonávat program jako SIMD jednotky (v daném čase zpracovávají všechny elementy stejnou instrukci programu), nebo SPMD, kde všechny jednotky vykonávají stejný program, avšak nemusí provádět stejnou instrukci. V obou případech jednotky mohou pracovat s různými daty.

Modelem popisujícím OpenCL z pohledu vykonávání programu je model exekuční, ten je rozdělen do dvou částí: jádra (kernels) a program hosta. Kód tvořící kernel je vykonáván na straně OpenCL zařízení. Program hosta je pak klasický program vykonávaný na CPU.

Kernely jsou odesílány ke zpracování na zařízení tak, že je specifikován prostor indexů (index space) a kernel. Na OpenCL zařízení je následně vytvořena instance kernelu (work-item nebo vlákno) pro každý bod tohoto prostoru. Všechny instance provádějí stejný kód, avšak díky větvení kódu mohou vykonávat jeho různé části. Samozřejmě mohou také pracovat s různými daty. Instance jsou dále sdružovány do skupin (work-groups). Vlákna tvořící jednu skupinu jsou prováděna paralelně pomocí výkonných elementů jedné výpočetní jednotky.

Posledním důležitým modelem je paměťový model, který definuje několik paměťových prostorů, z nichž k některým má přístup pouze kernel, k jiným pouze host a některé jsou společné. OpenCL definuje tyto prostory:

- **Globální (Global Memory)**: Do této paměti mají přístup všechny instance kernelu, mohou z ní jak číst, tak i do ní zapisovat. Host do této paměti zařízení přistupuje prostřednictvím paměťových objektů (buffers), které označují části této paměti, a to obvykle pomocí kopírování těchto oblastí do paměti hosta.
- **Konstantní (Constant Memory)**: Tento paměťový prostor má z pohledu hosta stejné vlastnosti jako globální. Instance kernelu však nemohou do této paměti zapisovat, ale pouze z ní číst.
- **Lokální (Local Memory)**: Jde o paměť sdílenou vlákny uvnitř jedné skupiny (work-group), ty ji mohou jak číst, tak do ní zapisovat. Pro hosta je však nedostupná.
- **Privátní (Private Memory)**: Je využívána jako paměť proměnných jednoho vlákna (work-item). Ostatní vlákna, ani host do ní nemají přístup.

Krom těchto prostorů je zde ještě paměť hosta, do které však nelze z kernelů přistupovat.

4.2 Cílový hardware

Jako cílový hardware byl zvolen grafický akcelerátor AMD Radeon HD 4870, tedy GPU RV770. Přestože je OpenCL multiplatformní a kernely jsou do značné míry hardwarově nezávislé, je třeba při jejich programování dbát na vlastnosti platformy, pro kterou jsou určeny. V opačném případě může dojít ke značné ztrátě výkonu a nevyužití plného potenciálu hardwaru. Z tohoto důvodu jsou v této kapitole stručně popsány některé podstatné rysy tohoto GPU a z nich plynoucí důsledky pro implementaci. Některé z nich jsou platné obecně pro současné GPU, jiné u modernějších GPU již nemusí platit.

Základem jádra je 800 aritmeticko-logických jednotek (ALU), zde nazývaných Stream Processing Unit (SPU). Ty jsou uspořádány do pětic, kde jen každá pátá jednotka je schopná provádět transcendentní operace (exponenciální, logaritmické a trigonometrické funkce). Každá z těchto pětic tvoří spolu s jednotkou předpovědi skoků a bankou registrů jeden jednoduchý VLIW procesor, který je označován pojmem Stream Processor (SP). Tyto procesory jsou dále spolu s dalšími jednotkami (jako vyrovnávací paměť, jednotka mapování textur a další) spojovány do bloků. Každý tento blok bývá označován jako SIMD blok a v zásadě se tedy jedná o VLIW SIMD procesor, kterých má jádro 10. V terminologii OpenCL jeden SIMD blok tvoří Compute Unit (CU) a jeden VLIW procesor tvoří Processing Element (PE).

Jednotky SPU jsou schopny provádět operaci spojeného násobení a sečtení (Fused Multiply Add – FMA), maximálně je tedy jádro teoreticky schopné provést 2*800 operací v plovoucí desetinné čárce (FLOP) v jednom taktu. Celkový teoretický výkon tedy při frekvenci

750 MHz dosahuje hodnoty 1200 GFLOPS (v jednoduché přesnosti). Teoretická paměťová propustnost je pak při 256 bitové paměťové sběrnici a efektivní frekvenci 3600 MHz 115.2 GB/s.

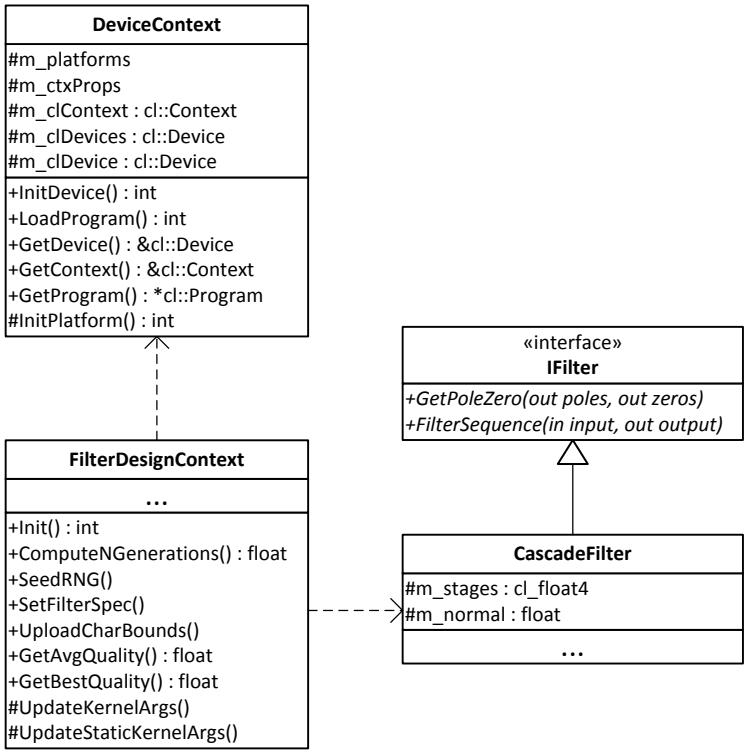
Dva předchozí odstavce obsahují pro programátora několik důležitých informací, které by měly být v implementaci zohledněny. Prvním důsledkem této architektury je, že každá instance kernelu je prováděna jedním VLIW procesorem o šířce 5 a je tedy vhodné, aby kód kernelu umožňoval jeho plné využití. To lze zajistit například jeho vektorizací, nebo jiným umožněním jeho paralelizace na úrovni instrukcí (ILP). Dále je třeba si uvědomit, že pouze jedna z pěti ALU je schopna vykonávat speciální operace. Druhou nevýhodou tohoto uspořádání (z pohledu programátora) je, že Compute Unit je procesor typu SIMD. Dojde-li tedy k rozdílnému větvení v rámci vláken prováděných jedním SIMD, je nutné provádění některých vláken pozastavit a jsou dokončeny až po zpracování zbylých, čímž dojde k nevyužití některých jednotek (tento jev bývá označován jako Thread divergence).

Dále je nutné zajistit optimální poměr aritmeticko-logických operací a přístupů do paměti. Z teoretických hodnot propustnosti paměťového systému a rychlosti výpočetních operací je zřejmé, že těchto výpočetních operací musí být pro plné využití výpočetních jednotek řádově více než operací paměťových. Značný vliv na datovou propustnost má také způsob čtení z paměti. V ideálním případě čtou všechna vlákna ze stejné adresy, nebo z adres jdoucích sekvenčně za sebou (tzv. coalesced memory access pattern).

Kapitola 5

Návrh rozhraní knihovny

V této kapitole je popsáno rozhraní knihovny pro návrh digitálních filtrů pomocí genetického algoritmu, akcelerovaného pomocí GPU. Základem knihovny je kontext zařízení (nejde přímo o OpenCL zařízení), který může být následně využíván jedním nebo několika kontexty pro návrh filtrů. Dále jsou součástí knihovny třídy pro specifikaci filtrů a třída pro reprezentaci digitálního filtru. Strukturu celé knihovny zachycuje v podobě UML diagramu obrázek 5.1.

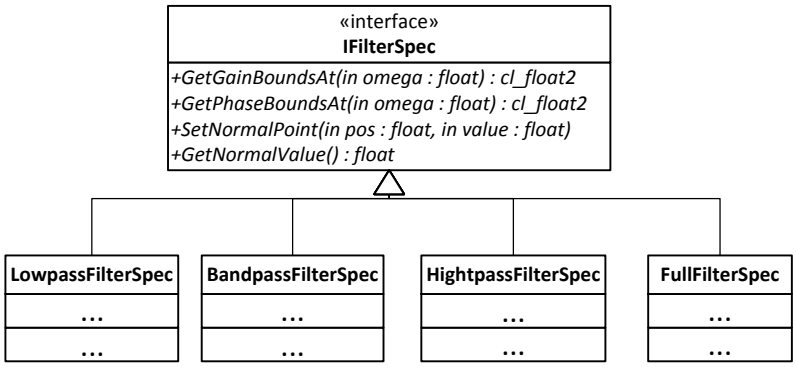


Obrázek 5.1: Zjednodušený třídní diagram knihovny

Úkolem třídy reprezentující kontext zařízení (zde nazvané *DeviceContext*) je spravovat OpenCL zařízení a kontext používaný knihovnou. Jejím dalším úkolem je správa OpenCL programů, což umožňuje vyhnout se jejich několikanásobnému překladu v případě, že je

stejný program používán několika kontexty pro návrh filtrů. Druhou základní třídou je třída nazvaná kontext návrhu filtru (*FilterDesignContext*). Tato zapouzdřuje samotný genetický algoritmus pro návrh filtrů (tedy jeho část vykonávanou na straně hosta – CPU). Navržený filtr je reprezentován třídou kaskádového filtru (*CascadeFilter*), která implementuje společné rozhraní pro případné další typy filtrů.

Méně důležitou část knihovny tvoří pomocné třídy pro zjednodušení zadávání specifikací filtrů. Jsou to třídy implementující rozhraní *IFilterSpec*, které umožňují zadávat specifikaci filtrů obvyklým způsobem pomocí intervalů specifikovaných násobky π. Tyto specializované třídy jsou k dispozici pro filtry typu dolní propust (třída *LowpassFilterSpec*), horní propust (*HighpassFilterSpec*) a pásmová propust (*BandpassFilterSpec*). Tuto část knihovny znázorňuje obrázek 5.2.



Obrázek 5.2: Zjednodušený diagram specifikačních tříd

Kapitola 6

Návrh filtrů pomocí genetických algoritmů

Jak bylo naznačeno v úvodu a ukázáno v pracích [8] a [1], je možné se na návrh digitálního filtru dívat jako na optimalizační problém. Přenosová funkce filtru pak tvoří model systému a parametry tohoto systému tvoří koeficienty filtru. Úkolem optimalizačního algoritmu (zde byl zvolen evoluční genetický algoritmus) je nalézt takové parametry (koeficienty), aby systém co nejlépe odpovídal zadaným požadavkům. Požadované vlastnosti systému zde tvoří zadané parametry filtru. Výhodou tohoto přístupu je značná volnost při zadávání parametrů filtru. Ve zde prezentovaném řešení se jedná o hledání určitého počtu koeficientů (odvozeného od specifikovaného řádu filtru), takových, aby charakteristika výsledného filtru co nejlépe odpovídala zadaným mezím. Složky charakteristiky jsou pak zadávány odděleně jako fázová (fázový posun filtru) a frekvenční (zisk filtru).

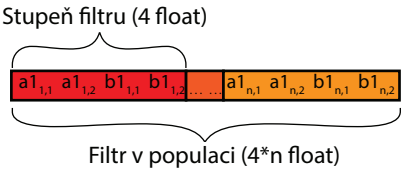
6.1 Genetický algoritmus

Jak již bylo naznačeno, k řešení problému byl zvolen spojitý evoluční GA, kde jedince v populaci představují jednotlivé kaskádové filtry a koeficienty jednotlivých stupňů tvoří geny těchto jedinců. Koeficienty filtrů jsou tedy spojité, při implementaci jsou jejich hodnoty reprezentovány typem float jazyka C++. Hlavní smyčka algoritmu je, oproti algoritmu uvedenému v kapitole 3, mírně modifikovaná. Hlavním rozdílem je, že smyčka začíná operátorem selekce a končí vyhodnocením kvality. Toto pootočení smyčky je způsobeno tím, že v první iteraci slouží operátor selekce k vygenerování výchozí populace filtrů.

6.2 Reprezentace filtrů

Základním úkolem, který je třeba vyřešit při použití genetického algoritmu k řešení nějakého problému, je volba vhodné reprezentace jedinců v populaci. Tato reprezentace musí být vhodná jak pro použití při implementaci genetických operátorů (aby nebylo nutné provádět zbytečné konverze), tak i pro použití při vyhodnocování cenové funkce (v tomto řešení jde o funkci určující kvalitu kandidátního řešení, jež má opačný význam). Z těchto důvodů se jeví jako vhodná kaskádová reprezentace filtru složená ze stupňů tvořených filtry druhého řádu, které mohou případně degradovat na filtry prvního řádu. Tyto filtry druhého řádu jsou dále reprezentovány jako čtveřice koeficientů, tvořených reálnými hodnotami. Takové uspořádání znázorňuje obrázek 6.1. Mimo důvodů spojených se stabilitou filtrů (viz. 2.2.4)

vedly k tomuto uspořádání i důvody implementační, především pak výše zmiňovaná šířka výpočetních jednotek cílového hardwaru (jeden stupeň filtru tvoří vektor reálných hodnot délky 4).



Obrázek 6.1: Reprezentace filtru v populaci

6.3 Genetické operátory a jejich implementace

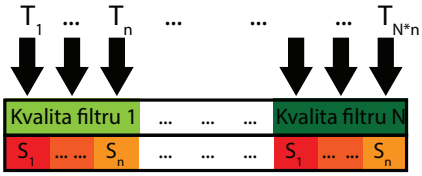
Genetické operátory, ačkoli zajišťují v genetickém algoritmu vždy stejné úlohy, je nutné navrhnout tak, aby jejich činnost odpovídala řešenému problému a reprezentaci každého jedince i celé populace. Zde se jedná především o modifikaci operátorů pro použití se spojitými GA. Další modifikace jsou pak důsledkem povahy problému a rysů cílové platformy. V této podkapitole je popsána činnost jednotlivých operátorů a některé implementační detaily. Z implementačního hlediska jsou operátory tvořeny procedurami prováděnými na straně GPU (OpenCL kernels) nad celou populací filtrů.

6.3.1 Operátor selekce

První operací, kterou je nutné provést před zahájením genetického algoritmu, je vytvoření prvotní populace. Zde se jedná o vygenerování náhodných filtrů daného řádu, které však všechny musí splňovat podmínku stability. Toto prvotní generování populace zajišťuje operátor selekce, který zároveň, v následujících iteracích algoritmu, nahrazuje špatné filtry v populaci novými náhodnými jedinci. Volba filtrů, které mají být nahrazeny, je zajištěna prahováním (nahrazeny jsou filtry s kvalitou nižší než zvolený práh). Prahová hodnota kvality je hledána na straně hosta (CPU) výběrem N-tého nejméně kvalitního filtru. Když je tato hodnota následně porovnána s kvalitou všech filtrů v populaci, dojde (za předpokladu, že v populaci není několik stejně kvalitních filtrů) k nahrazení právě N filtrů. Takto je zajištěn konstantní počet vyloučených a nahrazených filtrů, což zamezuje vyloučení příliš mnoha filtrů, které by mohlo nastat, pokud by byl jako práh použit průměr kvality filtrů v populaci. Samotné vyhledání hodnoty prahu je zajištěno seřazením populace a následným výběrem hodnoty na příslušné pozici.

Pro generování nového filtru byl zvolen přístup, kdy je nejdříve zvolena poloha pólu nebo nuly a tato je následně přepočítána na příslušné koeficienty filtru. Kromě této metody lze také generovat příslušné koeficienty filtrů přímo, avšak ukázalo se, že tato metoda nevykazuje vhodné rozložení výsledných pólů vytvořeného filtru. Jednotlivé metody generování filtrů jsou dále diskutovány v kapitole 6.4.

Jelikož jsou jednotlivé stupně z pohledu operátoru selekce nezávislé a jejich společnou vlastností je pouze kvalita celého filtru, lze tento operátor aplikovat paralelně na úrovni stupňů. Je pouze nutné, aby bylo zaručeno, že při prahování daného stupně bude použita kvalita filtru, kterému tento stupeň přísluší. Způsob paralelizace ukazuje obrázek 6.2, kde T_n jsou vlákna a $S_1 - S_n$ jednotlivé stupně filtrů.



Obrázek 6.2: Paralelní nahrazování filtrů

6.3.2 Operátor křížení

Operátor křížení umožňuje vznik kombinací jedinců v populaci nejlépe tak, aby došlo ke zvýraznění, nebo alespoň zachování jejich pozitivních vlastností. Častým řešením, používaným v případě binárních GA, je například prohození určitého počtu bitů ve vektorech tvořících vstupní dvojici jedinců. Toto by bylo možné realizovat v případě popisovaného řešení na úrovni jednotlivých stupňů (nikoli na úrovni koeficientů, protože tehdy není garantována stabilita výsledného filtru). Práce [8] však ukazuje lepší řešení, které spočívá ve využití lineární interpolace jednotlivých koeficientů filtru. Parametr interpolace α pak lze volit náhodně, nebo (pro zlepšení konvergence) v závislosti na kvalitě vstupních filtrů. Výpočet parametru interpolace a výpočet koeficientu filtru ukazují rovnice 6.1 a 6.2.

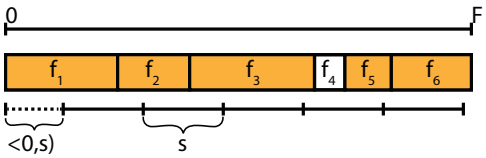
$$\alpha = 0.5(f^{(x)} - f^{(y)}) + 0.5 \tag{6.1}$$

Kde $f^{(x)}$ a $f^{(y)}$ je kvalita obou vstupních filtrů.

$$x_n^{(z)} = \alpha x_n^{(x)} + (1 - \alpha) x_n^{(y)} \tag{6.2}$$

Kde $x_n^{(z)}$ je nový koeficient a $x_n^{(x)}$, $x_n^{(y)}$ jsou koeficienty stupně obou rodičovských filtrů.

Pro výběr jedinců do množiny rodičů byl zvolen algoritmus stochastického vzorkování (Stochastic Universal Sampling – SUS) [2]. Ten pracuje tak, že uspořádá populaci do úsečky o délce součtu kvality všech jedinců (F). Každému jedinci je přiřazena část úsečky o délce odpovídající jeho kvalitě. Následně jsou na této úsečce voleny body s konstantním krokem $s = \frac{\sum f_i}{n}$, který je určen jako podíl celkové kvality populace a požadovaného počtu vybraných jedinců. První krok však nezačíná hodnotou nula, ale jeho začátek je posunut o náhodnou hodnotu (v uniformním rozložení) v intervalu $\langle 0, s)$. Jedinec je vybrán do množiny rodičovských jedinců, pokud začátek (nebo konec) některého kroku padne do intervalu určeného jeho kvalitou. Popsanou strukturu zachycuje obrázek 6.3. Tak je zajištěno, že pravděpodobnost výběru kvalitnějších jedinců je vyšší, avšak také je umožněn výběr méně kvalitních jedinců. Mimo vybrané jedince je do této množiny vždy zařazen nejlepší filtr v populaci.



Obrázek 6.3: Funkce algoritmu SUS

Zbytek populace je dále nahrazen (s volitelnou pravděpodobností) náhodnými kombinacemi jedinců z rodičovské množiny. Jednotlivé stupně nových jedinců jsou tvořeny nezávisle

(mohou být z dvojic stupňů různých filtrů), lineární interpolací dle kvality filtrů, jak je popsáno výše.

Algoritmus tvorby rodičovské množiny je implementován na hostu jako lineární průchod populace, kde jedinec je zařazen mezi rodiče, pokud pro něj platí, že hodnota s_Δ daná rovnicí 6.3 je menší než kvalita daného jedince a zároveň je větší než nula. Poté je možné paralelně pro každý stupeň zbylých jedinců vybrat dvojici z právě vytvořené množiny a nahradit ho jejich kombinací, toto je již realizováno na GPU.

$$s_\Delta = (\sum_{i=1}^{N} f_i - o) \bmod s \tag{6.3}$$

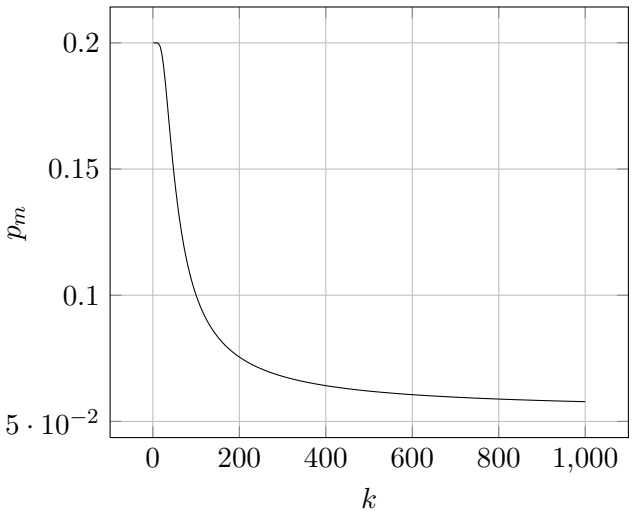
Kde f_i je kvalita jedince, N je index právě testovaného jedinece, o je posunutí prvního kroku a s je délka kroku.

6.3.3 Operátor mutace

Tento genetický operátor zajišťuje přísun nové informace do populace pomocí náhodných změn genů jedinců v populaci. V popisovaném řešení je však, krom vlastního operátoru mutace, zajišťován i operátorem selekce, kde dochází k nahrazování části populace novými filtry. Operátor mutace zvolený v prezentovaném řešení je založen na náhodné modifikaci náhodně zvolených stupňů jednotlivých filtrů. Modifikace celého stupně je nutná pro zajištění stability filtru po jeho mutaci. Dále je třeba poznamenat, že pravděpodobnost mutace se v průběhu genetického algoritmu snižuje dle rovnice 6.4, jejíž charakter ukazuje graf 6.4. Z mutace jsou vyloučeny filtry, které byly nahrazeny ve fázi selekce a také elitní filtry (jeden, nebo několik filtrů s nejvyšší kvalitou).

$$p_m(k) = \frac{b * e^{(k_0-k)/k}}{1 + e^{(k_0-k)/k}} \tag{6.4}$$

Kde b a k_0 určují počáteční hodnotu a strmost poklesu pravděpodobnosti mutace ($p_m(k)$) stupně filtru.

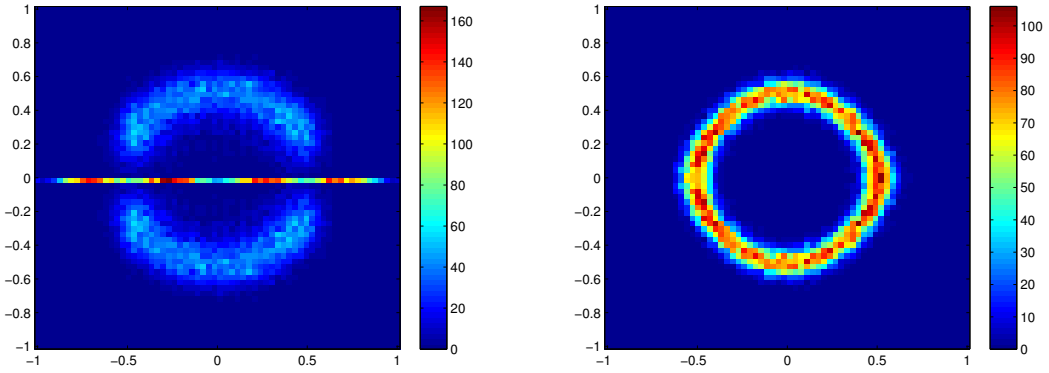


Obrázek 6.4: Průběh poklesu pravděpodobnosti mutace (funkce 6.4)

Samotnou modifikaci lze provést přímo nad koeficienty filtru přičtením náhodné odchylky ke koeficientům stupně filtru. Odchylky však musí být takové, aby si výsledný stupeň filtru zachoval stabilitu. Jinak řečeno, výsledný koeficient se musí nacházet uvnitř trojúhelníku představujícího oblast koeficientů stabilních filtrů (obrázek 2.5). Pro koeficienty a_1 a a_2 definující póly filtru lze toto zajistit definováním jejich maximálních odchylek do intervalů $a_{1\Delta} \in (-(a_2+1)-a_1, (a_2+1)-a_1)$ a $a_{2\Delta} \in (-1-a_2, 1-a_2)$ a následnou transformací koeficientu a_1 dle rovnice 6.5. Koeficienty definující nuly filtru však mohou nabývat libovolných hodnot (zde v intervalu $(-1,1)$), jelikož neovlivňují stabilitu filtru. Problémem v případě přímé modifikace koeficientů je však, stejně jako při generování filtrů, výsledné nepravidelné rozmístění pólů (případně nul) takto vytvořených filtrů. Výsledky dosažené touto metodou a výsledky dosažené metodou následující jsou znázorněny na obrázcích 6.5a a 6.5b.

$$a_{1n} = (a_1 + a_{1\Delta})\frac{a_2 + a_{2\Delta} + 1}{a_2 + 1} \tag{6.5}$$

Kde a_{1n} je nový koeficient, a_1 a a_2 jsou původní hodnoty koeficientů a $a_{1\Delta}$ ($a_{2\Delta}$) jsou náhodné odchylky koeficientů.



(a) Mutace v prostoru koeficientů

(b) Mutace v z-rovině

Obrázek 6.5: Hustota rozložení pólů po mutaci

Proto je zde využito výpočetně náročnějšího řešení, které spočívá v modifikaci filtru v jeho vyjádření pomocí nul a pólů v komplexní rovině. To je realizováno vypočtením kořenů polynomů druhého řádu, které tvoří jmenovatel a čitatel přenosové funkce modifikovaného stupně. Hodnoty reálné i imaginární složky kořenů tvořících póly a nuly filtru jsou pak použity jako střed náhodné veličiny s normálním rozložením. Následně jsou takto získané nové nuly a póly, dle rovnic 6.6 a 6.8, transformovány tak, aby ležely uvnitř jednotkové kružnice a následně převedeny, dle rovnic 6.9 zpět na koeficienty nového filtru.

$$a_{x,y} = \begin{cases} 1 & \text{pro } a_{x,y} > 1 \\ -1 & \text{pro } a_{x,y} < -1 \\ a_{x,y} & \text{jinak} \end{cases} \tag{6.6}$$

Kde $a_{x,y} = a_x + a_y i$ a jde o náhodně modifikovaný kořen jednoho z polynomů původního filtru.

6.4 Generování náhodných filtrů

Pro správnou činnost genetického algoritmu je nutné před jeho zahájením vygenerovat výchozí populaci náhodných řešení. Důležitou vlastností výchozí populace je její rozmanitost a co nejlepší pokrytí prostoru řešení. To umožnuje genetickému algoritmu hned od počátku identifikovat potenciálně zajímavé oblasti. V této podkapitole jsou proto popsány použité metody generování pseudonáhodných čísel a jejich použití pro generování náhodných filtrů. Vliv kvality generátoru na genetický algoritmus diskutuje například práce [4].

6.4.1 Generátory pseudonáhodných čísel

Jedním ze základních pilířů umožňujících reálnou použitelnost genetického algoritmu jsou generátory pseudonáhodných čísel. Jsou využívány jak pro řízení běhu algoritmu a zajištění funkce operátorů, tak při generování náhodných kandidátních řešení – náhodných filtrů.

Stejně jako ve většině případů i zde je základem celočíselný generátor s uniformním (rovnoměrným) rozložením. Zde použitý generátor [10] s periodou 2^{63} navíc obsahuje podporu pro rozdělení této periody mezi více vláken. Obsahuje také vektorovou implementaci, která je schopná generovat vektor nezávislých pseudonáhodných čísel o maximální délce 4. Nevýhodou je však pomalá inicializace generátoru, která je proto prováděna odděleně a stav generátoru v průběhu algoritmu je udržován v paměti grafické karty. Stav generátoru pak tvoří 64 bitů pro každý proud. Celkem tedy, v případě, že je využívána jeho vektorová verze, jde o 4*64 bitů pro každé vlákno (nebo v OpenCL terminologii work-item). Výstupem generátoru jsou 32 bitové hodnoty v intervalu $\langle 0, 2^{32} - 1 \rangle$, které jsou však pro použití v popisovaném algoritmu nejdříve normalizovány a následně transformovány do požadovaného intervalu.

Druhým použitým generátorem je generátor s normálním rozložením, který je využíván například operátorem mutace. Ten je založen na vektorové implementaci algoritmu Box and Muller [3]. Jeho výhodou je především to, že nejde o iterativní algoritmus, ale numerickou transformaci a také to, že pro vygenerování dvou nezávislých náhodných hodnot v normálním rozložení vyžaduje pouze dvě nezávislé hodnoty s uniformním rozložením.

Výpis 6.1: Generátor pseudonáhodných čísel v normálním rozložení

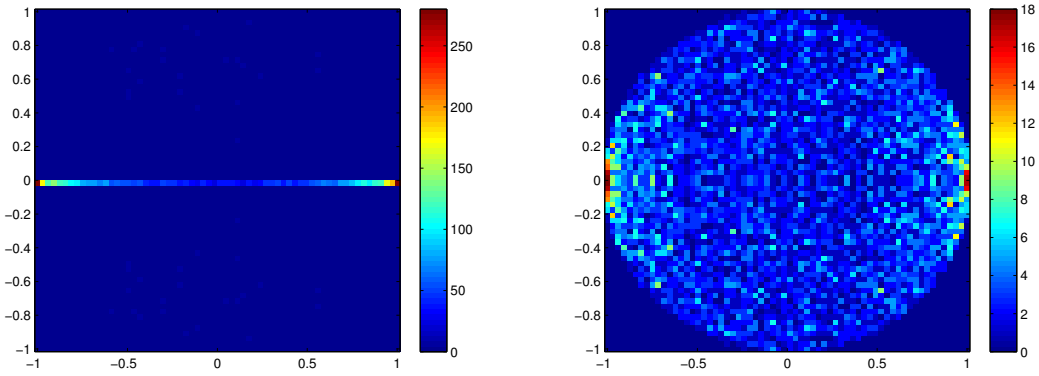
```
float4 RandNormalFloat4(float4 mi, float2 sigma,
    mwc64xvec4_state_t *s)
{
  float4 out;
  float4 randSeed = RandFloat4((float4)(.0f), (float4)(1.f), s);

  out = (sigma * sqrt((float2)(-2.0f) *
    log((float2)(1.0f) - randSeed.xz))).xxyy;
  out.xz = out.xz * sin(2.0f * M_PI_F * randSeed.yw);
  out.yw = out.yw * cos(2.0f * M_PI_F * randSeed.yw);

  return out + mi;
}
```

6.4.2 Generování náhodných filtrů

Generování náhodných filtrů je prováděno operátorem selekce hlavně pro vytvoření prvotní populace a následně pro nahrazení vyloučených filtrů při provádění selekce na základě jejich kvality. V této podkapitole jsou uvedeny dvě metody generování filtrů, z nichž první je triviální, avšak nevyhovuje požadavkům genetického algoritmu. Druhá metoda je založena na transformaci kořenů polynomu na koeficienty filtru a je vyhovující pro popisované řešení. Srovnání obou metod je zobrazeno histogramy 6.6a a 6.6b.



(a) Generování v prostoru koeficientů

(b) Generování v z-rovině

Obrázek 6.6: Hustota rozložení pólů při generování

Nejjednodušší možností, jak vytvořit náhodný filtr druhého řádu je náhodné vygenerování čtveřice reálných čísel (a_1, b_1, a_2, b_2) které tvoří koeficienty filtru. Takto vygenerovaný filtr však nemusí být stabilní, není garantované, že jeho koeficienty a_1, a_2 spadají do oblasti stabilních koeficientů (orázek 2.5). Proto je třeba omezit generované hodnoty koeficientů do intervalu $(-1, 1)$, tak, že bude platit $a_{1,2} \in (-1, 1)$, a následně transformovat a_1 dle hodnoty a_2, jak ukazuje rovnice 6.7.

$$a_1 = a_{1n}(a_2 + 1) \tag{6.7}$$

Kde a_{1n} je netransformovaný koeficient a_1.

Problém tohoto řešení je zřejmý z histogramu 6.6a, koncentrace pólů takto vytvořených filtrů je mnohonásobně vyšší na reálné ose. Pro pokrytí prostoru stabilních filtrů by bylo vhodnější, aby koncentrace jejich pólů (a nul) byla rovnoměrná uvnitř celé jednotkové kružnice.

Z tohoto důvodu prezentované řešení využívá druhý přístup, který je založen na náhodném vygenerování kořenů polynomu, tvořícího čitatele nebo jmenovatele přenosové funkce filtru (nul nebo pólů filtru) a jejich následné transformaci na koeficienty těchto polynomů (koeficienty $a_{1,2}$ a $b_{1,2}$ filtru). Kořeny musí ležet uvnitř jednotkové kružnice, což lze zajistit podobně jako v předchozí metodě omezením jejich hodnot jak na reálné, tak imaginární ose do intervalu $(-1, 1)$. Následně je opět nutné provést transformaci polohy v jedné či druhé ose, dle umístění na ose kolmé (rovnice 6.8). Za předpokladu, že platí $a_x = b_x \wedge a_y = -b_y$ lze odvodit transformaci 6.9, kterou je možno použít pro vypočtení koeficientů $a_{1,2}$ (obdobně

$b_{1,2}$) nového filtru. Takto vytvořené koeficienty pokrývají pouze část oblasti stabilních koeficientů (obrázek 2.5) avšak rozložení nul a pólů v oblasti jednotkové kružnice je optimálnější.

$$a_x = a_{xn}\sqrt{1 - a_y^2} \tag{6.8}$$

Kde a_{xn} je netransformovaná reálná složka polohy kořene polynomu.

$$\begin{aligned} a_1 &= 2a_y \\ a_2 &= a_y^2 + a_x^2 \end{aligned} \tag{6.9}$$

Kde a_y a a_x jsou imaginární a reálná složka polohy kořene polynomu, $a_{1,2}$ jsou koeficienty nového filtru.

6.5 Vyhodnocení kvality filtrů

Kvalita filtrů je vyhodnocována na základě odchylek jejich frekvenčních a fázových charakteristik od zadaných požadavků. Základem je tedy výpočet charakteristik každého filtru, který byl popsán v oddíle 2.2.2. Výpočet komplexní charakteristiky filtru tedy může být zapsán funkcí 2.9. V implementaci je pak využito toho, že hodnoty, kterých nabývá člen $e^{j\omega}$ závisí pouze na požadovaném počtu vzorků charakteristiky a lze je vypočítat dopředu a použít jako konstanty při vyhodnocování charakteristik všech filtrů. Jednotlivé vzorky charakteristik všech filtrů jsou vyhodnocovány paralelně na GPU.

Následuje samotný výpočet celkové chyby frekvenční charakteristiky, kterou tvoří součet kvadrátů odchylek od zadaného minima a maxima v každém vzorku charakteristiky (chybová funkce jednoho vzorku 6.10 a celková chyba 6.11). Ještě předtím je však třeba určit normalizační koeficient filtru K, který je, dle vzorce 6.12, určen jako převrácená hodnota zisku filtru ve zvoleném vzorku charakteristiky násobená požadovanou hodnotou zisku v tomto vzorku.

$$\varepsilon_f(n) = \begin{cases} |H(e^{j\omega})| - f_{max}(n) & \text{pro } |H(e^{j\omega})| > f_{max}(n) \\ |H(e^{j\omega})| - f_{min}(n) & \text{pro } |H(e^{j\omega})| < f_{min}(n) \\ 0 & \text{jinak} \end{cases} \tag{6.10}$$

$$E_f = \sum_{i=0}^{N} \varepsilon_f^2(i) \tag{6.11}$$

Kde $\omega = \pi(n/N)$, n je číslo vzorku a N je celkový počet vzorků.

$$K = \frac{G_0}{|H(e^{j\pi(n_0/N)})|} \tag{6.12}$$

Kde n_0 je vzorek vybraný uživatelem při zadávání specifikace filtru a G_0 je zisk požadovaný po normalizaci.

Druhou složku kvality filtru tvoří odchylka fázové charakteristiky, která je definována vzorcem 6.13. Opět se jedná o součet kvadrátů odchylek ve všech vzorcích charakteristiky. Protože v tomto případě jde o odchylky úhlů fázového posuvu, je nutné brát zřetel na jeho periodu 2π. Odchylky jsou proto počítány nejen vůči minimu a maximu v základní periodě, ale také vůči jejich obrazu v sousedních periodách ($+2\pi$ a -2π), rovnice 6.14. Za výslednou odchylku je pak považováno minimum z těchto tří hodnot.

$$\varepsilon_p(n,b) = \begin{cases} (arg(H(e^{j\omega})) + b) - p_{max}(n) & \text{pro } (arg(H(e^{j\omega})) + b) > p_{max}(n) \\ (arg(H(e^{j\omega})) + b) - p_{min}(n) & \text{pro } (arg(H(e^{j\omega})) + b) < p_{min}(n) \\ 0 & \text{jinak} \end{cases} \tag{6.13}$$

$$\varepsilon_{pmin}(i) = min(|\varepsilon_p(i,0)|, min(|\varepsilon_p(i,2\pi)|, |\varepsilon_p(i,-2\pi)|)) \tag{6.14}$$

$$E_p = \sum_{i=0}^{N} \varepsilon_{pmin}^2(i) \tag{6.15}$$

Kde opět platí, že $\omega = \pi(n/N)$, n je číslo vzorku a N je celkový počet vzorků.

Celková kvalita filtru je pak dána vzorcem 6.16 jako převrácená hodnota lineární kombinace frekvenční a fázové charakteristiky filtru. Parametr α je zadaný uživatelem a určuje velikost vlivu obou chyb na celkovou kvalitu filtru. Výpočet kvality probíhá paralelně pro všechny filtry.

$$f = \frac{1}{(1-\alpha)E_f + \alpha E_p} \tag{6.16}$$

Hodnota f se pohybuje v intervalu $\langle 0, 1\rangle$ a hodnoty 1 nabývá pro filtr, jehož charakteristiky se nacházejí uvnitř definovaných mezí. Dosáhne-li tedy kvalita některého filtru této hodnoty může být algoritmus ukončen - bylo nalezeno řešení plně odpovídající požadavkům.

Kapitola 7

Vlastnosti knihovny

V této kapitole jsou zhodnoceny některé vlastnosti výsledné implementace knihovny. V první části jde o shrnutí výsledků akcelerace algoritmu z pohledu rychlosti jeho běhu a využití výpočetních zdrojů, následující se zaměřuje na průběh návrhu základních typů filtrů.

Pro testování byl zvolen jako výchozí filtr s nekonečnou impulzní odezvou a řádem 4 (v knihovnou používané reprezentaci je tvořen dvěma stupni) typu dolní propust. Parametry tohoto a dalších testovaných filtrů zachycuje tabulka 7.1, kde pro filtry dolní i horní propust (LPF a HPF) platí:

- $|H(e^{j\omega})| \in \langle 1-\delta_1, 1\rangle$ pro
  - **LPF**: $\omega \in \langle 0, \omega_1\rangle$
  - **HPF**: $\omega \in \langle \omega_2, \pi\rangle$
- $|H(e^{j\omega})| \in \langle 0, \delta_2\rangle$ pro
  - **LPF**: $\omega \in \langle \omega_2, \pi\rangle$
  - **HPF**: $\omega \in \langle 0, \omega_1\rangle$

Pro filtr typu pásmová propust (BPF 8. řádu) platí:

- $|H(e^{j\omega})| \in \langle 0, \delta_2\rangle$ pro
  - **BPF**: $\omega \in \langle 0, \omega_1\rangle \vee \omega \in \langle \omega_4, \pi\rangle$
- $|H(e^{j\omega})| \in \langle 1-\delta_1, 1\rangle$ pro
  - **BPF**: $\omega \in \langle \omega_2, \omega_3\rangle$

Maximální pokles v propustném pásmu je -1.5dB ($\delta_1 = 0.1586$), maximální zisk v zádržném pásmu pak -30dB ($\delta_2 = 0.0316$). Hodnoty δ_1 a δ_2 jsou pro všechny filtry shodné. Mimo specifikované intervaly je zisk libovolný.

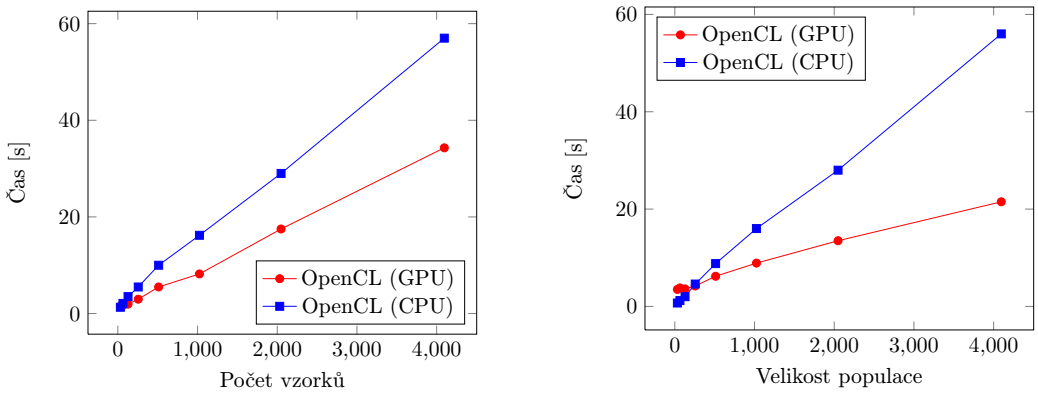
Type	ω_1	ω_2	ω_3	ω_4
LPF	0.2π	0.3π	-	-
HPF	0.7π	0.8π	-	-
BPF	0.2π	0.3π	0.7π	0.8π

Tabulka 7.1: Filtry použité k testování

7.1 Výkonnostní charakteristiky

Rychlost knihovny byla testována ve vztahu k základním parametrům GA, tedy k velikosti populace (obrázek 7.1b), počtu vzorků charakteristiky filtrů (7.1a) a počtu stupňů každého filtru (obrázek 7.2). Při změně jednoho z parametrů byly ostatním ponechány původní hodnoty, které jsou: 1024 generací, 1024 vzorků charakteristik a filtry o 2 stupních. K testům byl použit hardware popisovaný v oddíle 4.2 a procesor AMD FX-8350 s osmi hardwarovými vlákny a pracovní frekvencí 4GHz.

Samotné výsledky v zásadě odpovídají očekávání, na CPU dochází ve všech případech k téměř lineárnímu růstu doby běhu se zvyšujícím se řádem filtru, velikostí populace i délky jejich charakteristiky. Vliv způsobu paralelizace není tedy při akceleraci několika jádrovým CPU zřejmý.

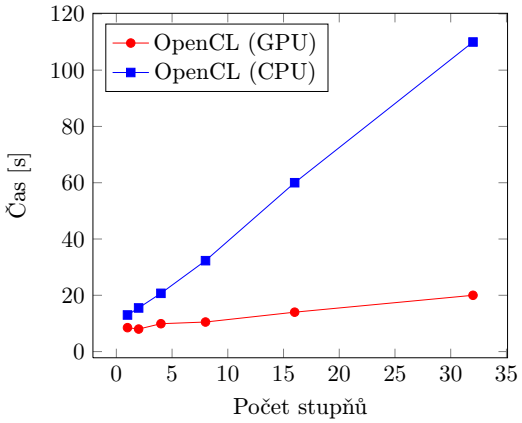


(a) Na délce charakteristiky

(b) Na počtu filtrů v populaci

Obrázek 7.1: Závislosti doby běhu algoritmu

Jiná situace ovšem nastává v případě běhu na GPU, kde, přestože zvýšení počtu vzorků charakteristiky má z části stejný význam jako zvýšení počtu filtrů v populaci, je zpomalování běhu v prvním případě prudší. To může být způsobeno tím, že vyhodnocování celkové chyby je paralelní pouze na úrovni filtrů a při jejich konstantním počtu a narůstání délky charakteristiky není GPU plně využíváno.

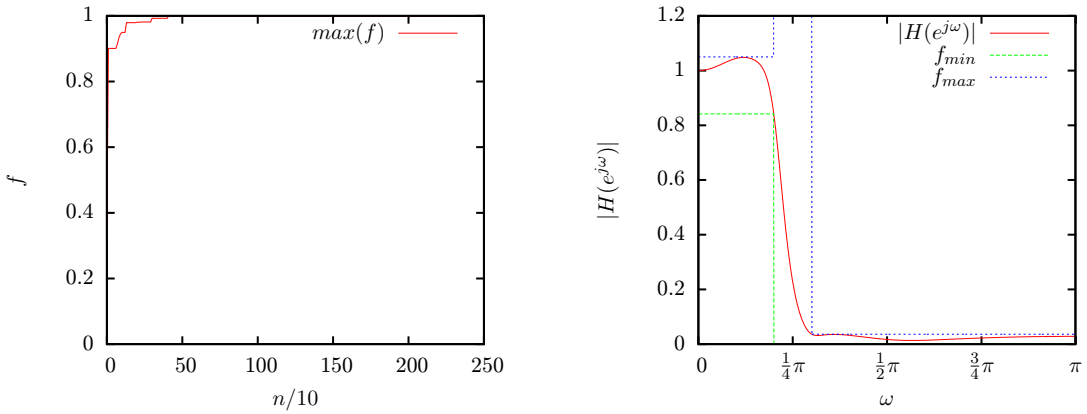


Obrázek 7.2: Závislost doby běhu algoritmu na počtu stupňů filtru

Za povšimnutí stojí také vliv zvyšování počtu stupňů filtrů (obrázek 7.2), který je na GPU až 5.5x nižší než na CPU. Tento jev je pravděpodobně způsoben tím, že počet stupňů neovlivňuje počet vzorků charakteristik, ani počet filtrů v populaci. A se stupni pracuje pouze část algoritmu implementovaná paralelně na GPU.

7.2 Konvergence algoritmu

Zde bude ukázán průběh návrhu základních filtrů s parametry popsanými v první části kapitoly 7. Parametry genetického algoritmu jsou voleny empiricky a nemusí tedy jít o jejich optimální hodnoty. Obrázky 7.3 a 7.4 ukazují průběh a výsledky návrhů filtrů 4. řádu typu dolní a horní propust. Rovnice 7.1 a 7.2 jsou pak jejich přenosové funkce.

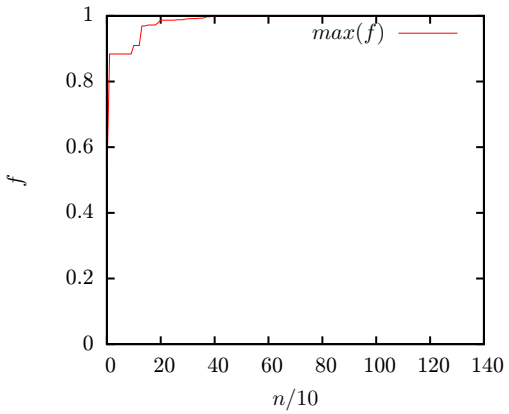


(a) Průběh návrhu filtru

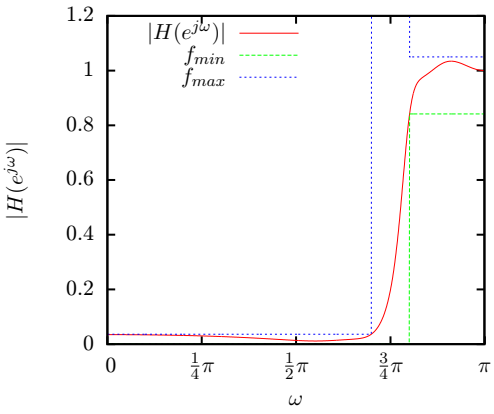
(b) Navržený filtr

Obrázek 7.3: Návrh filtru LPF 4. řádu

$$H_{LP}(z) = 0.061\frac{1 - 1.075z^{-1} + 0.859z^{-2}}{1 - 1.384z^{-1} + 0.761z^{-2}} \cdot \frac{1 + 0.237z^{-1} + 0.573z^{-2}}{1 - 1.215z^{-1} + 0.443z^{-2}} \tag{7.1}$$



(a) Průběh návrhu filtru

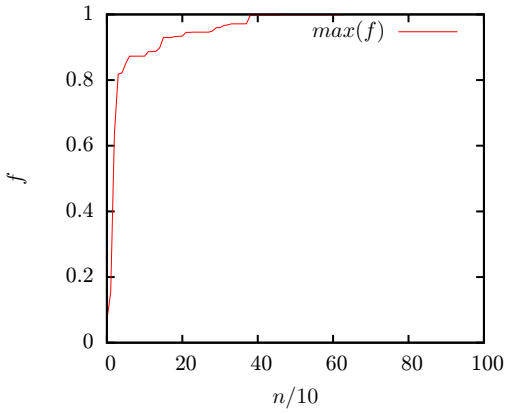


(b) Navržený filtr

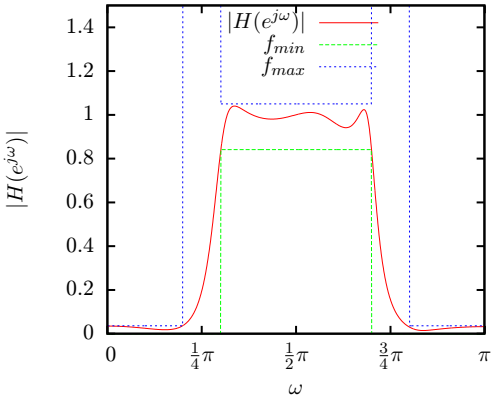
Obrázek 7.4: Návrh filtru HPF 4. řádu

$$H_{HP}(z) = 0.061\frac{1+0.277z^{-1}+0.633z^{-2}}{1+1.326z^{-1}+0.502z^{-2}}\cdot\frac{1+0.969z^{-1}+0.773z^{-2}}{1+1.428z^{-1}+0.803z^{-2}} \tag{7.2}$$

Jako test návrhu filtru vyššího řádu byl využit filtr typu pásmové propusti (řád 8), průběh a výsledek toho experimentu zachycuje obrázek 7.5. Rovnice 7.3 je pak jeho přenosovou funkcí.



(a) Průběh návrhu filtru



(b) Navržený filtr

Obrázek 7.5: Návrh filtru BPF 8. řádu

$$\begin{aligned} H_{BP}(z) = 0.16&\frac{1+1.068z^{-1}+0.457z^{-2}}{1+1.058z^{-1}+0.843z^{-2}}\cdot\frac{1-1.465z^{-1}+0.731z^{-2}}{1-0.666z^{-1}+0.397z^{-2}} \\ &\cdot\frac{1+1.577z^{-1}+0.842z^{-2}}{1+0.342z^{-1}+0.454z^{-2}}\cdot\frac{1-1.422z^{-1}+0.69z^{-2}}{1-1.037z^{-1}+0.781z^{-2}} \end{aligned} \tag{7.3}$$

Kapitola 8

Závěr

V této práci byl ukázán jeden z přístupů k návrhu digitálních filtrů s nekonečnou impulzní odezvou a kaskádovou strukturou. Prezentovaný přístup je založený na evolučním genetickém algoritmu, který je akcelerovaný pomocí GPU. Zrychlení algoritmu se projevuje především se zvyšujícím se řádem filtru, ale také při větších počtech jedinců v populaci, nebo větším počtu vzorků charakteristik filtrů. Celkově bylo dosaženo až více než 5 násobného zrychlení v porovnání s moderním vícejádrovým CPU.

Přesto naráží toto řešení na jisté limity a to především ze strany starších GPU bez kompletní podpory aktuálních verzí standardu OpenCL. V této oblasti se ukázalo, že nedostupnost atomických a synchronizačních operací poněkud znepříjemňuje návrh paralelního algoritmu. Tato omezení vyústila v nutnost rozdělení některých genetických operátorů na část zajišťující práci nad jedinci v populaci (na GPU) a část přípravnou (na CPU) jako je například seřazení kvalit jedinců v populaci, nebo výběr jedinců tvořících rodičovskou množinu při operaci křížení.

Z tohoto důvodu by jedním ze směrů dalšího rozvoje této práce mohl být přechod na novější verzi standardu OpenCL. Tím by mohlo dojít k přesunu dalších činností z CPU na akcelerátor. Jiným problémem řešení je, že umožňuje poze návrh filtrů sudého řádu (kvůli zvolené reprezentaci filtrů), což lze relativně snadno řešit modifikací struktury filtrů, avšak za cenu ztráty unifikace reprezentace jejich jednotlivých stupňů.

Sofistikovanější možností dalšího zlepšení vlastností algoritmu by mohlo být využití hybridního genetického algoritmu [5]. To by znamenalo, v případě kdy konvergence genetického algoritmu zpomalí, použít některý z optimalizačních algoritmů pro hledání lokálního minima, které mohou být v jeho okolí účinnější. Takto by mělo být možné dosáhnout zvýšení efektivity algoritmu, tedy omezení potřebného počtu generací.

Literatura

[1] Ahmad, S. U.: *Design of Digital Filters Using Genetic Algorithms*. Dizertační práce, University of Victoria, 2008.

[2] Baker, J. E.: Reducing Bias and Inefficiency in the Selection Algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, L. Erlbaum Associates Inc., 1987, s. 14–21.

[3] Brent, R. P.: Fast normal random number generators on vector processors. Technická zpráva, Australian National University, 1993, [Online], [cit. 2013-04-24].
URL http://maths-people.anu.edu.au/~brent/pd/rpb141tr.pdf

[4] Cantú-Paz, E.: On random numbers and the performance of genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann Publishers Inc., 2002, s. 311–318.

[5] Haupt, R. L.; Haupt, S. E.: *Practical Genetic Algorithms*. Wiley-Interscience, druhé vydání, 2004, ISBN 0-471-45565-2, 101-103 s.

[6] Infeachor, E. C.; Jervis, B. W.: *Digital Signal Processing: A Practical Approach*. Edinburg Gate: Pearson Education Limited, druhé vydání, 2002, ISBN 0-201-59619-9.

[7] Jan, J.: *Číslicová filtrace, analýza a restaurace signálů*. VUTTIUM, druhé vydání, 2002, ISBN 80-214-1558-4, 124-126 s.

[8] Jang, J.-D.; Kong, S. G.: Design of Optimal Digital IIR Filters using the Genetic Algorithm. *Journal of Fuzzy Logic and Intelligent Systems*, ročník 2, č. 2, červen 2002: s. 115–121.

[9] Proakis, J. G.; Manolakis, D. G.: *Digital Signal Processing: Principles, Algorithms, and Applications*. Prentice Hall, třetí vydání, 1996, ISBN 0-13-373762-4, 676-681 s.

[10] Thomas, D. B.: The MWC64X Random Number Generator. [Online], [cit. 2013-04-24].
URL http://cas.ee.ic.ac.uk/people/dt10/research/rngs-gpu-mwc64x.html

Seznam příloh

A Obsah CD **33**

B Manuál **34**

Příloha A

Obsah CD

- Zdrojové kódy knihovny a ukázkové aplikace v adresářích **/GAFilterDesign/src/**, **/GAFilterDesign/include/** a **/GAFilterDesign/kernels/**
- Projekty pro překlad na platformě Windows v adresáři **/GAFilterDesign/win/**
- Makefile pro překlad na platformě Linux v adresáři **/GAFilterDesign/linux/**
- Programová dokumentace vytvořená nástrojem Doxygen v adresáři **/GAFilterDesign/doc/**
- Tato práce ve formátu PDF v adresáři **/Thesis/**
- Zdrojové kódy této práce ve formátu LaTeXv adresáři **/Thesis/latex/**
- Přeložené spustitelné soubory ukázkové aplikace v adresáři **/GAFilterDesign/bin/**
- Plakát ve formátech EPS a PDF v adresáři **/Poster/**

Příloha B

Manuál

K překladu a spuštění ukázkové aplikace je potřebné běhové prostředí OpenCL (verze 1.0 a vyšší) na platformách Windows nebo Linux. Pro překlad je dále na platformě Windows možno využít Visual Studio ve verzích 2008 nebo 2010. V případě platformy Linux je vyžadován překladač GNU GCC (g++) a skriptovací prostředí make. Pro zobrazení výsledku a průběhu návrhu skriptem run_sample je na platformě Linux vyžadována přítomnost nástroje gnuplot.

Překlad a spuštění pod Windows

Na platformě Windows lze ukázkovou aplikaci a knihovnu přeložit standartním způsobem v prostředí Visual Studio 2008 nebo 2010 po otevření příslušného projektu. Ukázkovou aplikaci je následně možné spustit z adresáře **/GAFilterDesign/win/** příkazem **make_run.bat**. Výsledkem by mělo být navržení filtru typu dolní propust a zobrazení průběhu a výsledku návrhu. Ve stejném adresáři by měl být také vytvořen soubor s implementací filtru v jazyce C.

Překlad a spuštění pod Linuxem

Pod operačním systémem Linux je překlad a spuštění ukázkového návrhu filtru jednodušší, stačí pouze v adresáři **/GAFilterDesign/linux/** zadat příkaz **make run**. Pro spuštění návrhu s jinými parametry je možné použít příkaz **./run_sample.sh** se stejnými parametry jako v případě platformy Windows, které jsou popsány dále.

Testovací prostředí

- **Operační systém**: Windows 7 x64 (SP1), Linux OpenSUSE 12.2 x64
- **OpenCL prostředí**: AMD APP SDK 2.7 a 2.8
- **Ovladač GPU**: Catalyst 13.1 legacy a 13.4 (Linux - fglrx)
- **Hardware**: ATI Radeon HD 4870, AMD Radeon HD 7970 a AMD FX-8350
- **Překladač**: GNU GCC 4.7.1 (Linux)

Parametry ukázkové aplikace

- **--device-type cpu/gpu** - výběr cílového zařízení CPU nebo GPU
- **--population-size N** - počet jedinců v populaci
- **--char-len N** - počet vzorků charakteristiky filtru
- **--filters-order N** - řád filtru (sudé číslo $N \geq 2$)
- **--phase-weight R** - váha fázové složky kvality ($R \in \langle 0, 1 \rangle$)
- **--mutation-decay A B**
  - **A** - strmost poklesu mutace ($A > 0$)
  - **B** - počáteční pravděpodobnost mutace ($B \in \langle 0, 1 \rangle$)
- **--replace-part R** - část populace, která je v každé generaci nahrazena novými náhodnými jedinci ($R \in \langle 0, 1 \rangle$)
- **--elite-count N** - počet nejlepších jedinců, kteří nejsou ovlivňováni operátorem mutace ($N \in \{0, 1, ..., N_p - 1\}$, kde N_p je velikost populace)
- **--crossover-prob R** - pravděpodobnost křížení ($R \in \langle 0, 1 \rangle$)
- **--keep-part R** - část populace, která tvoří množinu rodičů pro křížení ($R \in \langle 0, 1 \rangle$)
- **--filter-type lpf/hpf/bpf/file_name ...** - typ navrhovaného filtru, nebo cesta k souboru s mezemi charakteristik
  - **lpf** $P_{end}\ PG_{min}\ PG_{max}\ S_{start}\ SG_{max}$
  - **hpf** $S_{end}\ SG_{max}\ P_{start}\ PG_{min}\ PG_{max}$
  - **bpf** $LS_{end}\ RS_{start}\ SG_{max}\ P_{start}\ P_{end}\ PG_{min}\ PG_{max}$

  Kde P_{end},P_{start} je konec resp. začátek propustného pásma. Intervaly jsou specifikovány v násobcích π a zisk v hodnotách $|H(e^{j\omega})|$. PG_{min},PG_{max} je zisk v propustném pásmu a SG_{max} zisk v zádržném pásmu. LS_{end} určuje konec zádržného pásma pásmové propusti (interval $\langle 0, LS_{end} \rangle$), RS_{start} pak začátek zádržného pásma na intervalu $\langle RS_{start}, 1 \rangle$.

  V případě specifikace souborem je na jeho prvním řádku uveden počet vzorků specifikace. Následující řádky pak obsahují limity charakteristik ve formátu:
  $G_{min}\ G_{max}\ P_{min}\ P_{max}$. Kde G jsou meze zisku a P jsou meze fázového posunu v intervalu $\langle -\pi, \pi \rangle$.
- **--gen-count N** - maximální počet generací ($N > 0$)
- **--normal-point A B** - normalizace zisku filtru v bodě $A \in \langle 0, 1 \rangle$ na hodnotu $B > 0$
- **--debug-file file_name** - specifikace souboru pro ukládání průběhu návrhu ve formátu: $max(f)\ avg(f)$, to jest kvalita nejlepšího filtru a průměrná kvalita celé populace
- **--seed N** - inicializační hodnota generátoru náhodných čísel

- **--debug-hz-file file_name** - specifikace souboru pro uložení charakteristik nejlepšího nalezeného filtru ve formátu:
  $|H(e^{j\omega})|$ G_{min} G_{max} $arg(H(e^{j\omega}))$ P_{min} P_{max} $real(H(e^{j\omega}))$ $imag(H(e^{j\omega}))$
- **--debug-poles-file file_name** - specifikace souboru pro uložení pólů a nul nejlepšího nalezeného filtru ve formátu: P_i $P_i i$ Z_i $Z_i i$
- **--output-filter-src file_name cascade/direct** - Uložení výsledného filtru v podobě implementace v jazyce C do souboru. Tato implementace může být kaskádová (původní navržený filtr), nebo přímá (ze stupňů filtru je vytvořen přímý filtr).