VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
Fakulta elektrotechniky a informatiky
Katedra informatiky

Bakalářská práce

Paralelní replaygian bez omezení
Parallel Reaplygain without limitation

Gorčica Libor

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne:

Rád bych na tomto místě poděkoval mému vedoucímu, Štěpánu Šrubařovi za pomoc při zpracování této práce.

Abstrakt

Náplní této bakalářské práce bylo především programování v jazyce C. Cílem bylo sjednotit zdrojové kódy třech aplikací, které implementují výpočet replaygain. Konkrétně se jedná o mp3gain, wavegain a vorbisgain. Po sjednocení těchto tří aplikací následovala implementace paralelního výpočtu a odstranění omezení na počet kanálů. Předpokladem pro řešení tohoto omezení bylo, že jednotlivé aplikace už mají vyřešeno dekódování pro více než dva kanály, což mp3gain nesplňoval. Výsledkem mého celoročního snažení je tedy aplikace, která podporuje všechny již zmíněné formáty a umožňuje paralelní výpočet. Z algoritmu pro výpočet replaygain jsem odstranil omezení na počet kanálů.

Klíčová slova

Replaygain, mp3gain, vorbisgain, wavegain.

Obsah

1 Úvod

V bakalářské práci jsem se zabýval výhradně programováním v jazyku C. Zpočátku jsem měl problémy se zorientovat ve velkém množství cizího kódu, který obsahoval tisíce řádků s minimální dokumentací. K programování jsem používal program gvim.

Začal jsem tedy analýzou jednotlivých programů a jejich následným spojováním. Využil jsem stabilní verze z repozitáře. Pro výpočet replaygain jsem použil soubor gain_analysis.c z programu vorbisgain. Wavegain a vorbisgain obsahovali některé velmi podobné hlavičkové soubory a funkce, proto nebyl velký problém je sjednotit. S mp3gain už to bylo horší. Programátor zvolil úplně jiný způsob implementace, a proto je jeho integrace o něco krkolomnější. Dále jsem se zabýval implementací paralelního výpočtu a nakonec odstraněním omezení na počet kanálů. Odstranění omezení na vzorkovací frekvence mi vedoucí doporučil nedělat kvůli vysoké náročnosti implementace. Vystačila by na další bakalářskou práci.

1.1 Popis struktury práce

V první části popisuji co je to replaygain, kdo jej vymyslel, k čemu se používá, jake audio formáty a přehrávače podporuje. Dále následuje stručný popis tagů s příklady použití. Dále velmi stručně popisuji používané způsoby komprese audio souborů.

V kapitole 2.5 Teorie zvuku a jeho vnímání, jak již název napovídá, popisuji základní principy šíření vnímání zvuku, výpočet intenzity, a také křivky stejných hlasitostí.

Bezprostředně následuje popis samotného algorimu replaygain. Aplikace filtrů, výpočet root mean square, teorie aplikace výpočtu na neomezený počet kanálů a nakonec pro úplnost algoritmus v pseudokódu.

V poslední kapitola teorie 2.9 Ukládání replaygain podrobně vysvětluji jaké hodnoty a jakým způseb se ukládají do hlaviček audio souborů.

Praktickou část jsem rozdělil na dvě kapitoly. Implementace a analýza výsledků. V implementaci popisuji, jak jsem postupoval při programování a své případné problémy. V analýze jsem provedl několik testů výkonosti svého programu.

2 Replaygain

2.1 Motivace

[1] Pokud máte doma velkou sbírku hudby různých stylů a interpretů, může se Vám stát, že některé skladby, nebo celá alba mají různou hlasitost než jiná. Pokud si například nastavíte náhodný výběr skladeb z Vaší sbírky, musíte neustále měnit hlasitost ve svém přehrávači, což je nepohodlné. To byl důvod proč se David Robinson rozhodl naimplementovat algoritmus, který hlasitost těchto skladeb vyrovná na podobnou úroveň, u které už téměř neuslyšíte rozdíl v hlasitosti. Jmenuje se replaygain a stal se prakticky standardem ve vyrovnávání hlasitosti skladeb a alb. Velkou výhodou a zároveň i komplikací je, že informace o změně hlasitosti se zapisuje do hlavičky souboru. To znamená, že se na Vaší skladbě jako takové nic fyzicky nezměnilo a informace je uložena jen v hlavičce souboru, kterou používá Váš přehrávač ke čtení doplňujících informací o skladbě. Např. ID3 tagy u mp3, kde se zapisuje jméno interpreta, název skladby atd. Komplikací tedy je, že Váš přehrávač musí o této informaci vědět, aby ji dokázal přečíst a také formát audio souboru musí alokovat alespoň 32 bitů, do kterých se tato informace zapisuje. Hlavním důvodem, proč zapisovat replay gain do hlavičky audio souboru, je možnost uložení více různých úrovní hlasitostí podle zvoleného módu výpočtu. Uživatel si pak může vybrat ten, který mu více vyhovuje. Další výhodou je, že můžete měnit hlasitost kolikrát chcete a nebude to mít žádný negativní vliv na kvalitu audio záznamu. Také je dobré zmínit, že díky ukládání replaygain do tagů je velice snadné udělat „krok zpět“ a vrátit tak vše do původní podoby. Prostě smažete informaci o replaygain z hlavičky, nebo zakážeme přehrávači číst replay gain značky.

2.2 Podporované formáty

Replaygain v této době podporuje celá řada používaných zvukových formátů.

Např:

- Mp3 – jeden z nejpoužívanějších formátů. Hlavně díky dobrému poměru velikost/kvalita.
- Vorbis – je projekt, který si klade za cíl vytvořit svobodný formát pro digitální multimédia.
- Wav –Vytvořen Microsoftem a IBM pro ukládání zvuku na PC. Není komprimovaný.
- Aac – Byl vyvinut jako následník formátu MP3 na středních až vyšších bitratech v rámci standardu MPEG4.
- Flac – je otevřený zvukový bezztrátový kodek.

2.2.1 Tagy jednotlivých formátů

Jak již bylo zmíněno, vypočtené replaygain hodnoty se ukládají do hlaviček audio souboru. Jednotlivé formáty však používají trochu jiný způsob ukládání. Formát mp3 využívá pro ukládání doplňujících informaci ID3v2 tagy. Flac a vorbis používají comment. Aac ukládá v mp4 meta souborech. Formát wav nemá žádné hlavičkové soubory, a proto je třeba upravit celou zvukovou stopu. Pozitivní fakt, který z toho plyne je, že pokud aplikujete replaygain na wav soubory, bude možné přehrát tyto soubory v libovolném přehrávači s požadovanou hlasitostí. Na druhou stranu nemá žádnou ze zmiňovaných výhod, takže si uživatel nemůže vybrat mezi album gainem a track gainem, protože fyzicky se hlasitost změní jen na jednu hodnotu. Další negativní fakt je nemožnost kroku zpět.

VorbisComment:

[4] Jako příklad struktury meta souboru jsem vybral vorbis comment.

Specifikace kanálů:

	Přední levý	Přední střední	Přední pravý	Boční levý	Boční pravý	Zadní levý	Zadní střední	Zadní pravý	LFE
1 kanál	✔								
2 kanály	✔		✔						
3 kanály	✔	✔	✔						
4 kanály	✔		✔			✔		✔	
5.1 sorround	✔	✔	✔			✔		✔	✔
6.1 sorround	✔	✔	✔	✔	✔		✔		✔
7.1 sorround	✔	✔	✔	✔	✔	✔		✔	✔

Specifikace hlavičky:

Do hlavičky mohou být zapisovány jednoduché řetězce, které popisuji audio soubor, kterému hlavička přísluší. Hlavička je v podstatě seznam obsahující vektory, ve kterých jsou uloženy požadované informace. Seznam může obsahovat až 2^{32} vektorů.

Formát vektorů:

Data se do vektorů ukládají nasledujícím způsobem

```
comment[0]="ARTIST=Bluetech"
comment[1]="TITLE=Leaving Winter Behind"
```

Comment tedy obsahuje index a na specifikovaný index ukládámé tzv. field name. Například artist a jeho hodnotu (field value) Bluetech. Field values používá standardní UTF-8 kódování pro snadnou reprezentaci různých jazyků. Field names nemusí být unikátní, to znamená že, pokud například skladbu nahráli tři různí producenti, mohou být uvedeni následujícím způsobem.

```
comment[0]="ARTIST=Dizzy Gillespie"
comment[1]="ARTIST=Sonny Rollins"
comment[2]="ARTIST=Sonny Stitt"
```

Field names:

- TITLE – název skladby.
- VERSION – pokud se nachází v jedné kolekci více stejných skladeb v různých verzích, může být toto pole použito pro jejich rozlišení.
- ALBUM – název alba.
- TRACKNUMBER – číslo skladby v albu.
- ARTIST – autor alba. V populární hudbě je zde většinou název skupiny nebo zpěváka. V klasické hudbě skladatel.
- PERFORMER – skladatel(é) kteří se podíleli na albu. V klasické hudbě to mohou být dirigent, orchestr, sólisté. V populární hudbě je zde obvykle stejná informace jako v ARTIST.
- COPYRIGHT – autorská práva.
- LICENSE – informace o licenci. Např. EFF Open Audio License.
- ORGANISATION – jméno organizace která produkovala skladbu.
- DESCRIPTION – krátký text s popisem skladby.
- GENRE – krátký text specifikující žánr.
- DATE – datum kdy byla skladba nahrána.
- CONTACT – kontakt na distributory nebo tvůrce skladby.
- ISRC – International Standard Recording Code.
- REPLAYGAIN_TRACK_GAIN – hodnota track (radio) gainu pro skladbu.
- REPLAYGAIN_TRACK_PEAK – hodnota špičky pro skladbu.
- REPLAYGAIN_ALBUM_GAIN – hodnota album (audiophile) gainu pro album.
- REPLAYGAIN_ALBUM_PEAK – hodnota špičky při album.

2.2.2 Možnosti komprese

Pro kompresi audio souborů se používají dva přístupy: ztrátova a bezeztrátová.

Bezeztrátová komprese

Jedná se o algoritmy, které umožňují přesnou zpětnou rekonstrukci komprimovaných dat. Bezeztrátová komprese se hodí všude tam, kde chceme zachovat kvalitu zvuku bez jakýchkoli ztrát.

Ztrátová komprese

Ztrátová komprese se naopak snaží zvukový záznam ořezat tak, aby zabíral jen zlomek původní kapacity, ale posluchač téměř nerozpoznal rozdíl. Například lidské ucho je schopno vnímat pouze, frekvence v určitém rozsahu. Typicky se uvádí 20 Hz až 20 kHz, takže uchovávat frekvence mimo tento rozsah je zbytečné. Obecný postup pro ztrátovou kompresi je jednoduchý. Nejprve se použije algoritmus, který oddělí důležitá data od nedůležitých, a následně se použije nějáký algoritmus pro bezeztrátovou kompresi.

Formáty používající ztrátovou kompresi:

mp3, vorbis, aac, wma.

2.3 Podporované přehrávače

Uvedu jen některé, které považuji za nejpoužívanější.

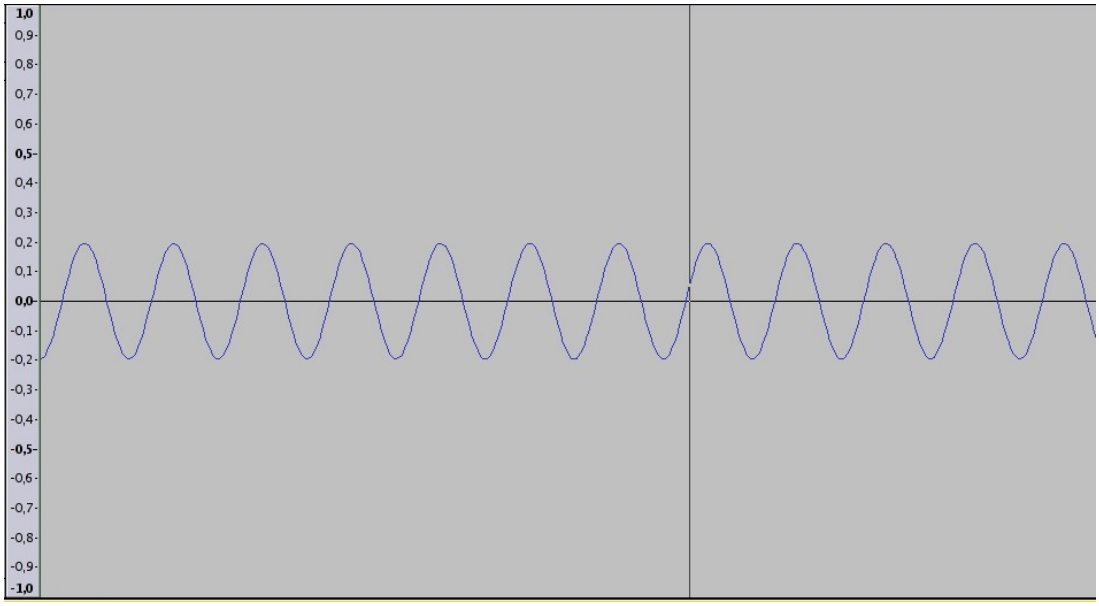
- Amarok – Přehrávač napsaný v qt
- Audacious – Podobný přehrávači winamp, systémy typu Unix
- foobar2000 – Windows
- Play – Mac OS X
- madplay – Linuxovský command-line přehrávač
- Mpd daemon – Unix-like systémy
- Mplayer – Přehrávač videa i hudby
- Rhytmbox – Přehrávač napsaný v GTK.
- Winamp – Známý přehrávač pro windows. Implementuje i vlastní replaygain výpočet, takže není třeba další program pro výpočet jako mp3gain, vorbisgain atd.
- XMMS2 – Další přehrávač podobný winampu, systémy typu Linux

2.4 Replaygain hodnoty

Po výpočtu replaygain mohou být v audio souboru uloženy dvě informace o hlasitosti. Track a album.

2.4.1 Track

Pokud nad svou hudební sbírkou sputíte tento výpočet, nastaví se hlasitost u každé písně zvlášť na referenční hodnotu 89 dB SPL definovaná v SMPTE RP 200 standardu. Budete mít tedy všechny skladby stejně hlasité tak, jako je tomu v rádiu pokud posloucháte stejnou stanici. Na obrázku 1 je sinus s frekvencí 1 kHz a hlasitostí na referenční úrovni 89 dB. Amplituda signálu dosahuje špičky viz. strana 22 hodnotě 0,2 v rozsahu 0 – 1.



Obrázek 1: sinus 1 kHz normalizovaný pomocí replaygain na 89 dB

2.4.2 Album

Problém s track módem pro výpočet spočívá v tom, že pokud máte album, kde zvukař již vyvážil hlasitost jednotlivých skladeb, například pomalejší skladby jsou tišší, rychlejší hlasitější, pak po aplikaci track gain tuto vyváženost ztratíte. Proto je tady album mód. Pokud tedy spustíte výpočet v tomto módu, bude se hlasitost vyrovnávat podle hodnot hlasitosti všech skladeb v albu, na kterém byl spuštěn. Problém album módu je, že stále pokud budete mít sbírku písní, kde budou jednotlivá alba mít vyrovnanou hlasitost a zvolíte například náhodné přehrávání, pak se stejně může stát, že skladba s heavy metalem nebo popem bude hrát o mnoho hlasitěji než klasika. To je důvod, proč je přidána ještě další možnost, a to nastavit míru korekce hlasitosti v dB ručně.

2.5 Teorie zvuku a jeho vnímání

2.5.1 Zvuk

[2] Zvuk je každé mechanické vlnění v látkovém prostředí, které je schopno vyvolat v lidském uchu sluchový vjem. Příčinou tohoto vlnění je existence vazebních sil mezi částicemi v látkovém prostředí. Tato vazba způsobí, že rozkmit jedné častice způsobený zdrojem zvuku ovlivňuje částice okolní. Tím se začne prostředím šířit vlna, která přenáší energii z jedné částice na druhou. Energie se přenáší pomocí podélného vlnění, kdy hmotné body kmitají ve směru šíření vlny. Dochází tedy ke ztlačování a rozpínání prostředí.

2.5.2 Hlasitost

Hlasitost je veličina, kterou nelze přesně definovat, protože každý jedinec má jinou citlivost sluchu. Pro objektivní zhodnocení intenzity zvuku byla zavedena intenzita zvuku *I*. Intenzita *I* je definována podílem výkonu *P* zvukového vlnění a plochy *S*, kterou vlnění prochází. Intenzita zvuku je přímo úměrná energii kmitání, které zvukové vlnění v daném bodě vzbuzuje. Tato energie pak závisí na druhé mocnině amplitudy výchylky a na druhé mocnině frekvence. Intenzitu zvuku tedy určují nejen změny tlaku vzduchu v daném místě, ale také výška tónu. Výška zvuku se určuje výhradně u tónů a je určena frekvencí. Lidské ucho je nejcitlivější na frekvence mezi 2 - 4 kHz. V souvislosti s intenzitou zvuku se vědci snažili najít nějakou míru, podle které bychom mohli měřit hlasitost zvuků ve vztahu k lidskému vnímání. Na základě experimentů byly nalezeny maximální a minimální hranice. Rozdíl mezi těmito dvěma hodnotami na číselné ose činí 10^{12}. Pro vyjádření hladiny intenzity je vhodné použít logaritmickou funkci, protože odpovídá našemu vnímání. Jako jednotka pro vyjádření hlasitosti byla zavedena jednotka Bel podle Alexandera Bella.

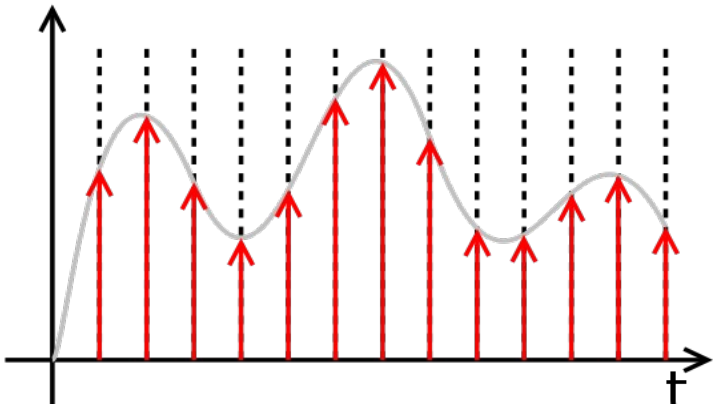
Má-li zvuk intenzitu *I*, pak v logaritmické stupnici lze vyjadřit hlasitost *L* jako $L = 10 \cdot \log \frac{I}{I_0}$, kde *I* je intenzita zvuku a I_0 je intenzita prahu slyšitelnosti.

Seznam pro srovnání akustického tlaku

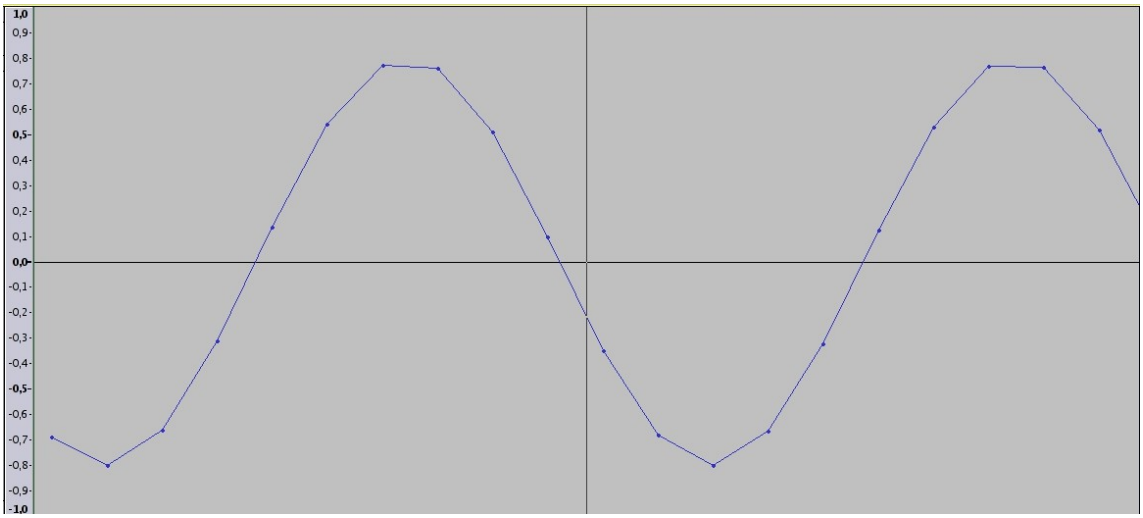
- Práh slyšitelnosti je 0 dB
- Šum ve studiu odpovídá asi 20 dB
- Tikot hodin odpovídá asi 30 dB
- Šepot z 10 cm odpovídá asi 50 dB
- Kytara z 40 cm odpovídá asi 60 dB
- Saxofon z 40 cm odpovídá asi 90 dB
- Brzdící vlak odpovídá asi 130 dB (práh bolesti)
- Vzlet tryskového letadla je více než 190 dB

2.5.3 Digitalizace

[5] Pro převod analogového signálu na digitální musíme provést digitalizaci. Místo dokonalého zkoumání analogového signálu vezmeme jen nezbytně nutné množství vzorků, v závislosti na požadované přesnosti, které budeme dále zpracovávat. Z obrázku 2 je patrné, že jsme ze spojitého signálu udělali nespojitý tvořený ze vzorků. Kvalita uloženého digitálního zvuku závisí na frekvenci vzorkování a množství informací uložených pro jeden vzorek. U vícekanáloveho zvuku je samozřejmě nutné vzorkovat každý kanál zvlášť. Velikost vzorku je obvykle 16 nebo 32 bitů. Pro digitalizaci zvuku se typicky používají hodnoty používane na CD, tedy kmitočet 44,1 kHz a 16 bitů na vzorek. Na obrázku 3 je ukázka vzorkovaného signálu s vzorkovací frekvencí 44,1 kHz.



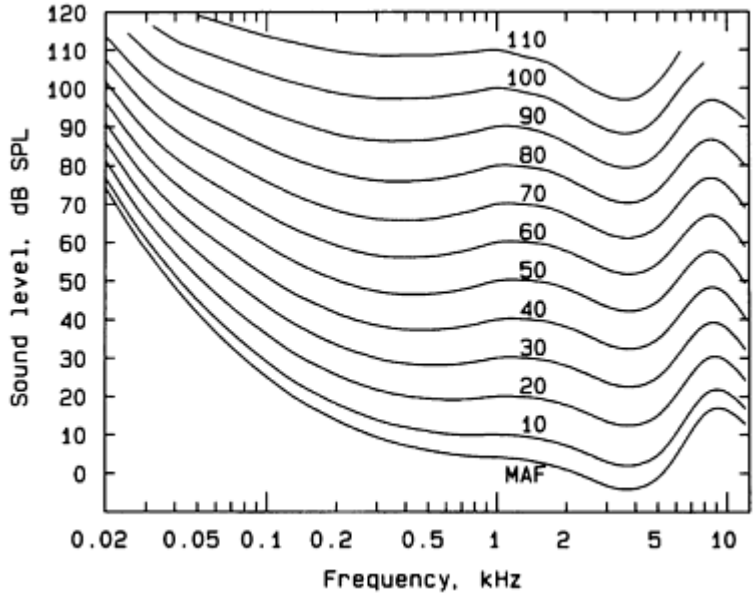
Obrázek 2: Vzorkování



Obrázek 3: vzorkovaný signál - 44100 Hz

2.6 Filtry

2.6.1 Vnímání zvuku



Obrázek 4: Munsonovy a Fletcherovy křivky

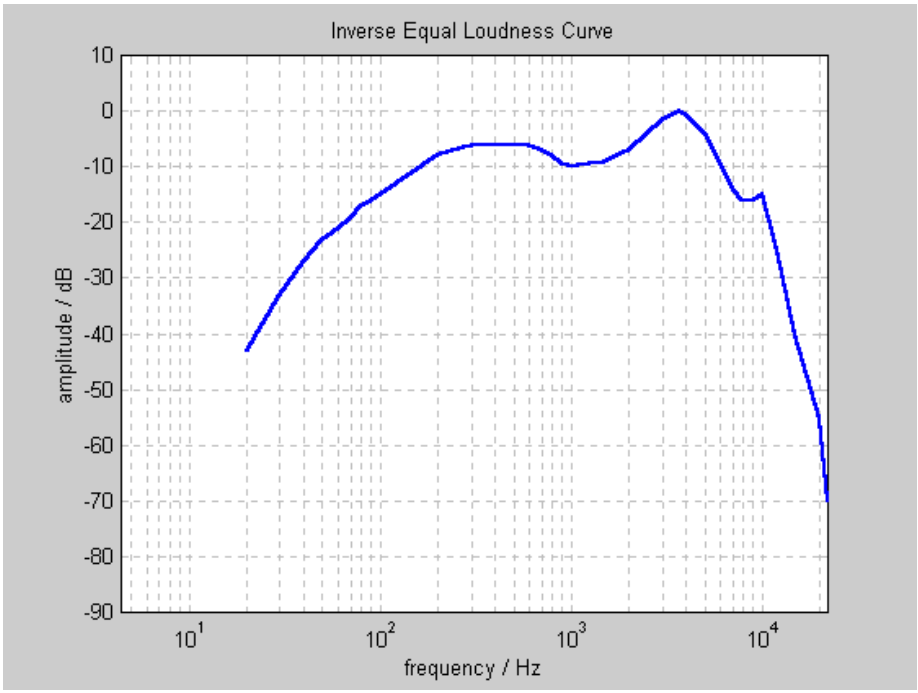
[3] Jak již bylo řečeno v teorii zvuku, hlasitost vnímaná lidským uchem se neodvíjí pouze od velikosti změn tlaku vzduchu, ale také od frekvence šířeného zvuku. V roce 1937 změřili Munson a Fletcher hlasitosti zvuků o různých frekvencích, tak jak je vnímá lidské ucho a vytvořili křivky, které se nazývají Munsonovy a Fletcherovy křivky 4. Pro testování byl definován referenční tón o frekvenci 1 kHz. Při testech se porovnávala hlasitost tohoto tónu s tóny o jiných frekvencích a výsledky se zaznamenávaly do grafu. Lidské ucho je schopno zaznamenat frekvence cca od 20 Hz – 20 kHz, kde horní hranice klesá věkem.

Na vertikální ose jsou hodnoty hlasitostí, které naměřily přístroje. Každá křivka je definována podle referenční hodnoty 1 kHz. Na horizontální ose jsou jednotlivé frekvence. Křivky popisují, jakým způsobem se mění vnímaná hlasitost zvuku v závislosti na frekvenci. Figurantovi byl vždy puštěn referenční tón a následně tón i jiné frekvenci. Figurant nastavoval hlasitost druhého tónu tak, aby došlo mezi oběma tóny ke shodě.

Příklad:

Pokud se podíváte na křivku označenou jako 60, znamená to, že byla definována pro tón o frekvenci 1 kHz a hlasitost 60 dB. Přesuňte se po této křivce na frekvenci 0,5 kHz. V tomto místě křivka dosahuje hlasitosti 55 dB. To znamená, že tón o frekvenci 0,5 kHz a hlasitosti 55 dB zní stejně hlasitě jako tón o frekvenci 1 kHz a hlasitosti 60 dB. Pokud se podíváte například na frekvence kolem 3 kHz, je z grafu patrné, že jsme na tyto frekvence nejvíce citliví.

Teď, když víme, jakým způsobem vnímáme hlasitost my, je třeba si uvědomit, že to neplatí obecně. Kdybychom zobrazili hlasitosti na počítači, dostali bychom vodorovné čáry, protože ve skutečnosti je hlasitost na všech frekvencích stejná. Abychom tedy mohli dělat nějáké výpočty vzhledem k našemu vnímání zvuku, budeme potřebovat nějakým způsobem simulovat tuto charakteristiku. K tomu se používají filtry. Není možné udělat jeden univerzální filtr pro všechny hlasitosti, protože pro každou hlasitost vypadá křivka trochu jinak. V replaygain se však používá pouze jeden filtr.



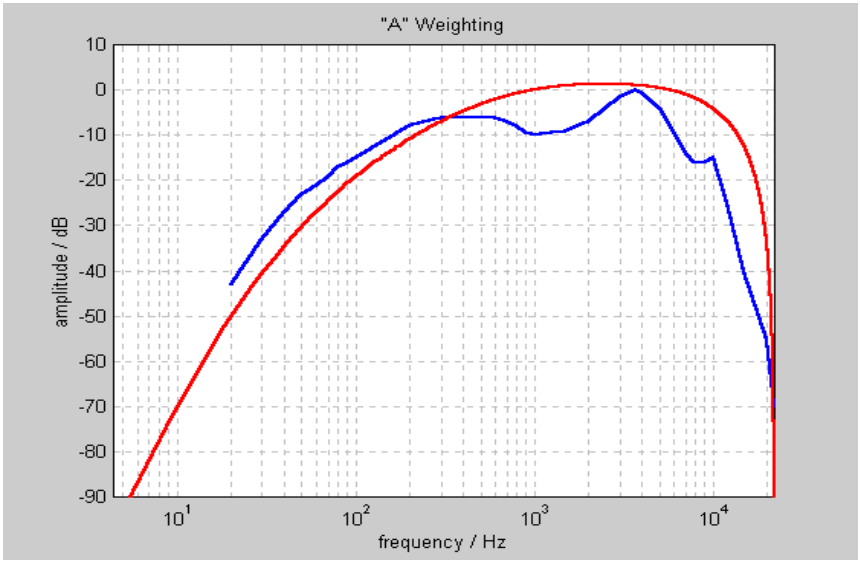
Obrázek 5: Průměrná inversní křivka pro implementaci filtru

Tato křivka 5 vznikla jako průměr uvedených křivek a je k nim inversní. Byla použita pro implementaci filtru. Při tvorbě filtru je třeba dosáhnout takové rychlosti, aby byla zachována dostatečná rychlost výpočtu. Výsledkem ideálního filtru vůči inversní křivce na obrázku je křivka, která ji kopíruje.

2.6.2 Druhy filtrů

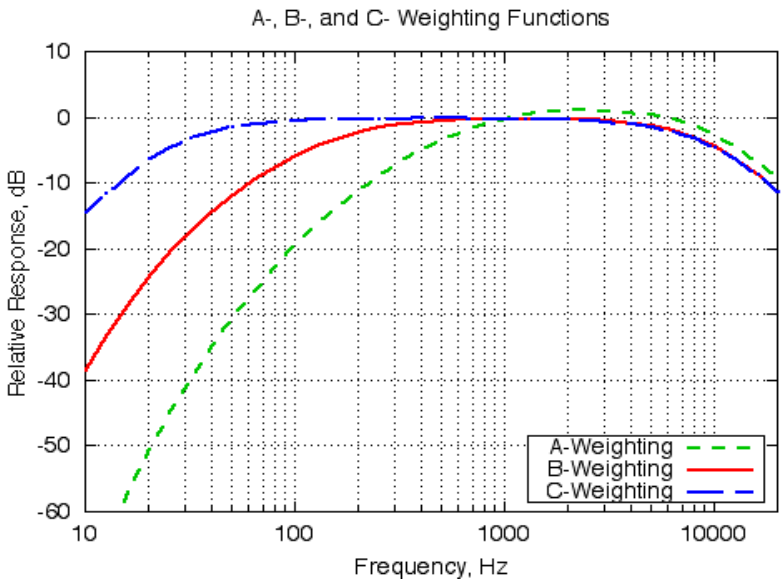
A, B, C vážené křivky

Zde uvedu několik standardních křivek, které se používají k simulaci charakteristiky vnímání zvuku.



Obrázek 6: A vážená křivka

A vážená křivka 6 je pouze aproximací a pro naše účely nedosahuje potřebné přesnosti.



Obrázek 7: A, B a C vážené křivky

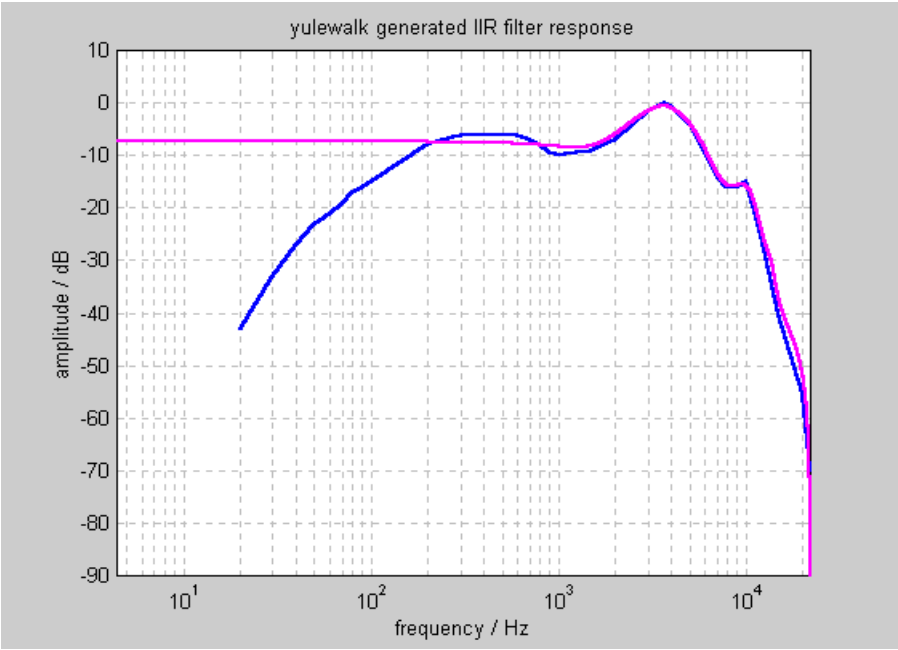
Žádná z těchto standardních křivek obrázek 7, ani jejich kombinace, nedosahuje dostatečné přesnosti.

IIR Filtry

Zde uvedu dva filtry, které byly použity pro výpočet replaygain.

Yulewalk IIR filter

Při vyšších frekvencích yulewalk aproximuje velmi přesně. Pro nižší frekvence ale už není jeho přesnost dostatečná, a proto bylo třeba naimplementovat ještě jeden filtr pro nižší frekvence.



Obrázek 8: Yulewalk filtr

Butterworth

Tento filtr na obrázku 9 je vhodný pro frekvence, při kterých yulewalk zaostával. Při použití obou filtrů už je aproximace dostatečná.

Obrázek 9: Zelená - butterworth, červená Výsledný filtr

2.7 Root mean square

Pokud aplikujeme v počítači filtry yulewalk a butterworth, změní se hodnoty hlasitosti přibližně tak, jak bychom je vnímali lidským uchem. Teď se tedy můžeme pustit do výpočtu hlasitosti záznamu.

Nejdříve je potřeba záznam rozsekat na části a na nich počítat jednotlivé hlasitosti. Části musí být tak malé, abychom při poslechu nebyli schopni zaznamenat změnu hlasitosti. David Robinson testoval úseky od 25 ms do 1 s. Nakonec vybral hodnotu 50 ms. Nyní tedy na tyto části aplikujeme vzorec pro výpočet root mean square zkráceně RMS.

$$rms = \sqrt{\frac{\sum_{i=1}^{N} a_i^2}{N}}$$

Pro všechny vzorky tedy spočítáme druhou mocninu a sumu těchto druhých mocnin podělíme počtem vzorků. Takto bychom postupovali, kdybychom měli zvukovou stopu jen s jedním kanálem. Pokud budeme mít víc než jeden kanál, je třeba spočítat sumu nezávisle pro všechny kanály a tu podělit počtem vzorků všech kanálů.

Slovně bychom tedy mohli popsat algoritmus následovně:

- pro každý kanál spočítej sumu druhých mocnin vzorků,
- spočítej sumu předchozích výsledků,
- výsledek vyděl počtem vzorků pro každý kanál,
- výslednou hlasitost ulož do pole,
- setřiď vzestupně prvky pole.

Nyní tedy máme hodnoty hlasitostí pro celou skladbu po 50 ms uloženy v poli. Potřebujeme vybrat jednu hodnotu, která bude reprezentovat danou skladbu. David Robinson provedl řadu testů a zjistil, že nejlepší hodnota subjektivního vnímání hlasitosti skladby je 95%. Nyní je potřeba rozlišit, jestli počítáme track gain nebo album gain. Pro track gain je to jednoduché. Podle referenční hodnoty 89 dB spočítáme, o kolik je potřeba změnit hlasitost skladby. Pro album gain si budeme postupně ukládat všechny vypočtené hodnoty rms do dalšího pole. Je třeba zmínit, že u paralelního zpracování je třeba, aby toto pole bylo mezi vlákny sdíleno. Po výpočtu hodnot pro všechny skladby se vybere opět hodnota 95% a podle ní se stejným způsobem určí album gain. Je to tedy úplně stejný způsob jako u track gain, jen bereme všechny skladby jako jednu.

2.8 Shrnutí

Popsali jsme si tedy vše, co je potřeba pro výpočet replaygain. Pro přehlednost uvádím algoritmus v pseudokódu.

```
sum=0;
foreach Track track in tracks
do
    foreach Block block in track
    do
        foreach Window_50ms window in block
        do
            foreach Channel ch in Channels
            do
                ch.filtered=filters(window.ch.samples);
            done

            foreach Channel ch in Channels
            do
                foreach Sample s in ch.filtered
                do
                    sum += s^2;
                done
            done
            rms = sqrt(sum / (window.num_samples * channels));
            dB = 10 * log10(rms+10^-10);
            arrayA.add(dB);
        done
    done

    foreach Item i in arrayA
    do
        arrayB.add(i);
    done
    arrayA.sort();
    track_gain = arrayA.get95_perVal();
    arrayA.clear();
done
arrayB.sort();
album_gain = arrayB.get95_perVal();
```

2.9 Ukládání replaygain

Pro ukládání informací o replaygain v hlavičce zvukového souboru se používají 3 hodnoty.

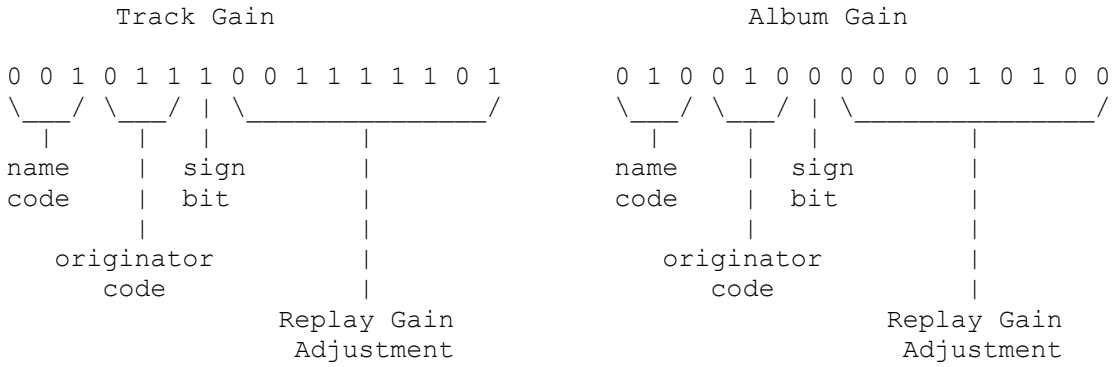
- Velikost amplitudy signálu
- Track gain (radio)
- Album gain (audiophile)

Rozsah

Vypočtená hodnota musí být v rozsahu od -51dB do +51 dB, který stanovil autor replaygain. Hodnoty, které jsou mimo tento rozsah by měly být buď přepočítány, jelikož se pravděpodobně jedná o omyl, nebo změněny tak, aby se vešly do rozsahu. Například se může stát, že pokud se hodnota bude počítat pro nějaký velmi tichý soubor, bude hlasitost změněna tak, aby odpovídala ideální hlasitosti. Ta může být i mimo tento rozsah. V praxi se ale považují za extrémy hodnoty od -23dB do +17dB a hodnoty od -18 dB do +2 dB jsou obvyklé.

Bitový formát

Zde uvádím názornou ukázku



Track gain byl vypočítán automaticky a výsledná hodnota je -12.5 dB.

U album gain uživatel přednastavil hodnotu +2.0 dB.

Name code:

- 000 – replay gain není nastaven
- 001 – použit track gain
- 010 – použit album gain
- jiná hodnota – rezervováno pro další rozšíření

Pokud v hlavičce zvukového souboru je rezervováno místo, ale replaygain nebyl použit, mohou být všechny hodnoty 0. Pokud je name code nastaven na 000, pak jej přehrávače ignorují. Pokud je name code nastaven na jinou hodnotu než 000, 001, 010, pak jej přehrávače ignorují.

Originator code:

- 000 - Replay gain nespecifikován
- 001 - Replay Gain přednastaven výrobcem nebo producentem
- 010 - Replay Gain nastaven uživatelem
- 011 - Replay Gain vypočítán
- jiná hodnota – rezervováno pro další rozšíření

Pokud spustíme výpočet replay gain a name code bude správně nastaven, ale originátor bude nastaven na 000, pak bude originátor ignorován. Pokud je originátor nastaven na nějákou hodnotu, kterou přehrávač nezná, pak bude originátor ignorován a bude proveden výpočet.

Sign bit:

Znaménkový bit. Pro 0 kladné hodnoty. Pro 1 záporné.

Replaygain adjustment:

Vypočtená nebo nastavená hodnota úrovně hlasitosti je reprezentována devíti bity.

Hodnota -3.1dB odpovídá 000011111.

Výchozí hodnoty:

0000000000000000 je použita, pokud zatím nebyl replaygain použit.

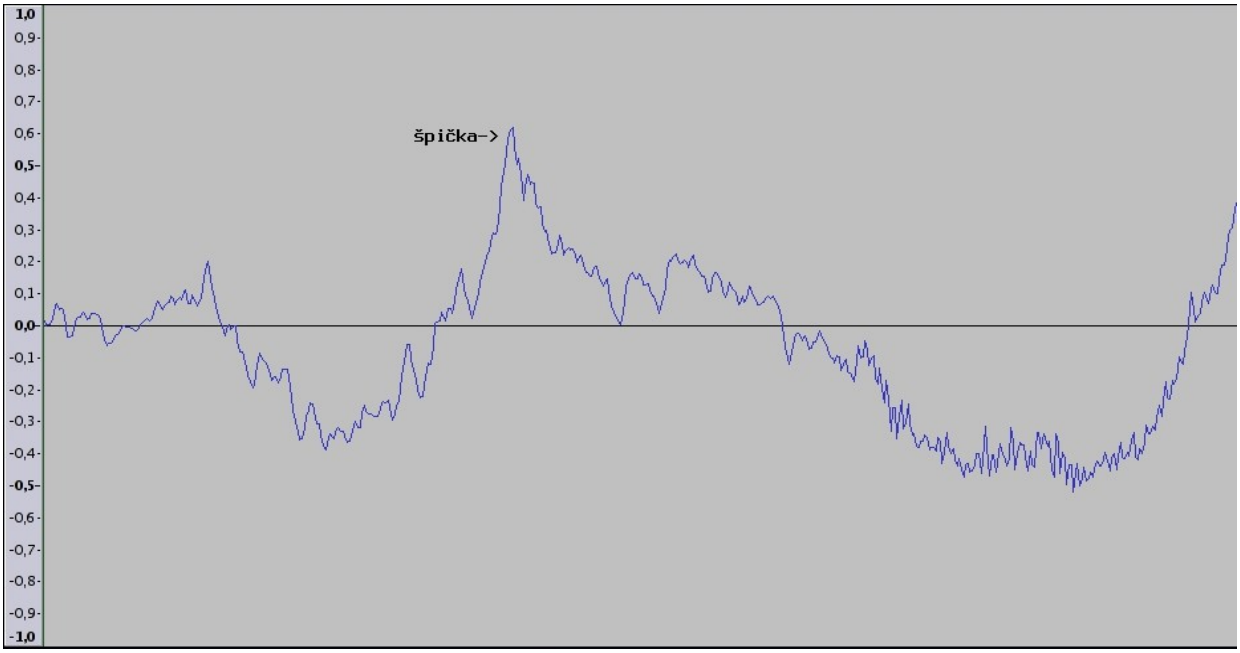
xxxyyy0000000000, kde xxx je name code a yyy je originator code, je validní hodnota, kde vypočtená hodnota regulace hlasitosti je 0dB. To znamená, že audio soubor má ideální hlasitost 89dB.

Illegal Values:

Hodnoty jako xxxyyy1000000000 nejsou validní, protože nelze mít zápornou nulu.

Špička

Špička (peak amplitude) je maximální velikost amplitudy (absolutní hodnota), kterou křivka dosáhla. Na obrázku je vidět špička, která má hodnotu 0,6 z intervalu 0 - 1. V tomto místě je tedy zvuková stopa nejhlasitější. Pro výpočet špičky se musí dekódovat celá skladba, a proto je výhodné si tuto hodnotu uložit v hlavičce audio souboru. Vypočtená hodnota může být použita pro detekci ořezu.



Obrázek 10: špička

Datový formát:

Maximální hodnota špičky musí být uložena jako celé neznaménkové 32-bitové číslo.

Nekomprimovaný soubor:

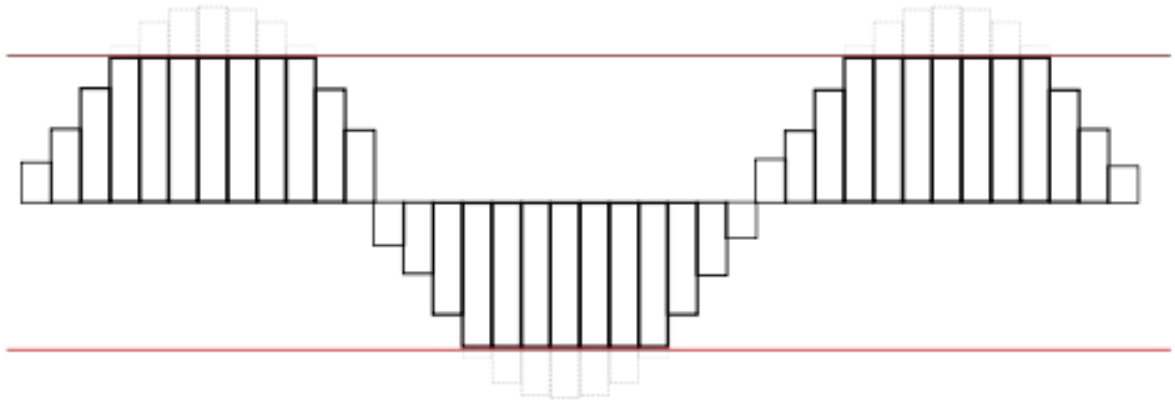
Stačí uložit absolutní špičky vzorku pro každý soubor.

Komprimované soubory:

Komprimovaný soubor musí být před analýzou dekódován. Někdy se může stát, že hodnota špičky před výpočtem replaygain je vyšší než maximální hodnota rozsahu, takže by došlo k ořezání. Zpravidla se ale po výpočtu replaygain, tedy zeslabení záznamu, vejdou do rozsahu.

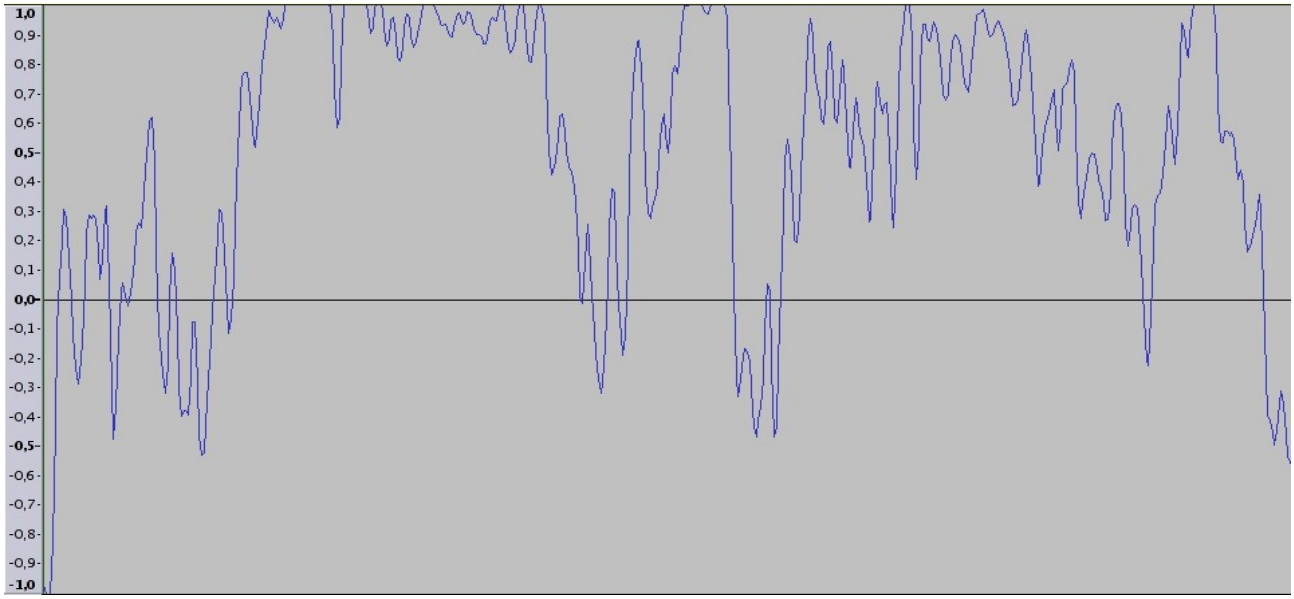
Ořez

[6] Zvukový výstup je vždy v nějákém intervalu (dnes typicky -1 – 1). Pokud audio signál tento interval překročí, zvuková karta běžně provede ořez viz. obrázek 11 (clipping) na maximum intervalu, čimž jej ovšem zkreslí. Řešením je buďto ponechat signál ořezaný, nebo celou skladbu zeslabit tak, aby se vešla do intervalu. Proto se ukládá špičková amplituda. Skladbu zeslabíme o rozdíl mezi špičkou a maximem intervalu.



Obrázek 11: Ořez

Na obrázku 12 je vidět jak v místě překročení intervalu 0 – 1 přestane být signál spojitý. V tomto místě došlo ke zkreslení, tedy ke ztlumení signálu na maximální úroveň.



Obrázek 12: Ukázka ořezu zvukového signálu

3 Vlákna

Pro paralelní zpracování funkcí v rámci jednoho procesu se používají vlákna. Ač se může zdát, že vlákna jsou velmi podobná procesům, protože jsou zpracovávána pseudoparalelně stejně jako procesy, jsou mezi nimi podstatné rozdíly. Pro každý proces si operační systém vede tabulku procesů s jedním záznamem pro každý proces. Tento záznam obsahuje mimo jiné informace o mapované paměti a otevřených souborech. Při přepínání procesů ve víceúlohovém systému je nutné vždy přemapovat paměť, což je časově náročná operace. Vlákna však běží v rámci procesu který jej spustil, to tedy znamená že při jejich přepínání se nemusí přemapovávat paměť, protože mezi sebou sdílejí paměť přidělenou procesu. Vlákna sdílejí většinu jeho prostředků. Mimo paměť také sdílejí otevřené soubory neboli file descriptory a signály. Skutečnost, že vlákna sdílejí stejnou paměť má jistě řadu výhod, ale také to přináší problemy, se kterými je třeba se vypořádat. Například pokud dvě vlákna zapisují do stejné proměnné, budou si ji navzájem přepisovat, což je většinou nežádoucí. Předpokládejme paralelní zpracování nasledujícího kódu dvěmi vlákny. Tento kód vychází z problému, který jsem řešil ve svém programu.

```
int process_files(FILE file_list)
{
      #pragma omp parallel
      {
            FILE file_t;

            while(file_list!=NULL)
            {
                  file_t=file;
                  if(file!=NULL)
                        file_list=file_list->next_file;
                  .
                  .
                  .
            }
      }
      return 0;
}
```

Pragma omp paralel (kapitola 3.1) označuje místo rozvláknění. Proměnná file obsahuje soubory a je sdílená oběma vlákny. Každé vlákno má svou proměnnou file_t do které uloží soubor z file, se kterým bude pracovat a následně uloží do proměnné file následující soubor. Předpokládejme, že nastane tato situace: vlákno A si uloží do proměnne file_t aktuální soubor z file, ale ještě dříve než stihne uložit do file následující soubor, uloží si vlákno B do file_t aktuální soubor. Obě vlákna teď pracují se stejným souborem, což samozřejmě nechceme. Došlo k tzv. souběhu a k jeho řešení je potřeba použít kritickou sekci. V kritické sekci může být v danný okamžik pouze jedno vlákno, což vyřeší předchozí problém.

```
int process_files(FILE file_list)
{
      #pragma omp parallel
      {
            FILE file_t;

            while(file_list!=NULL)
            {
                  #pragma omp critical
                  {
                        file_t=file;
                        if(file!=NULL)
                              file_list=file_list->next_file;
                        .
                        .
                        .
                  }
            }
      }
      return 0;
}
```

3.1 OpenMP

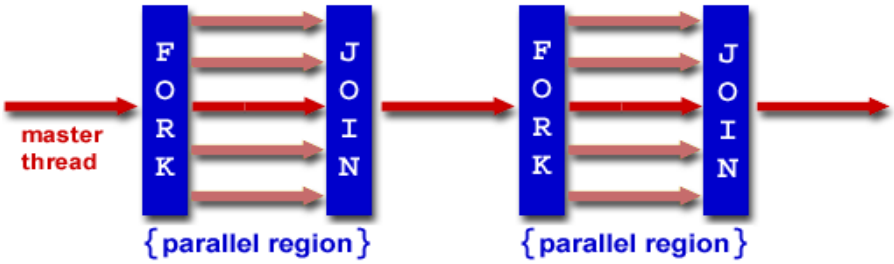
[7] OpenMP je rozhraní pro pro programování aplikací (API), které poskytuje relativně jednoduchý model pro programování paralelních aplikací. Proto jsem se jej také rozhodl použít ve svém programu místo POSIX threads (pthreads). API podporuje C/C++ a Fortran na UNIX a Windows NT systémech.

Obsahuje 3 základní API komponenty

- Direktivy překladače,
- Rozhraní definovaná v modulu omp_lib
- Proměnné prostředí

Programovací model:

OpenMp používa fork-join model viz. obrázek 13 paralelního zpracování. Každý OpenMP program začíná jako jeden proces označovaný jako hlavní vlákno (master thread). Hlavní vlákno provádí sekvenčně kód, dokud nenarazí na první paralelní sekci, která se označuje direktivou paralel. V tomto místě nastane větvení (fork) a hlavní vlákno vytvoří definovanou skupinu paralelních vláken. Sekce, která se provádí paralelně, je uzavřena v bloku. Na konci tohoto bloku dojde ke sjednocení vláken (join) a dále pokračuje opět hlavní vlákno sekvenčním zpracováváním. Programátor může za sebe řetězit libovolný počet takovýchto sekcí.



Obrázek 13: Fork-join model

Použití OpenMp:

Pro vytvoření skupiny vláken použijeme direktivu #pragma.

```
#pragma omp parallel
{
...
}
```

Pro specifikaci počtu vláken:

```
#pragma omp parallel num_threads(počet vláken)
{
...
}
```

nebo nastavíme proměnnou prostředí `export OMP_NUM_THREADS=4`.

Pro definici kritické sekce je to opět velice jednoduché:

```
#pragma omp parallel
{
      ...
      #pragma omp critical
      {
            ...
      }
      ...
}
```

4 Implementace

4.1 Sjednocování

V první fázi jsem musel provést podrobnou analýzu kódu jednotlivých aplikací. Na začátku jsem vybral jeden gain_analysis.c soubor, který obsahuje replaygain výpočet. Rozhodl jsem se pro vorbisgain implementaci. Následně jsem zkopíroval jednotlivé aplikace do jedné společné složky, kterou jsem pojmenoval allgain. Odstranil jsem funkce main, udělal jednu společnou a umístil jsem ji do souboru allgain.c. Vytvořil jsem Makefile a pokusil jsem se aplikaci přeložit. Při překladu jsem zjistil, že wavegain a vorbisgain mají několik společných funkcí. Nejspíš jeden vycházel z druhého. Mým dalším cílem tedy byla analýza těchto funkcí, abych zjistil kolik mají společného. Pokud jsem narazil na funkce nebo celé soubory, které si byly velice podobné, upravil jsem je tak, aby vyhovovaly potřebám obou programů, a umístil je do adresáře allgain. V ./allgain tedy najdete soubory, které jsou společné. Po tomto kroku už se mi podařilo vše přeložit a spouštět jednotlivé funkce. Každý program používal rozdílné parametry, čili můj další krok směřoval ke sjednocení těchto parametrů. Vybral jsem tedy všechny nezbytné pro výpočet replaygain. Struktura, ve které se tyto parametry nastavují, se jmenuje settings. Mp3gain všechny parametry nastavoval v metodě main, která obsahuje přez 1000 řádků kódu, takže bylo potřeba tuto strukturu začlenit. Z mého pohledu je mp3gain naimplementován asi nejhůř z hlediska přehlednosti kódu. V této části jsem musel značně zlepšit své schopnosti orientace v cizím kódu, protože jsem se nikdy před tím s ničím podobným nesetkal. Všechny projekty, které jsem dosud dělal, jsem dělal sám, a proto jsem kódu dokonale rozuměl. Teď jsem byl úplně v jiné situaci. Měl jsem před sebou hromadu kódu třech programů, kde každý z nich psal někdo jiný. Čili tři různé styly programování.

Wavegain i vorbisgain používaly k procházení adresářů soubor recurse.c, který umožňuje rekuzivní procházení adresářů do hloubky. To je určitě velmi zajímavé a byla by škoda jej do programu nezačlenit. Bylo tedy třeba nějak vymyslet, abych jej mohl využít i pro mp3 soubory. Rozhodl jsem se využít struktury file_list a vytvořil jsem funkci int process_files (FILE_LIST* file_list, SETTINGS* settings). Musel jsem udělat ve funkci recurse.c patřičné úpravy, aby byla volána právě tato funkce. Celý proces funguje tak, že uživatel zadá pomocí parametru, že chce procházet adresáře rekurzivně. V metodě main se zavolá funkce process_argument(argv[i], &settings). Tato funkce vždy pro danný adresář naplní strukturu file_list a zavolá funkci process_files, které předá tuto strukturu. Tento proces se opakuje pro každý libovolně zanořený adresář od kořenového, který specifikuje uživatel. Nyní tedy máme soubory, se kterými můžeme pracovat. Problém teď spočíval v tom, že všechny programy vždy počítaly s tím, že dostanou kolekci souborů, nad kterou budou počítat replaygain. Musel jsem tedy funkce těchto programů upravit tak, aby počítaly vždy jen s jedním souborem. Opět to pro wavegain a vorbisgain byl mnohem menší problém než pro mp3gain. Nakonec se mi podařilo vše upravit a funkcionalita je tedy následující:

- Získat požadované soubory do struktury file_list.
- Procházet jednotlivé soubory z této struktury a podle přípony volat funkce které umí tyto soubory dekódovat.
- V nich pak volat výpočet replaygain pomocí funkce analyze_samples.
- Na konci výpočtu track gain pro všechny soubory spočítat album gain.

4.2 Paralelizace

V této části programu jsem se tedy mohl pustit do paralelizace výpočtu. Nejdříve bylo potřeba rozhodnout na jaké úrovni paralelizovat.

Možnosti jsou následující:

a) paralelní zpracování jednotlivých souborů,

b) paralelní zpracování vzorků

Vedoucí mi doporučil paralelní zpracování souborů. Výhodou je relativně snadná implementace. Naproti tomu je nutné zpracovávat paralelně alespoň tolik souborů, kolik je vláken, aby bylo dosaženo potřebné efektivity. Část algoritmu jsem nastínil už v teorii a využil jej pro vysvětlení kritické sekce. FILE_LIST files tedy mají vlakna společný a vybírají si z něj soubory se kterými budou pracovat. Výběr souborů je umístěn v kritické sekci. V této části bakalářské práce jsem na vlastní kůži poznal, jak velký problém může být vytvářet paralelizaci nad programy, které na to nejsou připraveny. Každou globální proměnnou, kterou tyto programy obsahovaly jsem musel umístit do struktury, kterou si inicializuje každé vlákno zvlášť, což znamená že si je vlákna nemohou vzájemně přepisovat. Pojmenoval jsem ji global podle toho, že obsahuje globální proměnne a čítá téměř 100 proměnných. Asi si dovedete představit, kolik zabralo času provést patřičné změny v kódu. Odstranil jsem nějáké globální proměnné a přesunul je do společné struktury a po následném překladu jsem obdržel ohromné množství chyb. Při jejích opravách se mi samozřejmě často stávalo, že jsem někde něco přehlédl a za „odměnu“ jsem dostal neoprávněný přístup do paměti (segmentation fault). Problém této chyby spočívá v tom, že jsem se nedozvěděl na jakém řádku konkrétně se chyba nachází. Proto jsem se musel naučit pracovat s nějákým debugovacím nástrojem, jelikož v takové hromadě kódu jsou jen dvě možnosti. Naučit se pracovat s takovýmto programem, nebo začít znovu od poslední verze, kterou jsem měl uloženou. Ze začátku jsem samozřejmě začínal znovu. Naštěstí jsem po nějáké době začal používat gdb a to mi hodně pomohlo, ale ne vždy. Postupem času se mi tedy dařilo zprovozňovat jednotlivé časti a vše začínalo k mému údivu fungovat. Jediné, s čím jsem si neporadil, bylo dekódování mp3gain. Věnoval jsem mnoho úsilí, aby vše fungovalo jak má, ale bohužel si tam někde vlákna přepisují sdílené proměnné, nepodařilo se mi však najít kde. Proto jsem umístil dekódování do kritické sekce.

4.3 Omezení na počet kanálů

Posledním krokem při vývoji aplikace bylo odstranění omezení na počet kanálů. Hlavní myšlenkou bylo předat výpočetní funkci analyze_samples místo levého a pravého kanálu, reprezentovanými dvěma jednorozměrnými poli, dvourozměrvné pole s jednotlivými kanály a vzorky. Musel jsem tedy upravit funkce jednotlivých programů tak, aby předávaly analyze_samples dvourozměrné pole. Pak následovala úprava samotného gain_analysis.c souboru, ve kterém jsem patřičně upravil výpočet. Výpočet rms viz. kapitola 2.7 jsem upravil tak, jak jsem jej popsal v teorii.

5 Analýza výsledků

Program jsem přeložil a testoval na těchto strojích s uvedenými operačními systémy:
Gentoo:

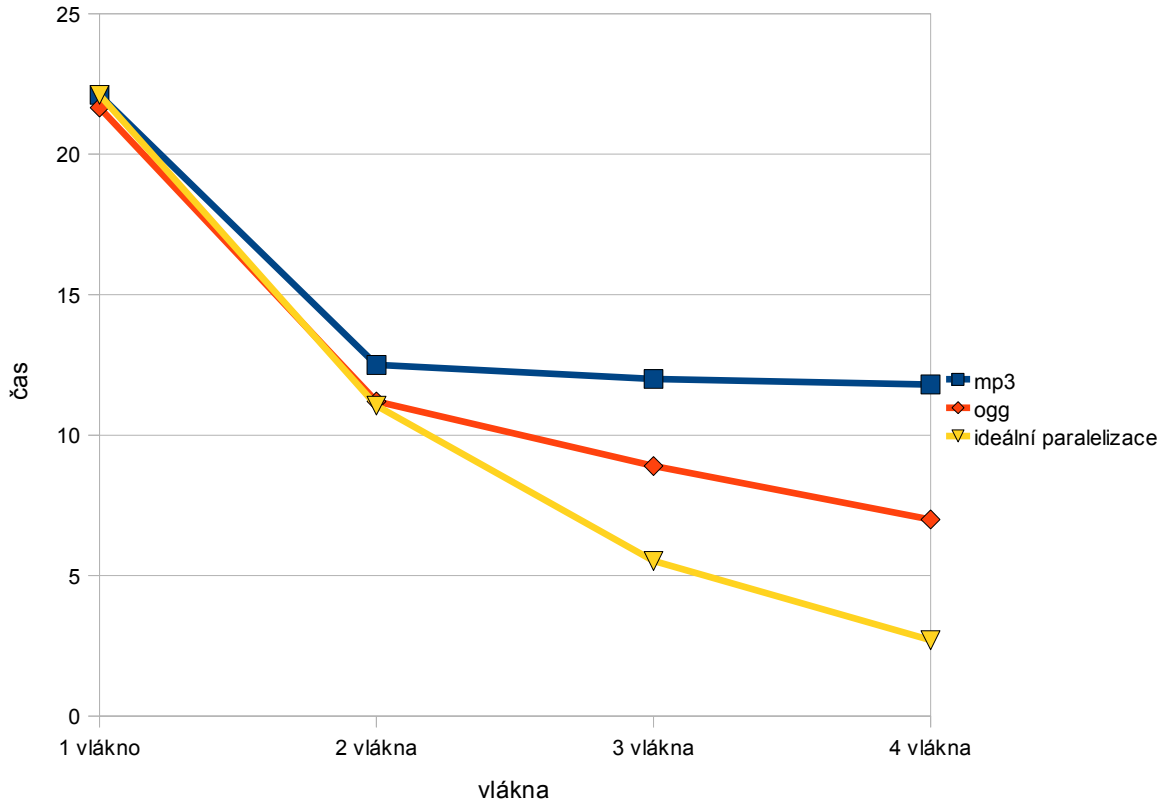
- 2.6.31-gentoo-r10 x86_64
- Intel(R) Core(TM)2 Duo CPU T7100 @ 1.80GHz GenuineIntel – 2 jádra

CentOs:

- tesla01 2.6.18 x86_64 GNU/Linux OS
- Dual-Core AMD Opteron(tm) Processor 2218 – 4 jádra

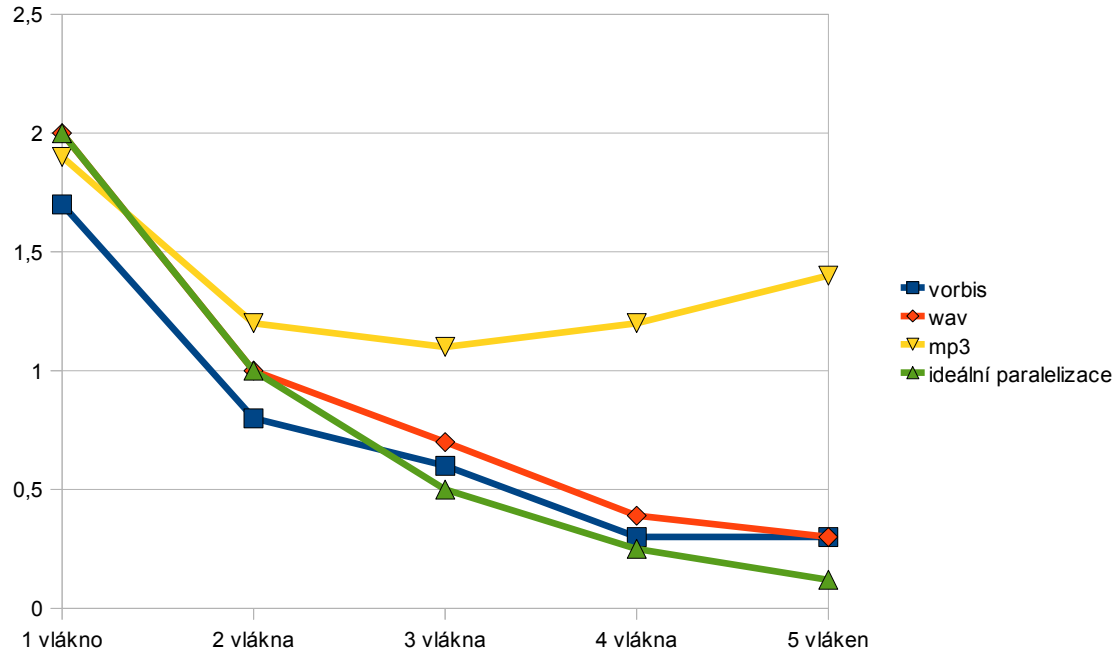
5.1.1 Testy paralelizace

Následující graf 14 zobrazuje výsledky časů pro 1 – 4 vlákna na čtyřjádrovém stroji tesla01. Na vertikální ose jsou hodnoty naměřeného času v sekundách a na horizontální ose počet vláken. Pro měření času jsem použil standardní program time pro měření doby běhu procesu. Testy jsem prováděl na stejném albu ve dvou formátech: mp3 a ogg. Stroj tesla01 jsem měl přístupný pouze přes síť, proto jsem z důvodu poměrně velké kapacity totožného alba ve formátu wav 549 MB rozhodl provést testy na jiných, menších souborech. Specifikaci parametrů těchto souborů uvádím v příloze. Porovnání všech tří formátů stejného alba ukazuji na grafu 16. Test byl proveden na dvoujádrovém stroji. Z grafu 14 je patrné, že u ogg se mi podařilo dasáhnout přibližně trojnásobneho výkonu oproti jednomu vláknu. Také je z grafu vidět, jak mp3 na více než dvou vláknech ztrácí svou efektivitu. Je to z důvodu kritické sekce umístěné v dekódování mp3 souboru.



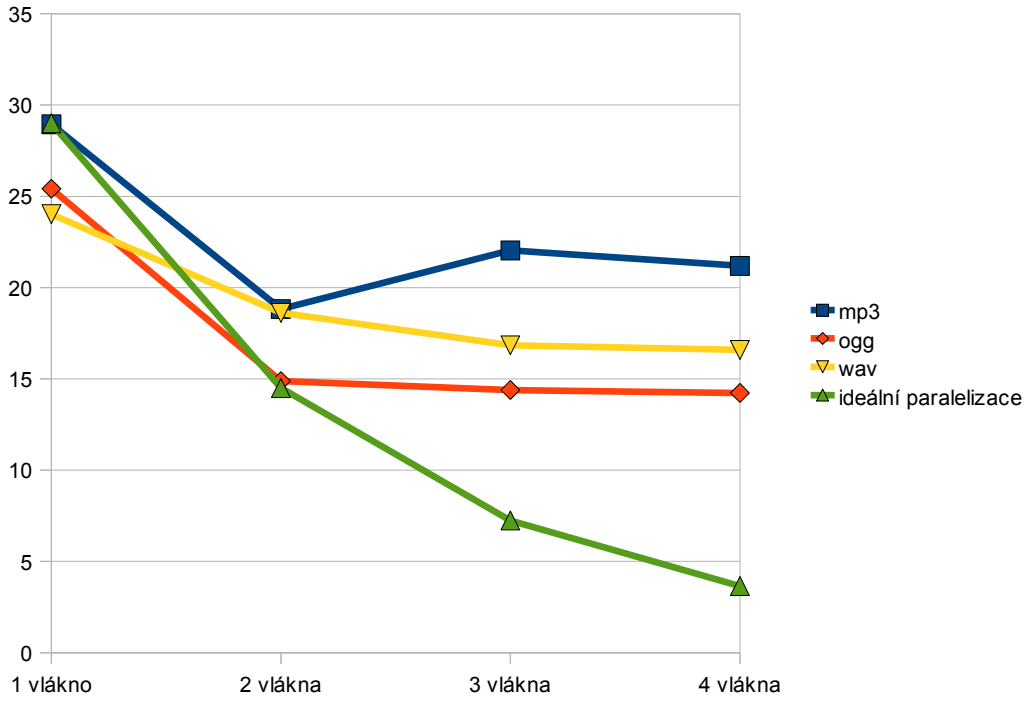
Graf 14:vzorek 1, tesla01 AMD – 4 jádra

Pro testování wav souborů na tesla01 jsem se rozhodl zkopírovat jeden záznam na tento server a udělat 8 kopii tohoto souboru. Audio záznam trvá 30 sekund, takže časy výpočtu jsou kratší, než u předchozích dvou vzorků. Pro porovnání jsem tento soubor převedl ještě do formátů vorbis a mp3. Výsledky viz. graf 15



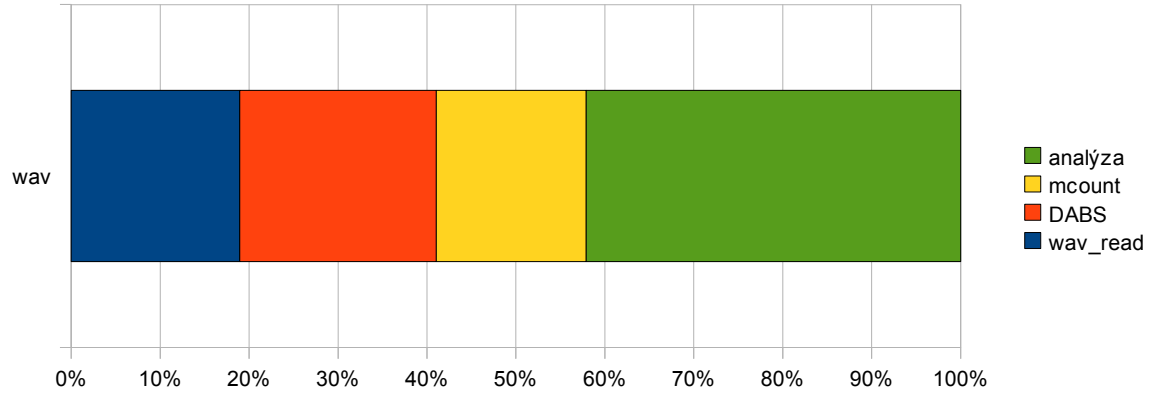
Graf 15: vzorek 2, AMD - 4 jádra

Na následujícím grafu 16 jsou už časy všech tří formátů vzorku 1. Z grafu je patrné, že vzhledem k této architektuře nemá smysl počítat ve více vláknech než je jader procesoru. V mp3 byly výsledky dokonce horší než pro dvě vlákna.

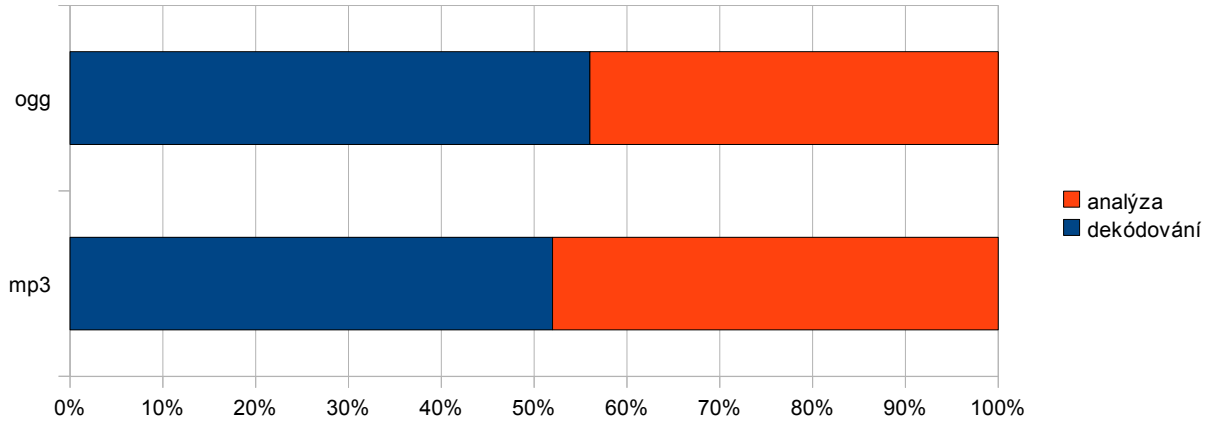


Graf 16: vzorek 1, Intel - 2 jádra

Na dalších dvou grafech prezentuji čas strávený jednotlivými funkcemi programu. Nejvíce nás tedy bude zajímat čas dekódování vůči analýze vzorků. Pro měření časů vykonávání funkcí jsem použil program valgrind s módem callgrind a programem kcachegrind pro zobrazení výsledků. Pro přehlednost jsem vybral jen ty funkce, které mají na celkový čas největší vliv. Pro mp3 a ogg jsem použil jeden graf, protože nejpodstatnější část zabírá dekódování a analýza. Z grafu 18 je patrné, že analýza trvá zhruba stejně dlouho. Z toho tedy můžeme odvodit, proč zpracovávání mp3 dosahuje srovnatelných výsledků s wav pro dvě vlákna, i když je dekódování v kritické sekci. Jakmile se dostanou obě vlákna do smyčky načítání bloků tak na začátku musí druhé vlákno chvili počkat, než první vlákno dekóduje. Následně první vlákno začne provádět analýzu a druhé vlákno vstoupí do kritické sekce. Protože obě operace trvají přibližně stejně dlouho, budou se dále obě vlákna ve svých činnostech střídat. S příchodem dalších vláken tedy nastává problém, protože vždy bude jedno vlákno dekódovat druhé analyzovat a ostatní budou čekat ve frontě před kritickou sekcí.

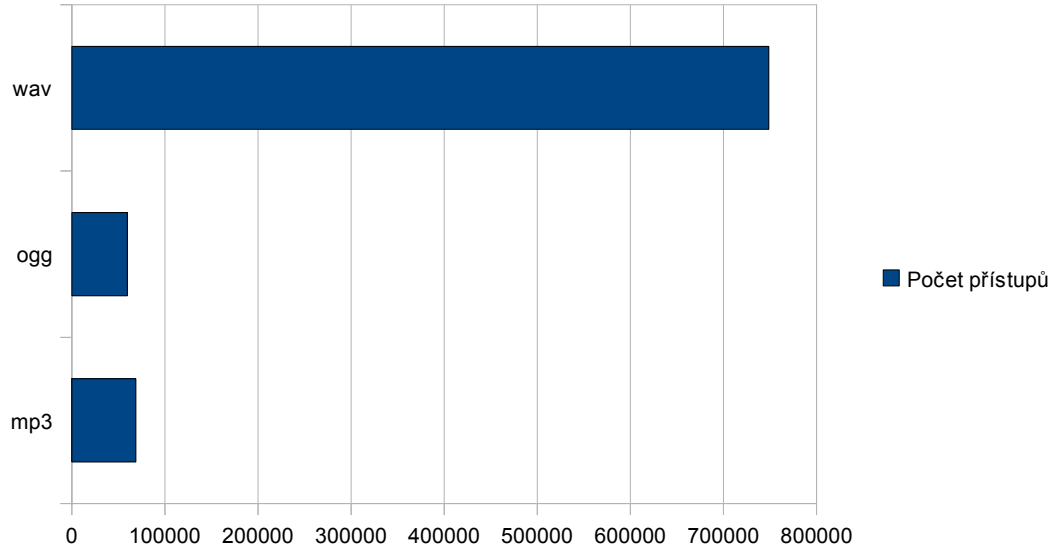


Graf 17: vzorek 1, wav - podíl časů funkcí



Graf 18: vzorek 1, podíl časů dekódování / analýza

Dalším faktorem, který negativně ovlivňuje efektivitu paralelního výpočtu, je přístup na disk. Pro měření diskových operací jsem použil program time. Z grafu 19 je patrné, že zdaleka nejnáročnější je zpracovávání wav souborů. To je také důvod nižší efektivity výpočtu ve vice vláknech naproti formátu ogg. Čím více vláken do výpočtu zahrneme, tím větší bude rozdíl mezi výsledky ogg a wav.



Graf 19: vzorek 1, Počet přístupů na disk

5.1.2 Test zpracování 6-kanálového audia

Při testování správnosti odstranění omezení na počet kanálů jsem použil stejný šestikanálový audio soubor ve formátech wav a ogg. Jak jsem již zmiňoval dříve, mp3 dekódování nepodporuje více než dva kanály takže jej nemůžu testovat. Pro kontrolu správnosti výsledků jsem provedl výpočty ještě v programech winamp a foobar2000. K mému překvapení každý z nich dával jiné výsledky. Po krátké úvaze se nabízí myšlenka, že každý z nich používá trochu jiný algoritmus pro výpočet. Náš algoritmus, který vychází ze stereo zpracování totiž počítá s tím, že do všech kanálů pošleme signál stejně hlasitě. To je sice teoreticky možné, ale prakticky se to tak většinou neděje. Proto zřejmě každý z programů používa trochu jiný algoritmus pro analýzu hlasitostí na jednotlivých kanálech a na základě této analýzy provede výpočet hlasitosti.

	Ogg	wav
Allgain	+0,27 dB	+4,77 dB
Foobar2000	-0,51 dB	+4,22 dB
Winamp	-5,79 dB	tento formát winamp neumí

6 Závěr

Z grafů, které jsem prezentoval v kapitole 5.1.1 Testy paralelizace je patrné, že mé snažení bylo úspěšné. Na vícejádrových strojích ve více vláknech je výpočet efektivnější než při použití výchozích aplikací. Spojení programů také proběhlo úspěšně, takže je pro uživatele mnohem pohodlnější aplikovat výpočty na své sbírce skladeb různých formátů. Mé řešení výpočtu na více než dvou kanálech je diskutabilní, ale vychází z výpočtu pro dva kanály, který navrhl David Robinson.

6.1 Vlastní přínos

Díky téměř nulové dokumentaci jsem se musel naučit orientovat v cizím kódu. Navíc každý z těchto programů implementoval někdo jiný, což jen zvyšovalo nutnost přizpůsobit se. Toto je v praxi velmi důležité, a proto si myslím, že to pro mě byla velmi dobrá zkušenost. Musel jsem pracovat s ohromným množstvím kódu kterému jsem nerozuměl. To vyžadovalo značné zlepšní mých schopností v programování a čtení kódu. Naučil jsem se také používat debugery jako gdb a valgrind, které mi byly velmi užitečné. Při zpětném pohledu už bych se asi pro implementaci rozhodl využít nějákého sofistikovaňejšího vývojového prostředí jako je třeba Kdevelop. Určitě by mi to ušetřilo nějakou práci. Rozhodl jsem se pro vim, protože jsem byl na něj zvyklý z malých projektů, které jsem dělal do školy atd. Na takto velký projekt už je asi vhodnější nějaké sofistikovanější prostředí. Mimo programování jsem se dozvěděl také mnoho zajímavých informací ohledně zvuku.

6.2 Další vývoj

Pro další vývoj určitě zůstávají filtry, na kterých jsem z důvodu vysoké implementační náročnosti nepracoval. Dále je také možné aplikaci rozšířit o další formáty. Ve zpracování mp3 je možné naprogramovat nebo využít nějaký dekóder, který umí více než dva kanály. Dále je možné aplikaci rozšířit o vektorové instrukce.

7 Literatura

[1] ROBINSON, David. *Replay Gain - A Proposed Standard* [online]. 2001 [cit. 2010-05-04]. Dostupné z WWW: http://replaygain.hydrogenaudio.org/.

[2] VŠETIČKA, Martin. *Encyklopedie fyziky* [online]. [cit. 2010-05-04]. Dostupné z WWW: http://fyzika.jreichl.com/.

[3] *Equal loudness contour* [online]. 2001 [cit. 2010-05-04]. Dostupné z WWW: http://en.wikipedia.org/wiki/Equal-loudness_contour.

[4] *Ogg Vorbis Documentation* [online]. 1994 [cit. 2010-05-04]. Dostupné z WWW: http://xiph.org/vorbis/doc/.

[5] *Diskrétní signál* [online]. [cit. 2010-05-04]. Dostupné z WWW: http://cs.wikipedia.org/wiki/Diskrétní_signál.

[6] *Clipping* [online]. [cit. 2010-05-04]. Dostupné z WWW: http://en.wikipedia.org/wiki/Clipping_(audio).

[7] *OpenMP Tutorial* [online]. [cit. 2010-05-04]. Dostupné z WWW: https://computing.llnl.gov/tutorials/openMP/.

8 Přílohy

8.1 Použité soubory

Při testování jsem použil tyto soubory s následujícími parametry pro kompresi.

Vzorek 1:

audio decoder: [mp3lib] MPEG layer-2, layer-3

AUDIO: 44100 Hz, 2 ch, s16le, 128.0 kbit/9.07% (ratio: 16000->176400)

Selected audio codec: [mp3] afm: mp3lib (mp3lib MPEG layer-2, layer-3)

- 3,9M track02.mp3
- 3,2M track03.mp3
- 3,4M track04.mp3
- 3,0M track05.mp3
- 3,5M track06.mp3
- 3,5M track07.mp3
- 2,6M track08.mp3
- 3,1M track09.mp3
- 7,4M track10.mp3

audio decoder: [pcm] Uncompressed PCM audio decoder

AUDIO: 44100 Hz, 2 ch, s16le, 1411.2 kbit/100.00% (ratio: 176400->176400)

Selected audio codec: [pcm] afm: pcm (Uncompressed PCM)

- 52M track02.wav
- 36M track03.wav
- 45M track04.wav
- 33M track05.wav
- 38M track06.wav
- 43M track07.wav
- 29M track08.wav
- 34M track09.wav
- 244M track10.wav

audio decoder: [ffmpeg] FFmpeg/libavcodec audio decoders
AUDIO: 44100 Hz, 2 ch, s16le, 112.0 kbit/7.94% (ratio: 14000->176400)
Selected audio codec: [ffvorbis] afm: ffmpeg (FFmpeg Vorbis)

- 3,3M track02.ogg
- 2,9M track03.ogg
- 2,9M track04.ogg
- 2,6M track05.ogg
- 3,0M track06.ogg
- 2,9M track07.ogg
- 2,2M track08.ogg
- 2,6M track09.ogg
- 6,2M track10.ogg

Vzorek 2:
audio decoder: [mp3lib] MPEG layer-2, layer-3
AUDIO: 44100 Hz, 2 ch, s16le, 128.0 kbit/9.07% (ratio: 16000->176400)
Selected audio codec: [mp3] afm: mp3lib (mp3lib MPEG layer-2, layer-3)

- 252K test.mp3

audio decoder: [ffmpeg] FFmpeg/libavcodec audio decoders
AUDIO: 44100 Hz, 2 ch, s16le, 112.0 kbit/7.94% (ratio: 14000->176400)
Selected audio codec: [ffvorbis] afm: ffmpeg (FFmpeg Vorbis)

- 208K test.ogg

audio decoder: [pcm] Uncompressed PCM audio decoder
AUDIO: 44100 Hz, 2 ch, s16le, 1411.2 kbit/100.00% (ratio: 176400->176400)
Selected audio codec: [pcm] afm: pcm (Uncompressed PCM)

- 2,7M test.wav

8.2 Příručka ke spuštění

Pro překlad programu je nutné mít nainstalovány následující knihovny

- libvorbis
- libsndfile

Při spuštění bez parametrů se zobrazí nápověda. Testováno jak na 64 tak na 32 bit systému. Program jsem překládal pouze na Os Linux. Mp3 se analyzují stejným stylem jako mp3gain s parametrem -s r. Analýza mp3 obsahuje menší problém, který se mi nepodařilo odstranit. Po aplikaci parametrem -r dává při následné analýze špatné výsledky. Proto je potřeba před další analýzou provést undo.