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

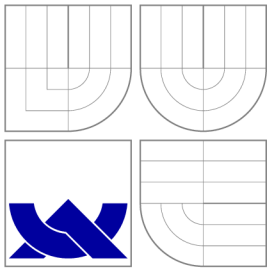
VYUŽITÍ GRAFICKÉHO PROCESORU JAKO AKCELERÁTORU - TECHNOLOGIE OPENCL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

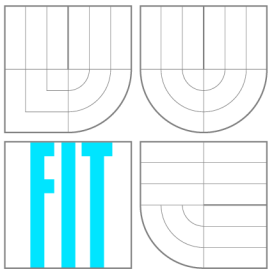
AUTOR PRÁCE
AUTHOR

JOZEF KOBRTEK

BRNO 2010



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

VYUŽITÍ GRAFICKÉHO PROCESORU JAKO AKCELERÁTORU - TECHNOLOGIE OPENCL

USE OF GPU AS ACCELERATOR - TECHNOLOGY OPENCL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JOZEF KOBRTEK

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Dr. Ing. JAN ČERNOCKÝ

BRNO 2010

Abstrakt

Tato práce pojednává o použití grafické karty a rozhraní OpenCL pro akceleraci převzorkování signálu při zpracování zvuku v reálném čase. V práci je analyzována architektura současných grafických karet a programovací model OpenCL, v testech je porovnán výkon GPU a CPU implementace algoritmu. Popsána je též integrace GPU implementace s rozhraním Steinberg VST.

Abstract

This work discusses usage of graphics card and OpenCL API for acceleration of real-time audio resampling. This work contains also analysis of current graphics cards architectures and performance tests comparing CPU and GPU implementations, and integration with Steinberg VST interface.

Klíčová slova

OpenCL, GPGPU, převzorkování, GeForce, Radeon, interpolace, decimace, filtr, SSE, SIMD paradigma, Steinberg VST

Keywords

OpenCL, GPGPU, resampling, GeForce, Radeon, interpolation, decimation, filter, SSE, SIMD paradigm, Steinberg VST

Citace

Jozef Kobrtek: Využití grafického procesoru jako akcelerátoru - technologie OpenCL, bakalářská práce, Brno, FIT VUT v Brně, 2010

Využití grafického procesoru jako akcelerátoru - technologie OpenCL

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. Dr. Ing. Jana Černockého a konzultanta z firmy Audiffex pana Ing. Lubora Přikryla.

.....
Jozef Kobrtek
17. května 2010

Poděkování

Chcel by som poďakovať vedúcemu Doc. Černockému za pomoc a pripomienky, ktoré prispeli k skvalitneniu tejto práce, a tiež za zapožičanie počítača, na ktorom som mohol prácu vytvoriť. Výrazným dielom prispeli tiež programátori firmy Audiffex - pán Ing. Lubor Přikryl, Ing. Jaromír Mačák a obzvlášť Ing. Michal Trzos, ktorým ďakujem za technickú pomoc a množstvo cenných informácií. Matejovi Kolejákovi a Michalovi Perejdovi ďakujem za započičanie grafických kariet na testy.

Obsah

Kapitola 1

Úvod

V posledných rokoch sa vplyvom intenzívneho vývoja pretransformovali grafické adaptéry na univerzálne procesory. Vysoký stupeň paralelizácie a teoretický výkon pohybujúci sa rádovo v teraflopoch otvára nové možnosti pri ich využití na náročné výpočty. Vďaka ich cene, dostupnosti a existencie kvalitných vývojových prostriedkov je možné si za pár tisíc korún zaobstarať výkon superpočítača do bežného PC.

Jedným zo softvérových rozhraní pre programovanie grafických kariet pre všeobecé výpočty je OpenCL. Tento otvorený štandard vznikol pred niečo vyše rokom, no za veľmi krátky čas si našiel množstvo priaznivcov, pričom záber OpenCL nie sú len grafické karty, ale aj procesory, DSP či rôzne akcelerátory - stačí tak, aby výrobca hardware pre svoje zariadenie naprogramoval ovládače schopné využiť jeho potenciál prostredníctvom OpenCL.

Zadaním tejto práce, vypísaným firmou Audiffex, je využiť práve možnosti OpenCL pri spracovaní zvuku v reálnom čase. Ako algoritmus pre implementáciu bolo zvolené prevzorkovanie signálu racionálnym pomerom I/D, ktorý je často používaný pri rôznych druhoch processingu, najmä kvôli potlačeniu aliasingu, ktorý by mohol vzniknúť pri samotnom spracovaní.

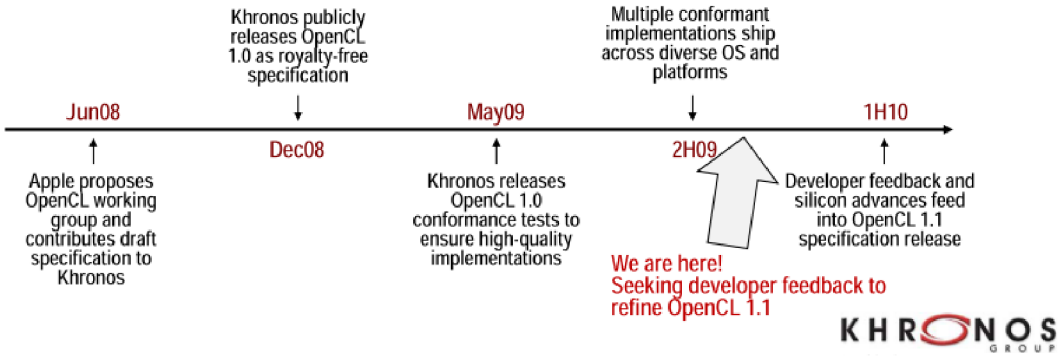
Výsledná implementácia by mala byť demonštrovaná buď ako samostatný celok, alebo formou pluginu. Nakoľko fy. Audiffex vytvára pluginy do profesionálnych programov pre spracovanie zvuku, rozhodol som sa demonštrovať implementovaný algoritmus ako plugin s využitím rozhrania Steinberg VST 2.4.

Prvú kapitolu som venoval analýze rozhrania OpenCL, jeho štruktúry a popisu štyroch základných častí architektúry - platformový, vykonávací, pamäťový a programový model. V skratke je tiež priblížený historický vývoj rozhraní pre všeobecné výpočty počítané na grafickej karte. Nasleduje analýza architektúr súčasných grafických kariet - ako zástupcov som vybral GeForce GTX 280 a GTX 480 od nVidie, a Radeon HD 5870 od AMD. V kapitole číslo 4 je stručne popísané rozhranie VST 2.4 pre tvorbu pluginov, piata kapitola rozoberá samotný algoritmus prevzorkovania. Po popise jednotlivých implementácií pre grafickú kartu, procesor a tvorby VST pluginu nasledujú výkonnostné testy, v ktorých porovnávam výkon GPU a CPU verzie algoritmu.

Kapitola 2

OpenCL

OpenCL, v skratke **Open C**ompute **L**anguage, je otvorené multi-platformové API určené pre programovanie paralelných procesorov. Predstavuje uniformné prostredie pre programovanie heterogénnych systémov, či sa jedná o univerzálne procesory, grafické karty, či špecialované procesory a akcelerátory ako napr. signálové procesory alebo procerory architektúry Cell.[22]



Obrázok 2.1: Vývoj OpenCL, prevzaté z [30]

OpenCL bolo vyvinuté spoločnosťou Apple v spolupráci s firmami Intel, AMD, IBM a nVidia. Štandard spravuje konzorcium Khronos, ktoré sa stará napr. aj o OpenGL či iné otvorené štandardy [9]. Prvá špecifikácia OpenCL bola vydaná v novembri 2008, avšak prvé stabilné OpenCL SDK pre programovanie na grafických kartách sa objavilo až v septembri 2009 (nVidia) - jedná sa teda o veľmi mladý štandard, čo dokladá aj časová os vývoja štandardu na obrázku 2.1 [9].

2.1 Historický vývoj použitia grafických kariet na všeobecné výpočty

Nakoľko v súčasnoti sa uplatňuje trend zvyšovania počtu jadier procesorov či počet výpočtových jednotiek, vznikla potreba pre efektívny programovací nástroj, ktorý by využil potenciál multi-jadrových procesorov. Podmienkou bol dostatočne nízkoúrovňový prístup k hardware, na druhej strane požadovaná určitá miera transparentnosti pri prideľovaní úloh

jednotlivým výpočtovým jednotkám/jadrám. Aplikácia SIMD (Single Instruction Multiple Data) techník napr. v x86 procesoroch je veľmi komplikovane použiteľná - vyžaduje znalosť assembleru (MMX a SSE inštrukčné sady), čo by spĺňalo podmienku pre nízkoúrovňový prístup, na druhej strane tu však nie je transparentnosť voči programátorovi z pohľadu prideľovania úloh - všetko si musíme ošetriť ručne, prípadne použiť programovací jazyk vyššej úrovne a využiť možnosť optimalizácie prekladača, avšak výsledný výkon zrejme nebude taký, ako by sme očakávali.

Karty generácie DirectX9 (GeForce FX - 7900; Radeon 9500 - X1900) disponovali separátnymi zreťazenými linkami (pipelines) pre spracovanie pixelov a vrcholov, pričom ich bolo možné programovať pre všeobecné výpočty s použitím OpenGL alebo DirectX. Prvotiny GPGPU [1] vznikli s použitím off-screen renderingu u OpenGL (t.j. použitie off-screen buffera namiesto zobrazovacieho framebuffera, kam sa ukladali výsledky), pričom programovanie bolo pomerne náročné - bolo nutné napr. transformovať polia na textúry. Toto bolo možné len s kartami podporujúcimi DirectX9 [18].

Neskôr sa na univerzite v Standforde vyvinul jazyk BrookGPU, ktorý predstavuje front-end pre GPGPU programovanie, stále však neposkytuje priamy prístup k hardware. Vychádza z jazyka C, pričom ako back-end je schopný využívať napr. OpenGL 1.3, DirectX9 alebo už zaniknutý štandard AMD Close To Metal, čím odpadá nutnosť transformácie algoritmov do grafického poňatia. Jeho výhodou je, že poskytuje rozhranie pre GPGPU na starších grafických kartách, ktoré OpenCL nepodporujú - napr. karty ATi Radeon X1800 [6].

Prielom nastal v roku 2006 vydaním GeForce radu 8, spolu s ktorou bolo uvedené aj rozhranie CUDA (Compute Unified Device Architecture). Vďaka unifikovanej architektúre DirectX10-kompatibilných kariet bolo možné využívať ich výpočtové jednotky pre všeobecné výpočty bez nutnosti využívať OpenGL či DirectX - CUDA, podobne ako OpenCL, pristupuje priamo k hardware. Práve CUDA je prvým API, ktoré spĺňa vlastnosti na nízkoúrovňový prístup, ale aj transparentnosť organizácie vlákien voči programátorovi. Ako základ využíva jazyk C/C++, teda je dostatočne nízkoúrovňové, navyše vďake modelu vykonávania je CUDA aj transparentné rozhranie, nakoľko sa programátor nemusí starať napr. o prepínanie vlákien na vykonávacích jednotkách. To znamená, že sme dostatočne blízko hardware na to, aby sme mohli algoritmy efektívne optimalizovať, na druhej strane sme od neho dostatočne ďaleko na to, aby sme nemuseli programovať každý jeden výpočtový element - od toho je tu CUDA runtime, prípadne je to v réžii karty. Jeho zásadnou nevýhodou je fakt, že sa jedná o proprietárne API, ktoré funguje len na grafických kartách od nVidie.

Výpočtový model OpenCL bol do značnej miery inšpirovaný práve rozhraním CUDA - typy pamätí sa líšia len názvami, podobne aj zoskupenia vlákien. CUDA je však jednoduchšia, najmä čo sa týka inicializácií (viac v kapitole venovanej analýze OpenCL).

Rozhraním, ktoré sa objavilo po OpenCL, je DirectCompute, ktoré je súčasťou DirectX verzie 11.

2.2 Hierarchický model OpenCL

Programovanie v OpenCL s využitím GPU zahŕňa prostriedky dvoch systémov - hostiteľkého procesora a grafického adaptéra. Podobne ako CUDA OpenCL nevyužíva OpenGL

[1] GPGPU - General-Purpose computing on Graphics Processing Units, použitie grafickej karty na všeobecné výpočty

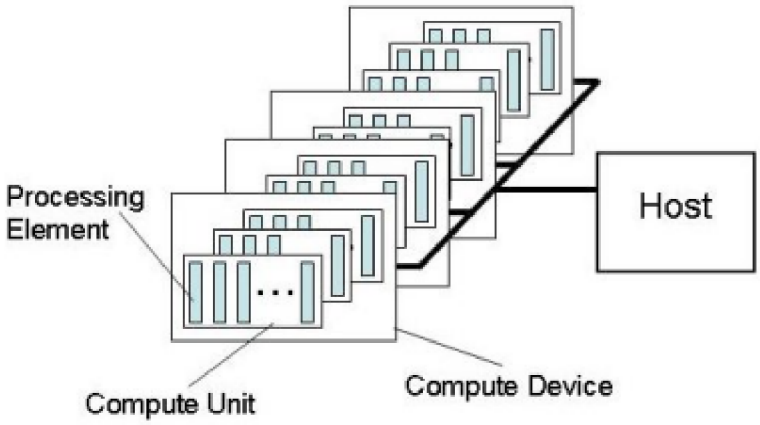
či DirectX ako back-end, ale siaha priamo hardware grafickej karty - koniec koncov jedná sa o nástroj pre programovanie akýchkoľvek procesorov. Väčšina informácií v tejto podkapitole bola voľne prevzatá z [22].

Architektúru OpenCL môžeme rozdeliť do niekoľkých hierarchicky usporiadaných modelov:

- Platformový model
- Vykonávací model
- Pamäťový model
- Programovací model

2.2.1 Platformový model

Model platformy popisuje hierarchiu jednotlivých častí systému, na ktorom pobeží nami vyvíjaná OpenCL aplikácia.



Obrázok 2.2: Platformový model OpenCL, prevzaté z [22]

Hostiteľský systém (**host**) pozostáva z jedného alebo niekoľkých OpenCL-kompatibilných procesorov (**compute device**). Môžeme mať napr. systém, v ktorom máme okrem x86-kompatibilného procesora[2] aj grafickú kartu podporujúcu OpenCL.

Každé zariadenie pozostáva z niekoľkých výpočtových jednotiek (**compute unit**). V reále sú to napr. jednotlivé jadrá procesora alebo multiprocesory grafickej karty. Tie sa dalej delia výpočtové elementy (**processing elements**), čo sú napr. SSE jednotky procesora alebo jednotlivé jadrá multiprocesorov grafickej karty [22].

2.2.2 Vykonávací model

Vykonávací model pozostáva z dvoch častí - na hostiteľskom procesore beží hostiteľský program, pričom na OpenCL zariadení(zariadeniach) sú spustené tzv. **kernely** - základné

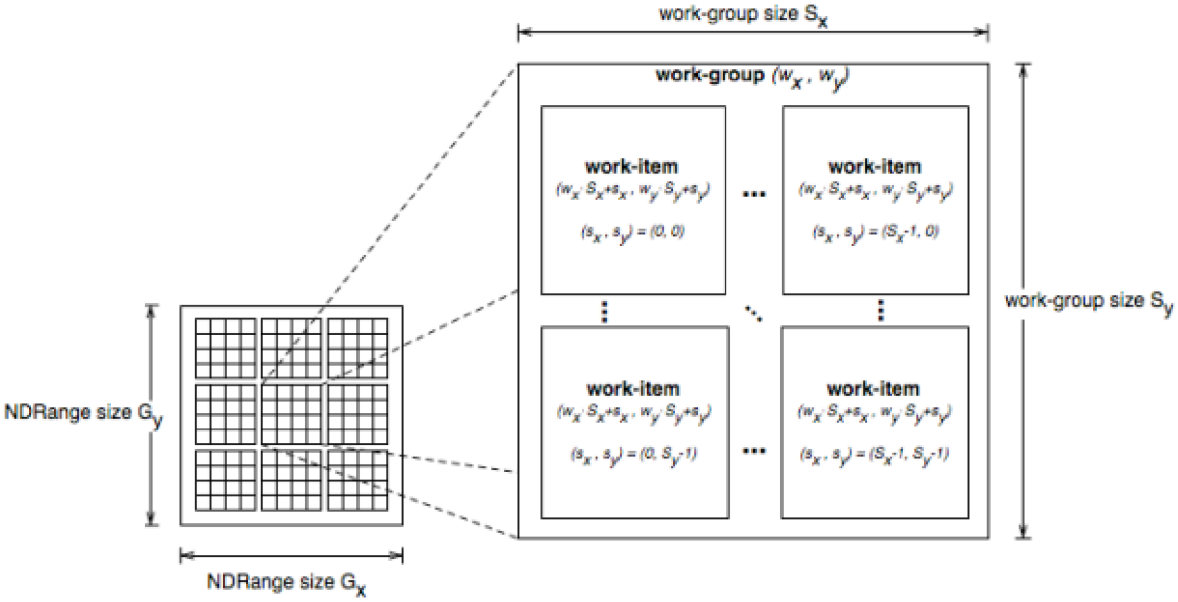
[2]SDK nVidie pre platformu Windows nezahrňuje podporu procesora, len grafickej karty, na rozdiel od AMD, ktoré má podporu CPU aj GPU

jednotky spustiteľného kódu na cieľovom zariadení (v našom prípade grafickej karte) [26]. Zaisťuje paralelné vykonávanie kódu na OpenCL zariadení.

Vykonávací model OpenCL práve definuje, akým spôsobom budú kernely na OpenCL-kompatibilných zariadeniach vykonávané. Inštancia kernelu spustená na jednom výpočtovom elemente sa nazýva **work item**, pričom tieto inštancie (vlákna) sú jednoznačne identifikované v rámci globálneho indexového priestoru - každá má v tomto prirestore pridelené jednoznačné ID.

Tento indexový priestor sa nazýva **NDRange** - N-dimenzionálny indexový priestor, pričom N môže nadobúdať hodnoty $1, 2$ alebo 3. To nám umožňuje rozdelenie inštancií kernelov do požadovaného počtu dimenzií odpovedajúce zložitosti riešeného problému (lineárny, 2-rozmerný - napr. matica, priestorový).

Aby sa dosiehol určitý stupeň granularity v usporiadaní work itemov, sú ďalej zoskupené do pracovných skupín (**work groups**). Tie sú, podobne ako work itemy, očíslované, pričom každé vlákno má okrem globálneho ID aj identifikátor v rámci skupiny. Vlákna v rámci pracovnej skupiny dokážu komunikovať medzi sebou prostredníctvom zdieľanej pamäte.



Obrázok 2.3: 2-dimenziálny NDRange a organizácia vlákien v OpenCL, prevzaté z [22]

Globálne ID vlákna je pritom možné vypočítať z lokálneho ID a ID pracovnej skupiny. Veľkosť pracovnej skupiny je však obmedzená, toto obmedzenie je u každého procesora iné, závisí na konkrétnom cieľovom hardware. Taktiež existuje limit pre počet vlákien v každej dimenzii.

Na hostiteľskom systéme sa tiež definuje **kontext** vykonávania kernelov. Kontext je abstraktná štruktúra, ktorá v sebe zapuzdruje celý vykonávací model, a zahŕňa v sebe :

- Zariadenia (procesory), na ktorých budú prebiehať výpočty
- Kernely, funkcie ktoré budú vykonávané na týchto procesoroch
- Pamäťové objekty, ku ktorým pristupuje hostiteľský systém a OpenCL zariadenia

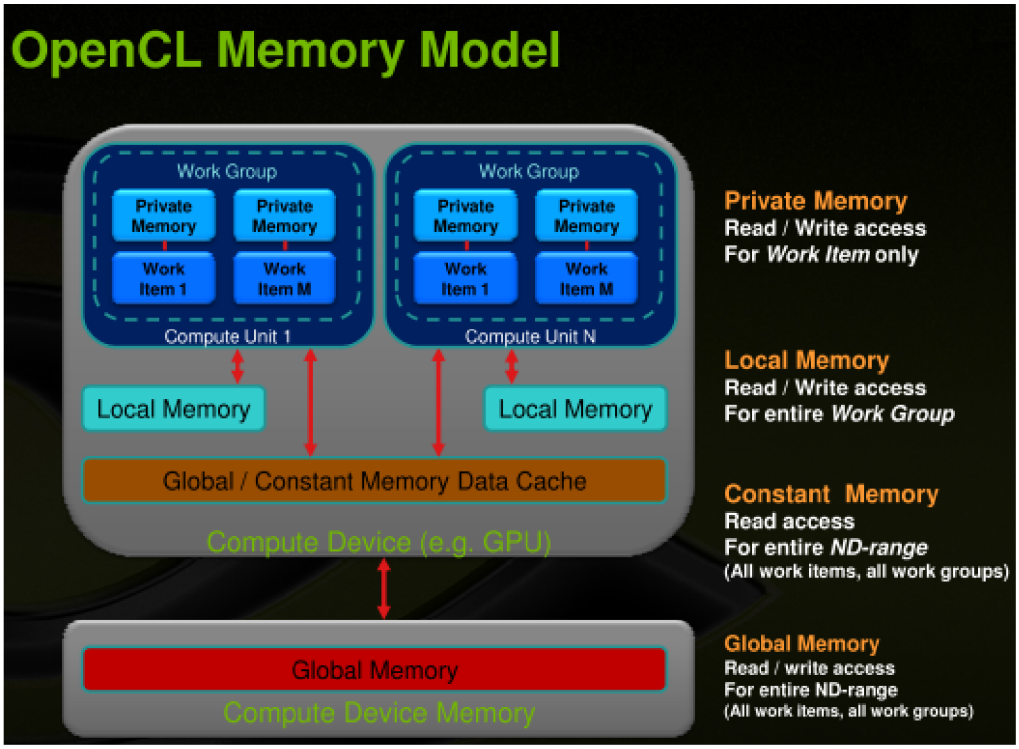
V rámci kontextu definujeme fronty príkazov k jednotlivým zariadeniam (tzv. **command queues**), do ktorých hostiteľ zaraďuje jednotlivé príkazy - vykonanie kernelu, manipulácia s pamäťovými objekatmi (presuny dát z a do zariadenia) alebo synchronizačné príkazy. Príkazy vo fronte môžu byť vykonávané v poradí, alebo mimo poradia, závisí na programátorovi. Pre synchronizáciu je možné buďto označiť volanie ako synchrónne, ale pomocou udalostí (**events**) zaručiť in-order vykonanie (dokončenie určitej úlohy môže vyvolať udalosť).

V softvérovom poňatí sa aplikácia vykonáva na hostiteľskom systéme, OpenCL kód na kontexte zariadení, kernel na konkrétnom OpenCL zariadení, pričom jedna work group je pridelená jednej jeho výpočtovej jednotke. Každý výpočtový element vykonáva inštrukcie jedného vlákna (work item).

2.2.3 Pamäťový model

OpenCL definuje 4 základné typy pamäte, v závislosti na lokácii a prístupu:

- Globálna pamäť
- Pamäť pre konštanty
- Lokálna pamäť
- Privátna pamäť



Obrázok 2.4: Typy pamäťových objektov a ich umiestnenie v systéme, prevzaté z [3]

Globálna pamäť je prístupná pre zápis aj čítanie všetkým vláknam. Z pohľadu grafickej karty sa jedná o video RAM, a nie je cache-ovaná. Jedná sa o najpomalšiu pamäť - prístup do nej stojí obvykle 400-600 hodinových cyklov u kariet nVidia GeForce [27].

Konštantná pamäť je typ globálnej pamäte, ktorú inicializuje hostiteľský systém, jednotlivé vlákna z nej môžu iba čítať. Na grafickej karte sa nachádza mimo čip, ale pre rýchlejší prístup je cache-ovaná (u kariet od nVidie) [27].

Lokálna pamäť predstavuje spoločnú pamäť pre skupinu vlákien v jednej work group. Do nej môžu jednotlivé vlákna v rámci skupiny zapisovať, a aj z nej čítať. Jedná sa o pamäť, ktorá sa nachádza priamo na grafickom čipe, prístup do nej je veľmi rýchly. U kariet od nVidie sa tiež jedná o pamäť, v ktorej sú uložené parametre kernelu pri jeho spustení.

Privátna pamäť je pamäť vyhradená len jednému vláknu, ktoré nad ňou môže vykonávať ako operácie zápisu, tak aj čítania. Z pohľadu grafickej karty sa jedná o najrýchlejšiu pamäť - registre.

Z pohľadu optimalizácií je vhodné minimalizovať použitie globálnej pamäte, pričom sa treba zamerať na caching dát do lokálnej pamäte, ktorá je nielen rýchlejšia, ale u nej nedochádza k problémom ako nezarovnaný prístup [27]. Okrem lineárnych pamäťových objektov existujú aj 2 a 3-rozmerné objekty - obrázky, textúry, alebo obsah frame-buffera (ak pracujeme na GPU). Pri ich sprcúvaní je však nutné využívať vstavaných funkcií OpenCL. Nachádzajú sa v **textúrovacej pamäti** s read-only prístupom, na druhej strane je však cache-ovaná, a v prípade dobrej 2D lokality načítavaných dát poskytuje vyššiu priepustosť než globálna pamäť, navyše nepozná problém nezarovnaného prístupu. Teoreticky je možné použiť ju namiesto globálnej pamäti, avšak práca s ňou je komplikovanejšia - len prostredníctvom vstavaných funkcií [3].

2.2.4 Programovací model

Hostiteľské funkcie OpenCL (vytváranie kontextu a pod) sú vďaka knižniciam dostupné v mnohých jazykoch, dokonca už aj v Pythone (pyOpenCL). [14] Samotný jazyk OpenCL, v ktorom sa píšu jednotlivé kernely, vychádza z jazyka ISO C99. Oproti tomuto štandardu však má niekoľko obmedzení [20]:

- Rekurzia
- Ukazatele na funkcie
- Bitové polia
- Polia s variabilnou dĺžkou
- Hlavičkové súbory
- Ukazatele na ukazatele len v rámci kernelu, nie ako parametre

Pribudli však vektorové dátové typy, synchronizačné príkazy a množstvo vstavaných funkcií, napr. pre prácu s obrazovými dátami (avšak zápis do 3D obrazu nie je možný). Taktiež sa tu objavujú identifikátory pamäťového priestoru a kernelov [33].

Vytvoriť program v OpenCL obnáša nasledovné kroky:

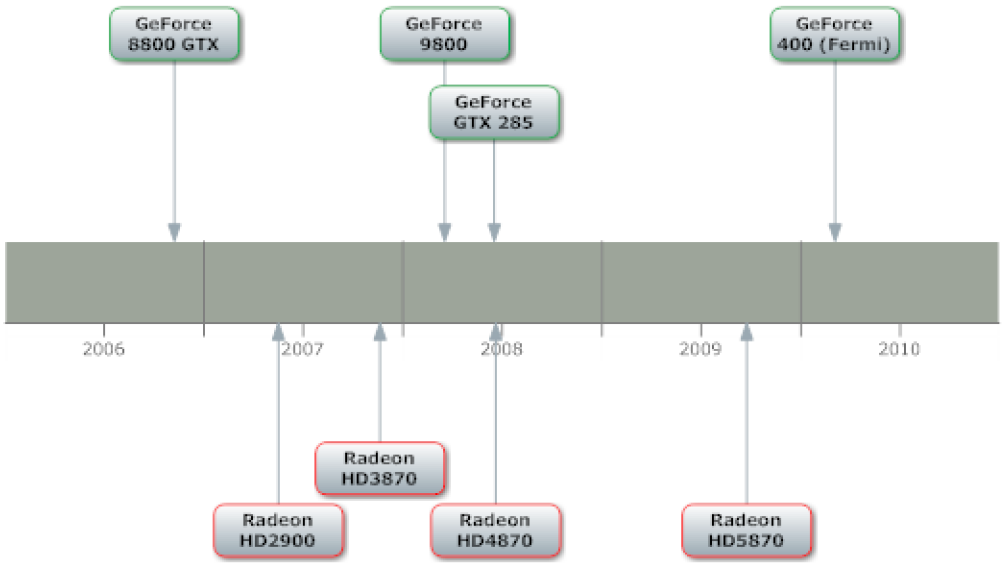
- Vytvorenie kontextu nad zariadeniami
- Vytvorenie pracovných front

- Kompilácia OpenCL programu zo zdrojového kódu, vytvorenie kernelov
- Alokácia pamäťových objektov
- Kopírovanie dát (pamäťových objektov) do zariadenia
- Vykonanie kernelu (kernelov), synchronizácia
- Získanie výsledkov

Kapitola 3

Architektúra grafických čipov

S nástupom DirectX10 generácie (v súčasnosti už DirectX11) grafických kariet sa otvorilo ich nové pole pôsobnosti na poli high-performance computingu. Vďaka ich unifikovaným výpočtovým SIMD jednotkám, ktoré nahradili dlhodobo zaužívaný koncept oddelených pipelines pre pixely a vrcholy, ich môžeme použiť na akékoľvek výpočty vyžadujúce vysoký stupeň paralelizácie.



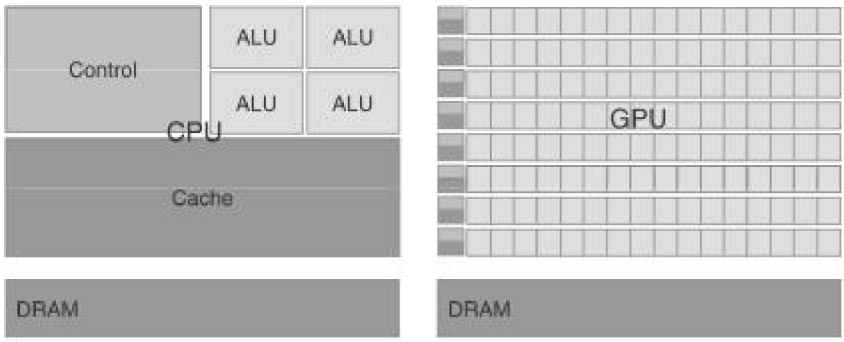
Obrázok 3.1: Uvedenie jednotlivých generácií grafických kariet s unifikovanými výpočtovými jednotkami od nVidie a AMD/ATi

Hoci všetky karty nVidie, ktoré boli uvedené po novembri 2006 (t.j. GeForce radu 8 a vyššie) podporujú OpenCL, u AMD to tak nie je. Karty AMD Radeon radu 2000 a 3000 s OpenCL nie sú kompatibilné a tento štandard nepodporujú [10]. Rada HD 4000 síce OpenCL podporuje, avšak čipy tejto generácie neboli preň navrhované, preto ich výkon značne zaostáva za kartami od nVidie - GeForce GTX 260 je v OpenCL približne 5x rýchlejšia ako HD4870, napriek tomu, že v grafických aplikáciach sa jedná o takmer rovnocenné karty [24]. Jedným z dôvodov je napr. emulácia lokálnej (najrýchlejšej) pamäti na globálnej (najpomalšej) [4]. Záleží tiež na konkrétnom algoritme.

3.1 Porovnanie grafických kariet a procesorov

Výkon súčasných procesorov pre všeobecné výpočty používané v dnešných PC (t.j. procesory architektúry x86) je zlomkom teoretického výpočtového výkonu grafických kariet. Je to dané v prvom rade ich odlišným zameraním, a z toho vyplývajúci odlišný dizajn. Počet jadier sa veľmi dlho držal na čísle 1, pričom len pred pár rokmi sa ich počet začal pomaly zvyšovať. V súčasnosti sú v desktopovom segmente procesory s 1 až 6-timi jadrami, niektoré nové serverové procesory majú až 12 jadier. x86 procesory sú tiež vybavené inštrukčnou sadou SSE (Streaming SIMD Extensions), umožňujúcu vykonávať jednu operáciu nad niekoľkými dátovými tokmi, ktoré sa využívajú predovšetkým v multimediálnych aplikáciach [23].

Procesory sú stavané na vykonávanie sekvenčného kódu, obsluhu periférií, diskov. Obsahujú z pravidla väčšie množstvo vyrovnávacej pamäte pre zníženie latencií pri prístupe do hlavnej pamäte RAM, prípadne iné pamäte typu cache ktoré urýchľujú napr. preklad virtuálnej na fyzickú adresu či skokovú predikciu. Pri programovaní poskytujú väčšiu kontrolu nad sekvenčným vykonávaním kódu [23].



Obrázok 3.2: Porovnanie architektúr procesorov a grafických čipov, prevzaté z [23]

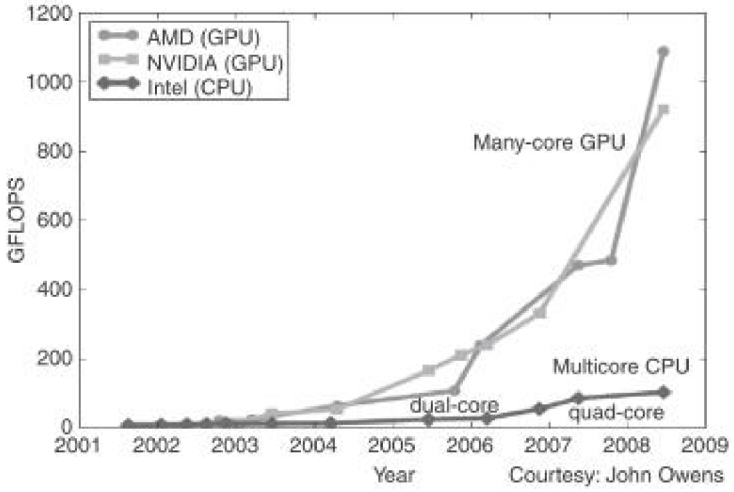
Filozofia návrhu súčasných grafických čipov je do značnej miery odlišná. Disponujú veľkým počtom jednoduchších jadier, pričom ich obslužná logika (prepínanie vlákien) je implementovaná priamo na úrovni hardware, čo umožňuje veľmi rýchle prepínanie kontextu. Jadrá grafickej karty nedisponujú tzv. out-of-order execution (vykonávanie inštrukcií mimo poradia), čo je doménou procesorov. Nemajú ani tak veľké vyrovnávacie pamäte (rádovo kilobajty) [23], za to sú vybavené veľmi rýchlou video RAM, pripojenou obvykle širšou zbernicou než procesor k systémovej RAM. Napr. dvojkanálové zapojenie pamäte typu DDR3 SDRAM má šírku 128 bitov [13], pričom pri použití najrýchlejších DDR3 modulov podľa oficiálnej špecifikácie JEDEC[1] - 1600MHz je teoreticky schopné dosiahnuť 25,6 GiB/s. Šírka pamäťovej zbernice u GeForce GTX 285 je 512 bitov, s použitím GDDR3 pamäti s frekvenciou 2484 MHz dosahuje priepustnosť 159 GiB/s. Podobnú priepustnosť má aj napr. AMD Radeon 5870 [11]. U nového high-end modelu, GeForce GTX 480, je pripustnosť pri 384-bitovej zbernici s použitím rýchlejších GDDR5 pamätí o frekvencii 3696 MT/s[2] až 177 GiB/s [12].

Potenciál pamäťovej priepustnosti využíva niekoľko stoviek výpočtových jadier - u súčas-

[1] podľa štandardu JEDEC JESD79-3D, september 2009 [21]
[2] MT/s - mega (10^6) transferov (dátových prenosov) za sekundu.

ného najvyššieho modelu nVidie (GeForce GTX 480) je to 480, AMD Radeon HD 5870 má 1600 Stream procesorov, avšak odlišnú architektúru. V prepočte by jednalo o 320 výpočtových jadier ekvivalentnými s jadrami na kartách GeForce [11].

Každá skupina výpočtových jadier zdieľa sadu aritmeticko-logických jednotiek, špeciálnych jednotiek pre výpočet goniometrických funkcií, prevrátenej hodnoty a výpočty s dvojnásobnou presnosťou. Kým aktuálne najvýkonnejší procesor určený na desktop, Core i7 980XE, dosahuje maximálny aritmetický výkon 79,9 gigaFLOPS[3], výkon grafických kariet sa meria na teraFLOPS - GeForce GTX 285 dosahuje 1,062 TFLOPS, GeForce GTX 480 1,344 TFLOPS, AMD Radeon HD 5870 až 2,72 TFLOPS [12] [11]. Teoretický výkon



Obrázok 3.3: Nárast výkonu procesorov a grafických čipov po rok 2009, prevzaté z [23]

a stupeň paralelizácie grafických kariet ich predurčuje k urýchleniu náročných výpočtov, simulácií či iných problémov, ktoré z týchto dvoch hlavných vlastností dokážu profitovať.

3.2 Grafické čipy nVidia GeForce

nVidia bola prvou firmou, ktorá na trh uviedla grafickú kartu navrhnutú pre použitie na všeobecné výpočty. Bola ňou GeForce 8800GTX, predstavená v novembri 2006, spoločne s ktorou bolo uvedené aj programovacie rozhranie CUDA. Na rozdiel od neskorších modelov nepodporuje napr. prácu s FP64 či atomické inštrukcie. Z jej architektúry však čerpali nasledujúce generácie grafických čipov GeForce, rada 9800 okrem zvýšenia počtu výpočtových jadier (Shader procesorov) pridala podporu pre atomické inštrukcie, GeForce GTX 200 zase aritmetiku s dvojnásobnou presnosťou.

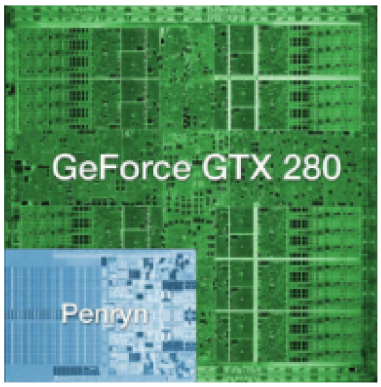
[3]FLOPS - Floating Point Operations Per Second, počet operácií v plávajúcej desatinnej čiarke za sekundu

3.2.1 GeForce GTX 280/285

Obrázok 3.4: Referenčná nVidia GeForce GTX 280, prevzaté z nvidia.com

GeForce GTX 280 bola uvedená na trh v júni 2008 ako najvýkonnejšia z 3. generácie DirectX10 grafických kariet spoločnosti nVidia.

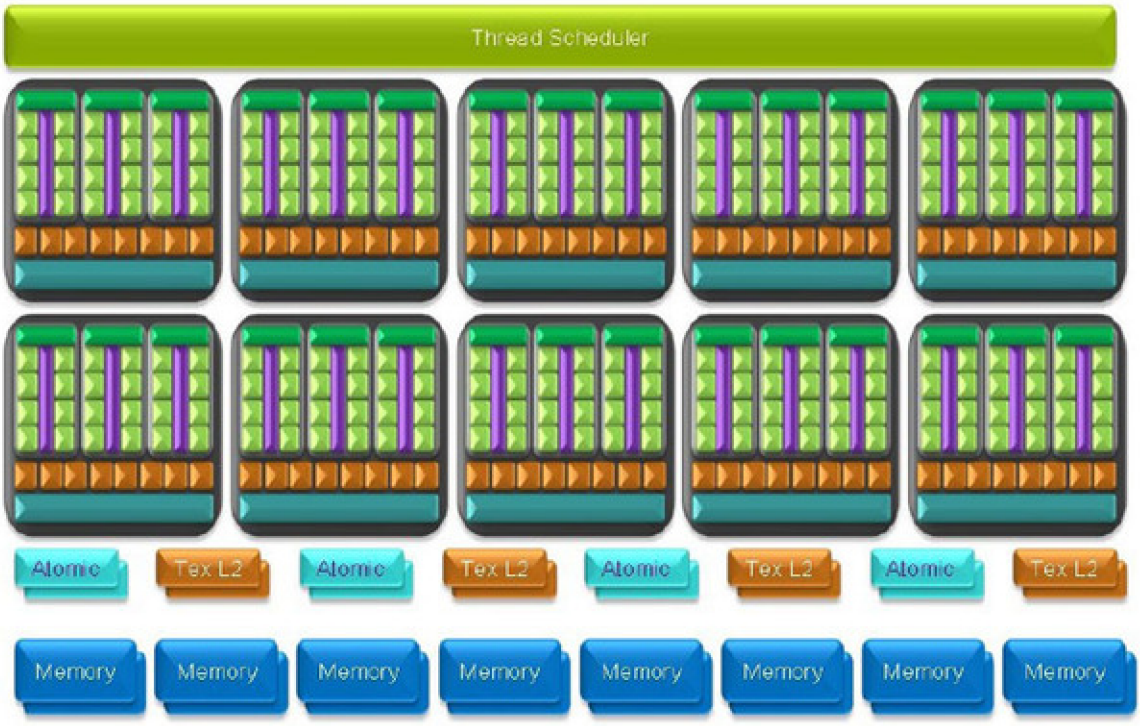
Jej jadro, označené aj ako GT200, pozostáva z 1,4 miliardy tranzistorov, vyrábaných 65 nm výrobným procesom, pričom jeho celková plocha činí 576 mm^2. Neskôr prešla výroba na 55 nm proces, potom je karta známa ako GTX 285. GT200 má 2 hlavné frekvenčné



Obrázok 3.5: Porovnanie plochy jadra GeForce GTX 280 a dvoj-jadrového procesora Core 2 Duo, prevzaté z techreport.com

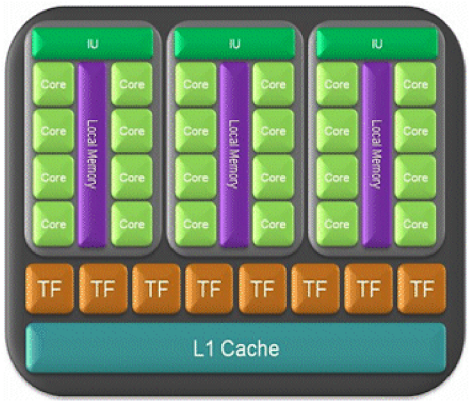
domény - jadro a samotné výpočtové procesory. Tie bežia na frekvencii viac ako dvojnásobnej oproti zvyšku čipu. (1296MHz oproti 600MHz).

Samotné výpočtové jadrá (Shader procesory), ktorých je 240, sú rozdelené do desiatich klastrov (TPC, Thread Processing Cluster) po 24, pričom vnútri každého TPC sa delia na 3 multiprocesory (SM, Streaming Multiprocesor), teda 8 Shader procesorov na jeden multiprocesor (obr. 3.6). V rámci klastra zdieľajú multiprocesory 8 textúrovacích jednotiek, ako vidno na obrázku 3.7. Z pohľadu OpenCL je každému multiprocesoru priradená jedna sada vlákien (work group). Každý SM disponuje 16KiB zdieľanej vyrovnávacej pamäte, pomocou ktorej môžu jednotlivé výpočtové jednotky v rámci multiprocesora medzi sebou komunikovať - v OpenCL terminológii sa jedná o lokálnu pamäť. Ďalej je vybavený dvoma



Obrázok 3.6: Architektúra grafického čipu GT200, rozloženie a hierarchia výpočtových jadier, prevzaté z [31]

jednotkami pre špeciálne funkcie (Special Function Unit), ktoré poskytujú natívnu implementáciu goniometrických funkcií, alebo napr. prevrátenú hodnotu. Prítomná je tiež jednotka pre počítanie v dvojnásobnej presnosti (FP64, 64-bitové číslo s plávajúcou desatinnou čiarkou) [31].



Obrázok 3.7: Detail na Thread Processing Cluster; IU - Instruction Unit, TF - Texture Filter, prevzaté z techreport.com

V každom multiprocesore sa nachádza tzv. warp scheduler (Instruction Unit na obr.

3.7), ktorý prideľuje inštrukcie jednotlivým výpočtovým jadrám. Warp označuje termín pre skupinu vlákien, u ktorých je naraz vykonaná rovnaká inštrukcia. Nakoľko každé výpočtové jadro obsahuje zreťazenú linku, v ktorej udržiava až 4 rozpracované inštrukčné toky (zo 4 vlákien), celkovo je naraz rozpracovaných až 32 vlákien na jednom multiprocesore, pozostávajúcom z ôsmych výpočtových jadier [25]. Preto je warp size rovná 32. Je teda vhodné, aby počet vlákien v rámci jednej work group bol deliteľný 32 pre efektívne využitie plánovača inštrukcií. Počet aktívnych vlákien v rámci multiprocesora je u GT200 1024, t.j. 32 warpov. Z tejto vlastnosti tiež vyplýva, že OpenCL kód by mal obsahovať čo možno najmenej podmienok, aby nedochádzalo k divergencii vykonávaného kódu medzi jednotlivmi vláknami. V opačnom prípade bude dochádať k neefektívnemu využitiu prostriedkov grafickej karty, nakoľko sa súbežne vykonávajú len tie vlákna, ktorých inštrukčný tok sa zhoduje - niekoľko Shader procesorov v multiprocesore môže v rámci jedného behu ostať nevyužitých.

Aritmetický výkon pri výpočtoch s jednoduchou presnosťou dosahuje 933 GFLOPS (GTX 285 1,062 TFLOPS). Nakoľko jednotky, ktoré vykonávajú výpočty s dvojnásobnou presnosťou, sú 12-krát pomalšie, pri plnej záťaži s výpočtami s dvojitou presnosťou dosahuje čip výkon len 80 GFLOPS. Štyri atomické jednotky sú navrhnuté pre atomické operácie nad pamäťou, pričom obchádzajú vyrovnávacie pamäte na čipe, poskytujúc tak granulárny prístup do pamäte. Tieto jednotky sú využité, pokiaľ v OpenCL voláme atomické funkcie pre prácu s pamäťou [31].

Pri vývoji tejto práce bola použitá karta GeForce GTX 275, ktorá sa oproti jadru z GTX 285 líší menším počtom Render Output jednotiek, užšou pamäťovou zbernicou (448 oproti 512 bitov) a tým pádom aj menším množstvom grafickej pamäte (896 MiB oproti 1024 MiB u GTX 285). Počet Shader procesorov je však zhodný - 240. Teoretický výkon GeForce GTX 275 je 1,010 TFLOPS [12].

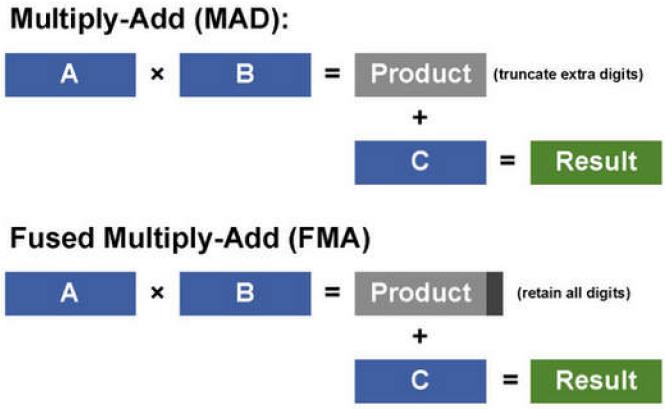
3.2.2 GeForce GTX 480

GeForce GTX radu 400 je prvou generáciou grafických kariet nVidia s podporou DirectX verzie 11, uvedená v marci 2010. Oproti predchádzajúcej generácii popísanej vyššie stúpol počet tranzistorov na 3 miliardy, čo z grafického jadra GF100 robí v dobe písania práce najkomplexnejší grafický čip kedy vyrobený. Z hľadiska GPGPU sa jedná o výrazný posun vpred, nakoľko GF100 je na rozdiel od ostatných grafických čipov MIMD procesor - dokáže vykonávať viacero programov súbežne [19].

Oproti kartám GeForce GTX radu 200 došlo k zmene rozloženia výpočtových jednotiek. Počet multiprocesorov klesol z 30 na 15 (pôvodne ich malo byť až 16), počet výpočtových jednotiek vzrástol na 480, pričom pod jeden multiprocesor ich spadá až 32, viď obrázok v prílohe A.1. Multiprocesory sú organizované do štyroch klastrov (GPC, Graphics Porcessing Cluster) po 4. Vzrástli tiež veľkosti vyrovnávacích pamätí - L1 zo 16 na 48KiB, L2 z 256 na 768KiB, veľmi výrazným prínosom je však pridaná vyrovnávacia pamäť pri prístupe do globálnej pamäte. Taktiež došlo k zdvojnásobeniu počtu registrov, ktorých je na jednom multiprocesore až 32768, zvýšil sa počet rozpracovaných vlákien na jednom multiprocesore z 1024 na 1536 (48 warpov) [17].

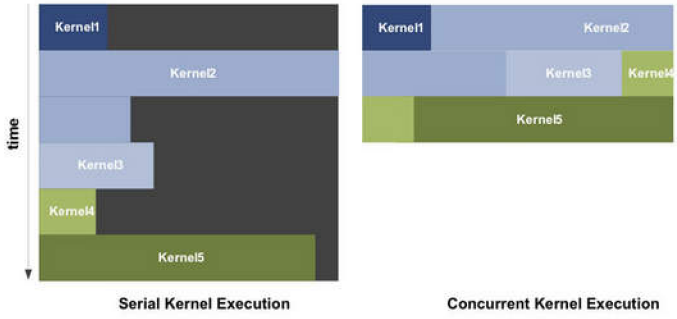
Najvýznamnejšími prínostmi sú výrazne zvýšený výkon pri počítaní s dvojnásobnou presnosťou a možnosť vykonávať na každom multiprocesore iný kernel. U predchádzajúcej generácie obstarávala výpočty s dvojnásobnou presnosťou dedikovaná jednotka, pričom každý multiprocesor mal len jednu. Prepad výkonu bol v takomto prípade až 12-násobný oproti výpočtom s jednoduchou presnosťou. U GF100 boli tieto jednotky odstránené, výpočty s FP64 totiž obstarávajú priamo jednotlivé výpočtové jadrá v dvoch prechodoch, nakoľko sú

32-bitové. Tým pádom sa výkon prepadne len o polovicu. Pribudla tiež podpora pre štandard IEEE 754-2008, s čím súvisí požiadavky na zaokrúhľovanie algoritmy, ako aj inštrukcie FMA (Fused Multiply Add), pri ktorej dochádza len k jednému zaokrúhľovaniu výsledku (viď obr. 3.8), na rozdiel od dvoch pri doteraz používanej inštrukcii MAD (Multiply-Add), čo má za následok zvýšenie presnosti [17] [2].



Obrázok 3.8: Porovnanie inštrukcií MAD a FMA, prevzaté z [17]

U doterajších grafických kariet nVidie (vrátane konkurencie) bolo možné súčasne spustiť len jeden kernel (funkcia vykonávaná na GPU). V prípade rozsiahlych funkcií nedochádzalo k plytvaniu prostriedkov grafickej karty, ale u menej náročných funkcií mohla ostať časť výpočtových jednotiek nevyužitých. Čip GF100 umožňuje súbežné vykonávanie niekoľkých kernelov, na každom multiprocesore môže bežať iný. Prepínanie úloh na procesore je obvykle časovo náročné, na GF100 trvá prepnutie úlohy len 25 mikrosekúnd [17].



Obrázok 3.9: Porovnanie sériového a paralelného vykonávania kernelov, prevzaté z [17]

Jednou z noviniek je tiež skoková predikcia. GF100 paralelne rozpracuje obe varianty možného kódu ešte pred samotným vyhodnotením podmienky, pričom pri vyhodnotení podmienky sa určí, ktorá z variánt sa zahodí.

GeForce GTX 480 disponuje 384-bitovou pamäťovou zbernicou, ku ktorej je pripojených 1536 MiB GDDR5, pričom dáta z nej sú prenášané rýchlosťou 3696 MT/s. Celková pamäťová priepustnosť činí 177,4 GiB/s. Teoretický výkon vo výpočtoch s jednoduchou presnosťou je

1,344 TFLOPS [12].

3.3 Grafické karty AMD Radeon

Prvé grafické čipy od AMD (niekdajšia ATi), ktoré boli navrhnuté s ohľadom na GPGPU výpočty, sú karty súčasnej (v dobe písania) generácie HD 5000. Architektúrou vychádza z rady HD 2900, avšak ani tá generácia, ani HD 3000 OpenCL nepodporujú. Séria HD 4000 sú prvé karty AMD, ktoré podporu majú, avšak u dochádza u nich k výkonnostným prepadom, nakoľko neboli pre použitie s OpenCL navrhované [4]. Rada Radeon HD 5000 je priamym konkurentom kariet GeForce GTX 400.

3.3.1 Radeon HD 5870

Radeon HD 5870 je najvýkonnejšou jednočipovou kartou prvej generácie DirectX11 kompatibilných grafických kariet AMD, uvedená bola v septembri 2009. Jej architektúra sa však od dôb HD 2900 konceptuálne veľmi nezmenila.

Obrázok 3.10: AMD Radeon HD 5870, prevzaté z techmagnews.com

Architektúra čipu RV870, ktorý je základom HD 5870, je do značnej miery odlišná od čipov GeForce. Obsahuje 1600 aritmeticko-logických jednotiek (ALU, marketingovo označované ako Stream procesory). Tie sú zoskupené po piatich do blokov, pričom jeden SIMD procesor pozostáva zo 16-tich takýchto blokov (teda 80 ALU jednotiek na SIMD procesor). Týchto procesorov je na čipe 20, rozdelených po 10 z každej strany zbernice, ako ilustruje obrázok v prílohe A.2. Tento koncept používa AMD už od HD 2900. Jednotlivé ALU bloky (Stream Cores [16]) dostávajú inštrukcie v podobe VLIW (Very Long Instruction Word, veľmi dlhé inštrukčné slovo), ktoré môže obsahovať 1 až 5 64-bitových skalárnych inštrukcií a najviac dve 64-bitové konštanty. ALU jednotky v rámci bloku však nie sú rovnocenné - štyri z nich sú 32-bitové, piata je 40-bitová, ktorá okrem základných operácii nad celými

a desatinnými číslami (FP32 aj FP64, vrátane FMA inštrukcie) podporuje transcendentálne funkcie[4] , ktoré vykonáva v jednom takte. Medzi ne patria napr. goniometrické funkcie či výpočet logaritmu. Každému bloku prislúcha 1024 128-bitových registrov usporiadaných ako matica 4x256. Výpočty s dvojnásobnou presnosťou prebiehajú tak, že si jednotlivé polovice čísel prerozdelia dve ALU jednotky, pričom jedna počíta s vyššími bitmi, druhá s nižšími. Veľkosť lokálnej zdieľanej pamäte SIMD procesora je 32KiB, L1 cache má 8KiB.

Vlákna v rámci jednej pracovnej skupiny (work group) sú na SIMD procesore vykonávané ako niekoľko za sebou idúcich behov - wavefronts. Počet vlákien v jednej wavefront je rovný 64, preto pre optimálne využitie prostriedkov grafickej karty AMD je vhodné nastaviť veľkosť work group na číslo deliteľné 64 [15].

Teoretický aritmeický výkon Radeonu HD 5870 je až 2,7 TFLOPS [11].

[4]Transcendentálne funkcie sú matematické funkcie, ktoré sa nedajú presne vyjadriť konečnou sekvenciou základných algebraických operácií ako sčítanie a násobenie

Kapitola 4

Steinberg VST 2.4

VST (Virtual Studio Technology) je voľne dostupné rozhranie od firmy Steinberg, pomocou ktorého je možné vytvárať moduly (pluginy) do VST-kompatibilných programov pre spracovanie zvuku. Tieto moduly môžeme rozdeliť na niekoľko skupín:

- Inštrumenty - pluginy generujúce zvuk, napr. virtuálne syntetizátory
- Efekty - spracovávajú zvukové dáta, ktoré dostanú na vstup
- MIDI efekty - spracovávajú MIDI správy

Plugin obstaráva úpravu vstupných dát podľa implementovaného algoritmu a parametrov získaných z editora - grafického rozhrania. Hostiteľký program, tzv. VST Host, zabezpečuje načítanie pluginov a smerovanie dát medzi nimi. Podpora VST je v súčasnosti veľmi rozšírená, či už medzi komplexnými nástrojmi ako Ableton Live, Cubase, FruityLoops, alebo menšími programami ako Audacity. Existuje tiež množstvo tzv. mini-hostov, čo sú programy pre podporu ladenia a testovania pluginov.

Samotný vývojový kit pre verziu VST 2.4 predstavuje sadu tried napísaných v C++, pričom okrem tried pre popis samotného pluginu existujú triedy pre popis užívateľského rozhrania pluginu, tzv. editora - VST GUI. V nich nájdeme triedy pre popis tlačidiel, posuvníkov či iných ovládacích prvkov. Zaujímavosťou VST je, že všetky dáta aj parametre sú normalizované do intervalu $< 0 \ldots 1 >$.

Knižnice VST SDK sú kopatibilné so systémami Microsoft Windows a Mac OS [5].

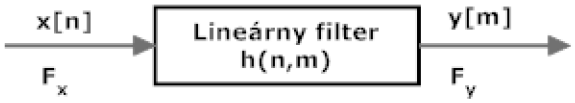
Obrázok 4.1: Príklad pokročilého VST pluginu - virtuálny syntetizátor Kiesel Helga, prevzaté z kiesel.is

Kapitola 5

Prevzorkovanie digitálneho signálu

V rozličných odvetviach spracovania digitálnych signálov sa stretávame s problematikou zmeny vzorkovacej frekvencie signálu. Či už sa jedná o telekomunikačné systémy, kde sa pracuje s rôznymi typmi signálov v rôznych kódovaniach, alebo pri úprave signálov, pokiaľ sa chceme vyhnúť aliasingu (najmä nadvzorkovaním). Informácie v tejto kapitole boli voľne prevzaté z [28].

Proces prevzorkovania môže byť demonštrovaný ako lineárna filtrácia, pri ktorej do systému s prenosovou funkciou $h(n, m)$ vstupuje signál $x[n]$ charakterizovaný vzorkovacou frekvenciou F_x, pričom výstupom je signál $y[m]$ so vzorkovacou frekvenciou F_y.



Obrázok 5.1: Prevzorkovanie ako lineárny filter

Pomer vzorkovacích frekvencií

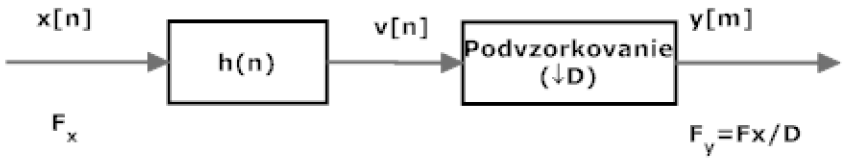
$$\frac{F_x}{F_y} = \frac{I}{D} \tag{5.1}$$

je racionálny podiel, pričom v ideálnom prípad sú I (interpolačný faktor) a D (decimačný faktor) prvočísla.

Prevzorkovanie teda pozostáva z dvoch procesov - interpolácie (nadvzorkovanie) o faktor I a z decimácie faktorom D (podvzorkovanie).

5.1 Decimácia

Pokiaľ by sme so vstupným signálom $x[n]$ so spektrom $X(\omega)$ pustili na výstup len každý jeho D-tý vzorok, výsledný signál by bol aliasovaný okolo frekvencie $\frac{F_x}{2D}$. Aby sme sa vyhli aliasingu, je nutné zúžiť frekvenčné pásmo na maximálnu frekvenciu $F_{max} = \frac{F_x}{2D}$.



Obrázok 5.2: Proces decimácie

Ideálny spodnofrekvenčný filter eliminujúci frekvenčné pásmo v rozsahu $\frac{\pi}{D} < \omega < \pi$ by mal mať frekvenčnú charakteristiku $H_D(\omega)$ spĺňajúcu podmienku

$$H_D(\omega) = \begin{cases} 1, & |\omega| \leq \frac{\pi}{D} \\ 0 \end{cases} \tag{5.2}$$

pričom výstupom filtra bude sekvencia $v[n]$ (viď obr. 5.2) daná predpisom

$$v[k] = \sum_{k=0}^{\infty} h(k)x(n-k) \tag{5.3}$$

Pre výstup $y[m]$ platí:

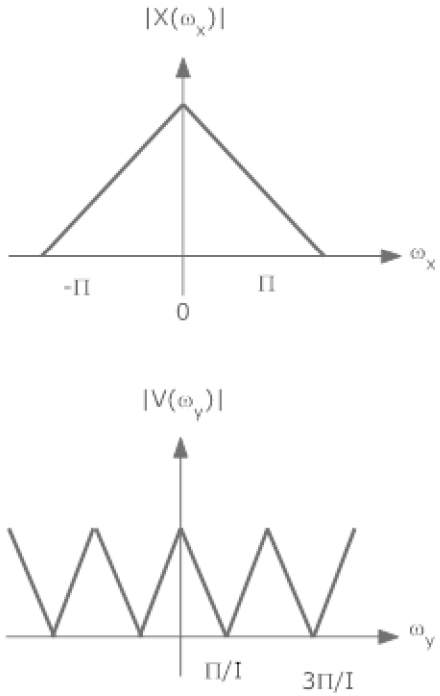
$$\begin{aligned} y[m] &= v(mD) && (5.4) \\ &= \sum_{k=0}^{\infty} h(k)x(mD-k) && (5.5) \end{aligned}$$

5.2 Interpolácia

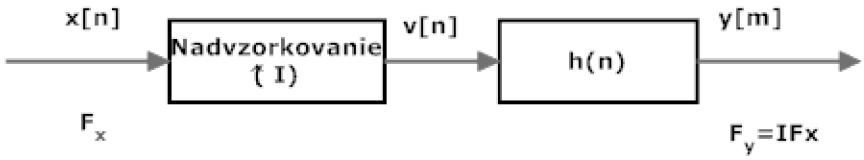
Zvyšovanie vzorkovacej frekvencie vstupného signálu faktorom I dosiahneme interpoláciou o $I-1$ nových vzorkov medzi dvoma susednými vzorkami. Ak pomenujeme takto získanú postupnosť ako $v[n]$, jej spektrum získame úpravou Z-transformácie:

$$\begin{aligned} V(z) &= \sum_{m=-\infty}^{\infty} v(m)z^{-m} && (5.6) \\ &= \sum_{m=-\infty}^{\infty} x(m)z^{-mI} && (5.7) \\ &= X(z^I) && (5.8) \\ V(\omega_y) &= X(\omega_y I) && (5.9) \end{aligned}$$

kde ω_y v rovnici 5.9 je kruhová frekvencia vo vzťahu s výslednou vzorkovacou frekvenciou F_y, pričom platí $\omega_y = 2\pi F/F_y$. Ak porovnáme spektrum pôvodného signálu $X(\omega_x)$ s výsledným $V(\omega_y)$ (obr 5.3) zistíme, že sa vo výsledku periodicky opakujú obrazy spektra $X(\omega_x)$.



Obrázok 5.3: Spektrum vstupného signálu a po interpolácii bez filtrácie



Obrázok 5.4: Schéma interpolátora

Nakoľko len časť spektra v rozmedzí $0 \leq \omega_y \leq \pi/I$ je unikátna, frekvencie nad π/I sa odstránia prefiltrovaním cez spodnofrekvenčný filter s impulznou odozvou:

$$H_I(\omega_y) = \begin{cases} I, & |\omega_y| \leq \frac{\pi}{I} \\ 0 \end{cases} \tag{5.10}$$

Výsledná prefiltrovaná sekvencia $y[m]$ bude mať predpis podľa vzťahu 5.11:

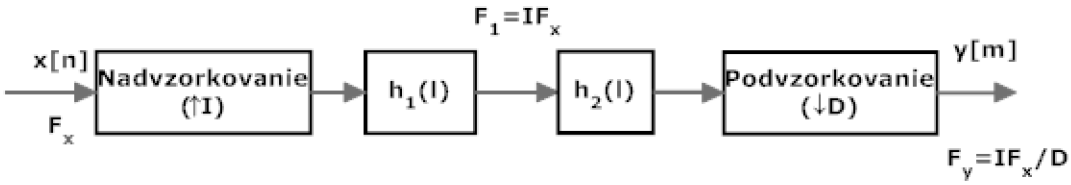
$$y[m] = \sum_{k=-\infty}^{\infty} h(m-k)v(k) \tag{5.11}$$

Nakoľko sa v neprefiltrovanej sekvencii $v(k)$ poväčšine nachádzajú nuly a len každý I-ty vzorok je nenulový, je možné zápis $y[m]$ upraviť na:

$$y[m] = \sum_{k=-\infty}^{\infty} h(m-kI)x(k) \tag{5.12}$$

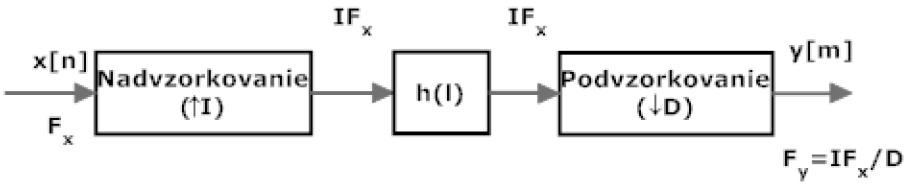
5.3 Prevzorkovanie racionálnym podielom I/D

Pokiaľ chceme zmeniť vzorkovaciu frekvenciu vstupného signálu pomerom I/D, môžeme to dosiahnuť najprv jeho interpolovaním, následne decimáciou. V praxi to znamená spojiť decimátor a interpolátor, pričom prvotná schéma vyzerá ako na obr. 5.5



Obrázok 5.5: Spojenie interpolátora a decimátora

V tejto konfigurácii sú za sebou zaradené 2 filtre $h_1(l)$ a $h_2(l)$, ktoré fungujú na rovnakej vzorkovacej frekvencii IF_x. Tým pádom môžu byť skombinované do jedného nízkofrekvenčného filtra $h(l)$, pričom jeho vlstnosti musia byť kombináciou oboch filtrov.



Obrázok 5.6: Spojenie interpolátora a decimátora s jedným filtrom

Aby bol výsledný filter schopný potlačiť aliasing ako u interpolácie tak u decimácie, musí prepúšťať maximálne najnižšiu spomedzi frekvencií π/I a π/D. Postačuje teda jeden z filtrov, ktorý prepúšťa menšiu maximálnu frekvenciu [29].

Po týchto úpravách je možné vyjadriť výstupný signál $y[m]$ ako:

$$y[m] = \sum_{n=-\infty}^{\infty} h(nI + (mD \bmod I))x\left(\left\lfloor \frac{mD}{I} \right\rfloor - n\right) \tag{5.13}$$

5.4 Zefektívnenie interpolácie a decimácie

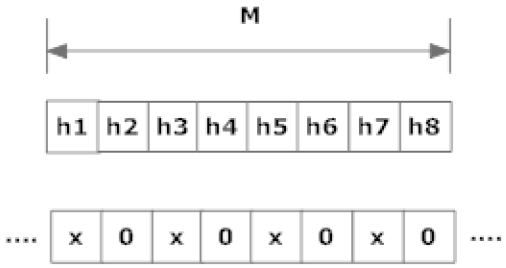
Oba spomenuté algoritmy sú v ich priamej forme neefektívne. V prípade interpolátora sa do vstupného signálu vkladá I-1 núl medzi dva susedné vzorky, následne je prefiltrovaný. Počas filtrácie je však v dôsledku prítomnosti množstva núl na vstupe väčšina výpočtov zbytočná, nakoľko ich výsledkom je nula a v konečnom dôsledku len spomaľujú konvolúciu s koeficientami filtra.

Naproti tomu decimátor zachováva len jeden z D výstupov filtra, predišlo by sa tak zbytočným výpočtom, ktoré sa aj tak na výstupe zahodia.

Riešením oboch problémov je zabudavať interpolátor aj decimátor priamo do štruktúry filtra a rozložiť filter na sadu polyfáz.

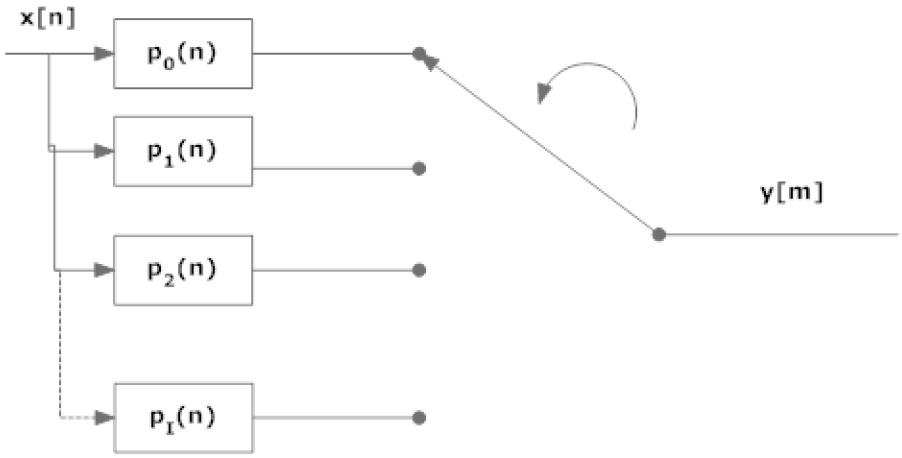
5.5 Polyfázová dekompozícia

Princípom polyfázového usporiadania FIR filtra je zmenšenie dĺžky M FIR filtra na skupinu kratších filtrov - polyfáz. Každá z polyfázových zložiek filtra bude mať dĺžku $K = \frac{M}{I}$, preto je vhodné zvoliť dĺžku filtra tak, aby bola deliteľná interpolačným faktorom, alebo ho doplniť na konci nulami.



Obrázok 5.7: Interpolácia, I=2, M=8, potom K=4. x - nenulový vzorok

Keďže pri interpolácii sa medzi každé 2 po sebe idúce vzorky vloží $I - 1$ núl, iba K z M vstupov bude nenulových. Preto budú prenásobené koeficientmi $h(p)$, $h(p + I)$, $h(p + 2I)$ $\ldots h(p + M - I)$, pričom p je číslo aktuálnej polyfázovej zložky. Ilustratívne sa dá vyjadriť ako komutátor otáčajúci sa proti smeru hodinových ručičiek (obr 5.8).

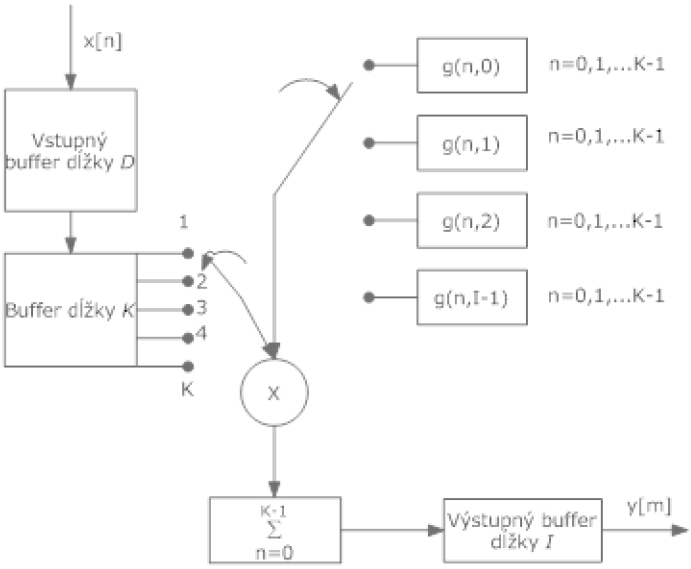


Obrázok 5.8: Interpolátor ako komutátor

Pre samostatný decimátor pláti rovnaká schéma, ale zrkadlovo otočená (prepínanie na výstupe).

5.6 Časovo-premenlivá polyfázová štruktúra filtra

Na obrázku 5.9 vidno výslednú optimalizovanú štruktúru polyfázového prevzorkovacieho filtra.



Obrázok 5.9: Schéma pre polyfázový systém prevzorkovania racionálnym pomerom I/D

Systém na uvedenom diagrame spracúva bloky vzorkov o dĺžke K koeficienatami filtra (jednou polyfázou) taktiež s dĺžkou K, pričom takýchto sád koeficientov je I. Pre každý blok I výstupných vzorkov existuje D vzorkov na vstupe. Týchto D vstupných vzorkov je postupne posúvaných do druhého buffera, vždy po jednom. Posunutie sa uskutoční len ak sa zaokrúhlený podiel $\lfloor \frac{mD}{I} \rfloor$ zvýši o jedna. Ako na tomto vzorci vidno, čím väčší interpolačný faktor, tým menej často sa bude buffer posúvať, naopak, čím väčší decimačný faktor, tým rýchlejšie. Pre každý výstup $y[l]$ sa vynásobia vzorky v druhom bufferi s odpovedajúcou sadou koeficientov $g(n,l)$ pre $n = 0, 1, ...K-1$ a výsledky sa posčítajú. $y[m]$ sa vo všoebecnosti vypočíta podľa vzťahu 5.14

$$y[m] = \sum_{n=0}^{K-1} g\left(n, m - \left\lfloor \frac{m}{I} \right\rfloor I\right) x\left(\left\lfloor \frac{mD}{I} \right\rfloor - n\right) \tag{5.14}$$

kde $g(n,l)$ predstavuje polyfázové rozdelenie filtra $h(k)$ na I polyfáz o dĺžke K.

Kapitola 6

Návrh a implementácia

Cieľom práce bolo vytvoriť v praxi použiteľnú triedu (knižnicu) akcelerujúcu prevzorkovanie prostrednícvtom grafickej karty. Ako jazyk implementácie bol zvolený C++ pre jednoduchšiu integráciu so Steinberg VST 2.4 API pre výsledný plugin.

6.1 Návrh

Aby bola výsledná trieda použiteľná v praxi, bolo treba si položiť požiadavky, ktoré by mala spĺňať.

Nakoľko v programoch spracovávajúce zvuk (Ableton Live!, Steinberg Cubase a pod.) je dĺžka buffera so vstupnými vzorkami meniteľná počas behu, preto je nutné, aby sa objekt vzniknutej triedy vedel s týmto vysporiadať. Problém je najmä v grafickej karte, kde je potreba pri zmene dĺžky vstupu odpovedajúci buffer na karte správne preinicializovať. Podobne je to aj so zmenou filtra.

Samotný algoritmus prevzorkovania je obvykle demonštrovaný pre sériové vykonávanie na CPU. Pre potreby OpenCL je nutné ho transformovať do SIMD paradigma - pozerať sa na problém nie zvonku ako celok, ale "zvnútra"- ako je možné ho rozdrobiť na malé ale rovnaké kúsky?

Pre potreby testovania oproti procesoru je tiež nutné urobiť implementáciu prevzorkovania pre procesor.

Výsledná triedu sa na záver implementuje do VST pluginu.

6.2 Implementácia

Vývoj prebihal na platforme Microsoft Windows 7 (32-bit) v spojení s Visual Studiom 2008. Ako grafická karta bola počas vývoja použitá GeForce GTX 275 s ovládačmi verzie 197.45, spoločne s vývojovým kitom CUDA Toolkit 3.0.

6.2.1 GPU implementácia

Aby bola OpenCL implementácia dostatočne výkonná, je nutné využívať prostriedky grafickej karty čo najefektívnejšie. Nakoľko na GPU môže súčasne bežať veľké množstcvo vlákien, bolo treba zistiť, ktorá časť systému by sa dala namodelovať ako výsledok každého z vlákien. Do úvahy pripadalo:

- jednotlivá polyfáza

- výstupný vzorok

Pokiaľ by sme uvažovali o prístupe "čo vlákno, to polyfáza", pre každú z nich by sa na základe dĺžky vstupu a pomeru I/D zistilo, ktoré výstupné prvky by pripadli tomuto vláknu (resp. číslu subfiltra). Pre každý z nich by sa musel určiť počiatočný index na vstupe, od ktorého by prebiehala konvolúcia s filtrom. Takýto spôsob je však pre grafickú kartu nevýhodná, nakoľko počet polyfázových zložiek odpovedá interpolačnému faktoru, ktorý sa nezvykne pohybovať v tak vysokých číslach. Pre viacvláknovú implementáciu na procesore by však tento postup bol výhodnejší.

Vo výslednej implementácii každé z vlákien na grafickej karte počíta jeden výstupný vzorok. Zo vzorca 6.1

$$y[m] = \sum_{n=0}^{K-1} g\left(n, m - \left\lfloor \frac{m}{I} \right\rfloor I\right) x\left(\left\lfloor \frac{mD}{I} \right\rfloor - n\right) \tag{6.1}$$

bolo možné odvodiť, že pre každý výstupný index m sa použije polyfáza podľa vzorca 6.2:

$$p = (m \times D) \, mod \, I \tag{6.2}$$

a index, od ktorého prebehne konvolúcia s filtrom podľa vzťahu 6.3 [29]:

$$n = \left\lfloor \frac{mD}{I} \right\rfloor \tag{6.3}$$

Pre kartu bude dokonca výhodnejšie, ak veľkosť vstupu bude veľká, o to viac sa totiž prejaví zrýchlenie oproti procesoru.

Na hostiteľskom systéme bolo treba platformu inicializovať. Počas vývoja programu došlo k zmene v špecifikácii OpenCL ohľadom výberu platformy. Doposiaľ nebolo možné mať viacero OpenCL implementácii na jednom systéme, avšak so zavedením OpenCL ICD (Installable Client Driver) si uživateľ (programátor) môže vybrať, na ktorej z dostupných platforiem chce vykonávať výpočty [8]. Funkcia `clCreateContextFromType()` už neakceptuje `NULL` ako paramter pre popis platformy, nakoľko výber už nie je automatický. Konštruktor triedy `cl_resampler` teda spomedzi dostupných platforiem vyhľadá tú, v ktorej sa nachádza grafická karta, a nad ňou vytvorí kontext. V rámci kontextu tiež vytvorí frontu príkazov, načíta a skompiluje OpenCL program, a vytvorí z neho kernel.

Funkciou `setRatioFilterCoeff()` sa nastaví prevzorkovací pomer, odkaz na výstupný buffer (musí byť predalokovaný) ako aj pole s filtrom a počtom jeho koeficientov. Funkcia na grafickej karte vytvorí buffer pre koeficienty filtra a buffer, do ktorého budú presúvané staršie vzorky (inicializovaný nulami).

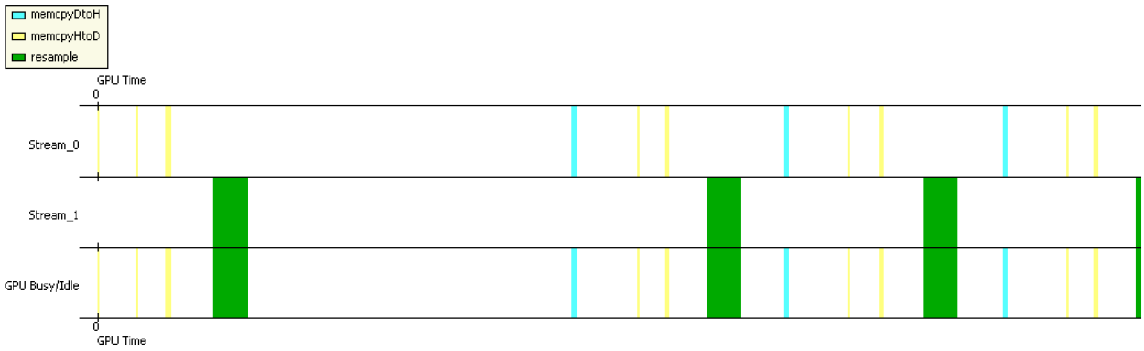
Obrázok 6.1: Proces filtrácie

V tejto fáze sa tiež koeficienty prekopírujú do grafickej karty, spolu s bufferom so staršími vzorkami. Pokiaľ je funkcia opätovne zavolaná už po zavolaní funkcie `resample()`, zmení

veľkosť buffera so vstupnými vzorkami a nahrá do GPU nový filter. Ostatné buffery označí ako nevytvorené - v prípade zmeny prevzorkovacieho pomeru.

O ich vytvorenie sa stará funkcia `resample()` - pri prvom volaní vytvorí podľa dĺžky vstupu adekvátne miesto na grafickej karte, a na základe dĺžky vstupu, interpolačného a decimačného faktoru určí dĺžku výstupu. Nastaví tiež parametre pre spustenie kernelu. Potom dôjde k nakopírovaniu dát do grafickej karty a spustenie samotného výpočtu. Po prevzatí výsledku sa vzorky z konca vstupu prekopírujú na na začiatok buffera so staršími samplami, následne sa toto pole natiahne do GPU. Pri volaní funkcie `resample` sa vždy kontroluje, či nedošlo k zmene dĺžky vstupu, pokiaľ áno, vstupný a výstupný buffer sa zahodí a vytvorí nanovo.

Prvé volanie `resample()` trvá dlhšie ako ďalšie nasledujúce. Je to z dôvodu, že prebieha vytváranie pamätí na GPU. Rozdiel ukazuje profiler na obr. 6.2.



Obrázok 6.2: Funkcia `resample` pri prvom a ďalšom volaní, zelená - bežiaci kernel, modrá - výber výsledku z GPU, žltá - nahratie dát do GPU

6.2.2 CPU implementácia

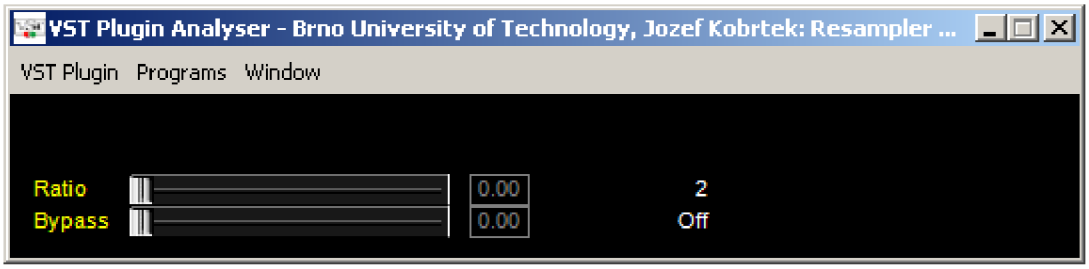
Ako základ pre CPU implementáciu som zvolil knižnicu dodávanú spolu s programom ScopeFIR (http://www.iowegian.com/scopefir). V rámci CPU implementácie boli naprogramované rovnaké funkcie, ako u triedy GPU implementácie.

6.3 Integrácia s VST 2.4

Pri integrácii triedy GPU implementácie a rozhrania VST som použil šablónu pre tvorbu pluginov od pána Ing. Jiřího Schimmela (Audiffex). Táto šablóna zjednodušuje prácu pri tvorbe pluginov s rozhraním VST 2.4.

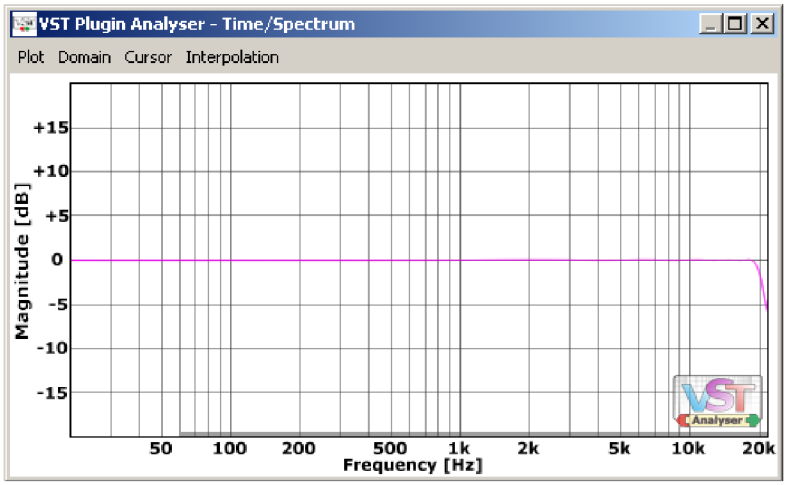
Trieda `CVSTTemplate` dedí po triede `AudioEffectX`, čo je základná trieda pre popis VST pluginu - efektu. Okrem implementácie viruálnych metód pre nastavovanie a zobrazovanie parametrov obsahuje metódu `processReplacing`, v ktorej prebieha samotný proces úpravy signálu. Pre zjednodušenie práce s parametrami, ktoré sú normalizované do intervalu $< 0 \ldots 1 >$, sú v rámci šablóny implementované funkcie `getRealValue` a `setRealValue`, ktoré umožňujú programátorovi prácu s denormalizovanými hodnotami parametrov, ktorých interval bol špecifikovaný pri definícii parametrov.

Vo funkcii `processReplacing` sú použité 2 objekty typu `cl_resampler`, pričom jeden z nich funguje ako upsampler (vykonáva nadvzorkovanie o *Ratio*), druhý ako downsam-



Obrázok 6.3: Užívateľské rozhranie implementovaného pluginu, ktorý bol spustený v programe VST Plugin Analyser

pler (podvzorkuje o *Ratio*). Najprv idú vstupné dáta upsampleru, následne putuje jeho výstup do downsamplera. Sada filtrov pre oba objekty boli vygenerované programom Matlab funkciou `resample`, ktorá okrem výstupu vracia tiež filter, ktorým prevzorkovanie vykonala.



Obrázok 6.4: Frekvenčná charakteristika pluginu

Plugin pre demonštračné účely obsahuje 2 parametre - faktor prevzorkovania, a bypass. Tie sú prostredníctvom rozhrania nastaviteľné pomocou posuvníkov, viď obr. 6.3. Zapnutie bypassu spôsobí, že prevzorkovanie nie je aktívne, dôjde len k prekopírovaniu vstupu na výstup. Nakoľko bola použitá sada filtrov pre každý jeden nad a podvzorkovací pomer, zmenou tohto pomeru sa frekvenčná charakteristika od tej na obr. 6.4 nemení.

Výsledkom je dynamicky linkovateľná knižnica DLL, ktorá sa za behu načíta vo VST hostiteľskom programe. Testovanie knižnice prebiehalalo v mini-hoste VST Plugin Analyser.

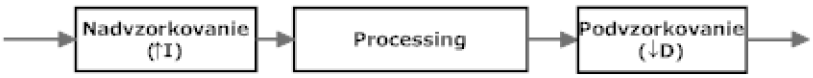
Kapitola 7

Testovanie výkonu

Výkon GPU implementácie som sa snažil zmerať s nastaveniami, ktoré sa bežne používajú pri real-time spracovaní zvuku. Pre ilustráciu potenciálu grafických kariet som tiež otestoval, aký bol ich prínos aj s parametrami mimo štandardných hraníc.

Prevzorkovanie sa bežne používa v súvislosti s odstránením aliasingu pri určitej úprave signálu. Nakoľko táto úprava by mohla spôsobiť aliasing, je signál najprv nadvzorkovaný, na to sa signál spracuje, na záver sa opäť podvzorkuje.

Dĺžka vstupného buffera sa pri spracovaní zvuku v reálnom čase pohybuje v rozmedzí 64-256 vzorkov, teda pri vzorkovacej frekvencii 44KHz to odpovedá časovému úseku 1,5 až 5,8 miliseknudy. Interpolačný a decimačný faktor obvykle nadobúdajú hodnôty 2-9, špeciálny prípad je prevzorkovanie medzi 44,1 a 48 KHz, kedy je tento faktor rovný 160/147. Jedným z testovaných parametrov bol aj vplyv dĺžky filtra na výkonnosť.



Obrázok 7.1: Spracovanie signálu s využitím resampleru

Pri testovaní sa vytvorí vstupný buffer a filter, ktoré sa náhodne nainicializujú dátami. S parametrami pre konkrétny test (interpolácia, decimácia, dĺžka filtra) sa opakuje prevzorkovanie 1000-krát pričom sa meria čas, koľko trvalo prevzorkovanie týchto tisíc iterácii. Tento postup sa pre každý test opakuje 10-krát, aby sa na grafickej karte postupne vytvorilo niekoľko kontextov, časy dosiahnuté v jednotlivých behoch sa sčítajú. Výledok testu je primerná hodnota prevzorkovania jednej sady dát (t.j. jedného vstupného buffera), výsledky testov sú zobrazené v milisekundách. Čas bol meraný s využitám funkcie Windows API `QueryPerformanceCounter`, ktorá poskytuje presnosť merania času 10^{-6} sekundy.

Počas návrhu testov bolo zistené, že rozdiel medzi dvoma behmi rovnakého testu je na grafickej karte väčší než u procesora. Je to spôsobené pravdepodobne tým, že PCI Express je zbernica postavená na prepínaní dátových packetov, pričom na PCI-e sú okrem iného pripojené aj periférie ako SATA radič či zvukový kodek. Tento poznatok bol použitý pri určovaní počtu opakovaní v každom z testov.

7.1 Použitý hardware

Testy prebiehali na nasledovnej konfigurácii:

- CPU: Intel Core 2 Duo E6550 2,23GHz
- Základná doska: MSI P7N SLi Platinum
- RAM: 4GiB DDR2 800MHz, dual-channel zapojenie
- Pevný disk: Western Digital 400GB 7200RPM
- Zdroj: 650W
- Operačný systém: Windows 7 Professional, 32-bit
- Ovládače: GeForce 197.45

Použité grafické karty:

- GeForce 9600GT
- GeForce 8800GTX
- GeForce GTX 275

Vlastnosti použitých grafických katiet získané programom GPU Caps Viewer. Na otestovanie sa bohužiaľ nepodarilo zapožičať grafickú kartu od AMD.

Názov karty	Výpočtových jednotiek	Počet multiprocesorov	Jednotiek na multiprocesor	Pamäť (MiB)	Takt (MHz)
GeForce 9600GT	64	8	8	512	1625
GeForce 8800GTX	128	16	8	768	1350
GeForce GTX 275	240	30	8	896	1404

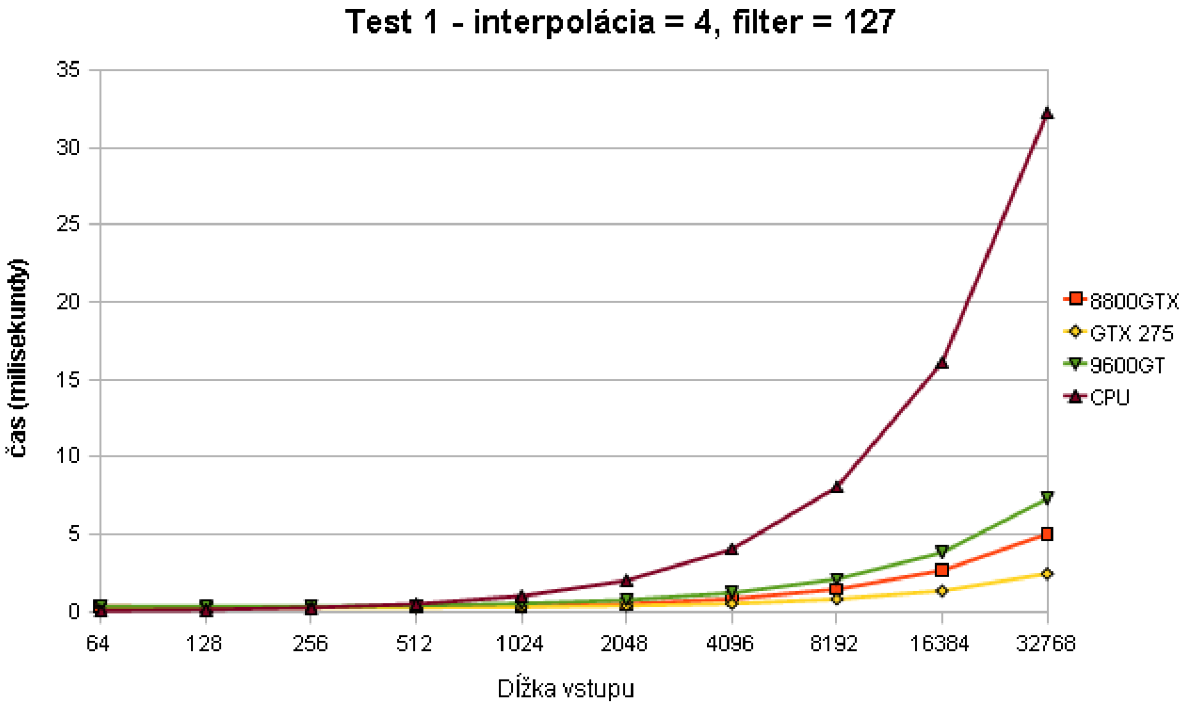
Tabuľka 7.1: Vlastnosti použitých grafických kariet

7.2 Test 1 - zväčšovanie dĺžky vstupu

Pri tomto teste sa zisťoval vplyv dĺžky vstupu na výkon. Dĺžku filtra bola nastavená na 127, interpolácia (a neskôr decimácia) na 4, čo odpovedá prevzorkovaniu signálu zo 44 KHz na 192 KHz.

Dĺžka vstupu	8800GTX	GTX 275	9600GT	CPU
64	0,343	0,308	0,347	0,064
128	0,287	0,262	0,324	0,128
256	0,290	0,265	0,333	0,252
512	0,309	0,271	0,403	0,503
1024	0,371	0,304	0,537	1,008
2048	0,518	0,395	0,766	2,014
4096	0,825	0,524	1,211	4,028
8192	1,449	0,817	2,103	8,051
16384	2,661	1,344	3,819	16,103
32768	5,027	2,453	7,297	32,236

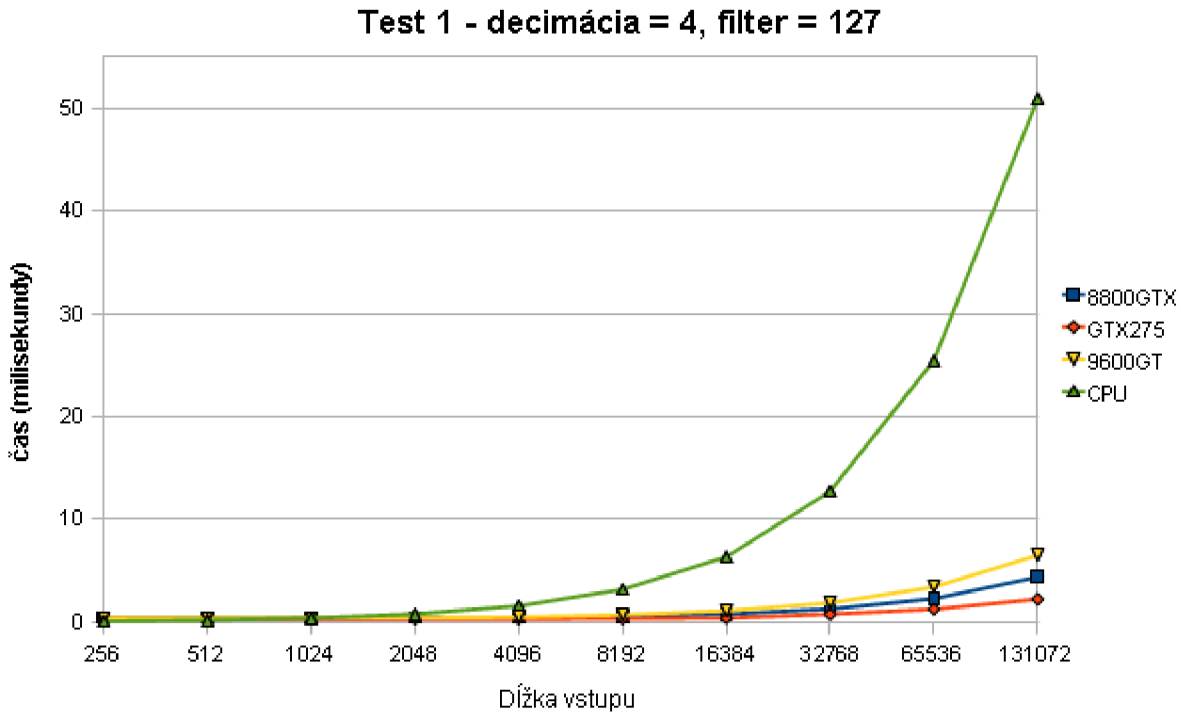
Tabuľka 7.2: Test 1 - interpolácia



Obrázok 7.2: Test 1 - interpolácia

Dĺžka vstupu	8800GTX	GTX 275	9600GT	CPU
256	0,443	0,332	0,448	0,100
512	0,419	0,294	0,435	0,199
1024	0,422	0,307	0,445	0,398
2048	0,435	0,329	0,451	0,796
4096	0,443	0,335	0,504	1,591
8192	0,513	0,357	0,707	3,185
16384	0,784	0,418	1,114	6,366
32768	1,280	0,773	1,886	12,742
65536	2,300	1,262	3,433	25,479
131072	4,386	2,246	6,511	50,926

Tabuľka 7.3: Test 1 - decimácia



Obrázok 7.3: Test 1 - decimácia

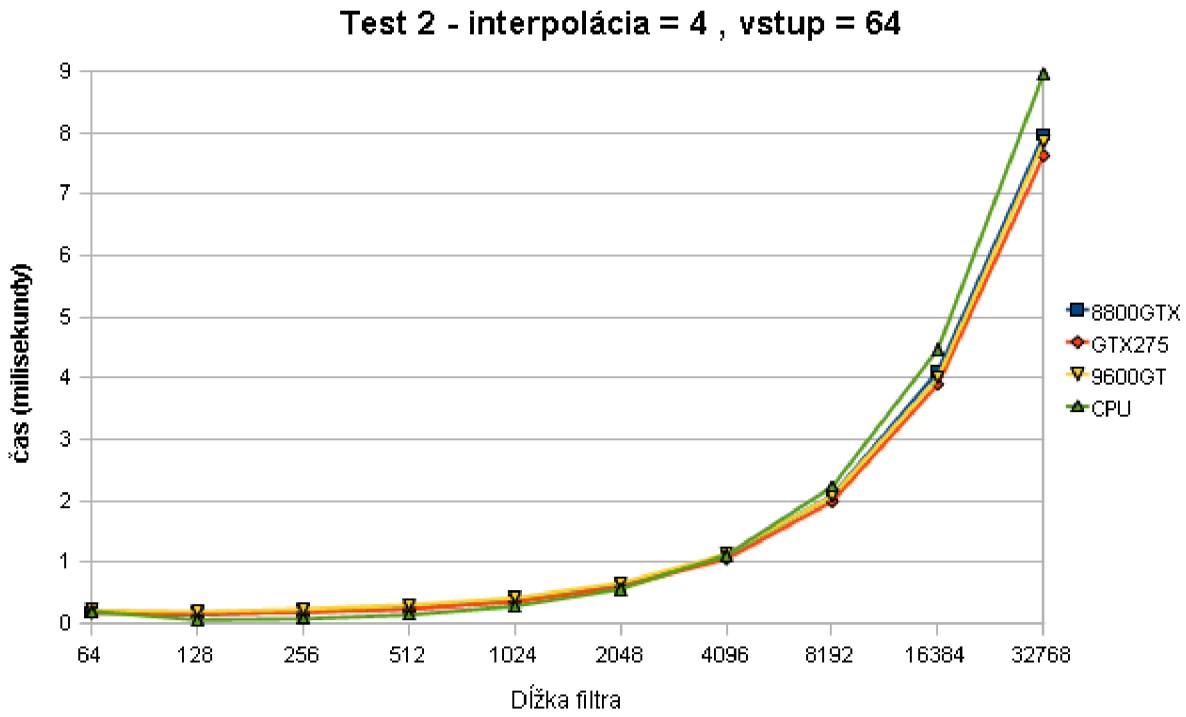
Pri interpolácii aj decimácii faktorom 4 jasne dominujú grafické karty nad procesorom. Zrýchlenie sa však prejavuje až od dlhších vstupov (512), do dĺžky 256 vzorkov dominuje pri interpolácii procesor. U decimácie je situácia rovnaká - pri 2048 vstupných vzorkoch (t.j. 512 výstupných) nastáva prevaha grafických kariet nad procesorom. Ak by sme predlžovali vstup, zrýchlenie oproti procesoru by rástlo, pri 32768 vstupných vzorkoch u interpolácie je až 13-násobné u GTX 275. Takáto dĺžka vstupu je však pre real-time spracovanie zvuku neprípustná.

7.3 Test 2 - zväčšovanie dĺžky filtra, najmenší bežný vstup

V tomto teste bol vstup rovný 64 vzorkov, prevzorkovací faktor 4 a dĺžka filtra rástla rástla dvojnásobne od hotnoty 64. Pri decimácii bol nastavený vstup na 256 vzorkov (64×4).

Dĺžka filtra	8800GTX	GTX 275	9600GT	CPU
64	0,172	0,169	0,216	0,197
128	0,148	0,156	0,195	0,051
256	0,187	0,183	0,230	0,070
512	0,241	0,235	0,296	0,138
1024	0,364	0,349	0,413	0,278
2048	0,621	0,593	0,658	0,559
4096	1,107	1,050	1,129	1,113
8192	2,083	1,986	2,076	2,228
16384	4,110	3,903	4,013	4,475
32768	7,972	7,636	7,863	8,966

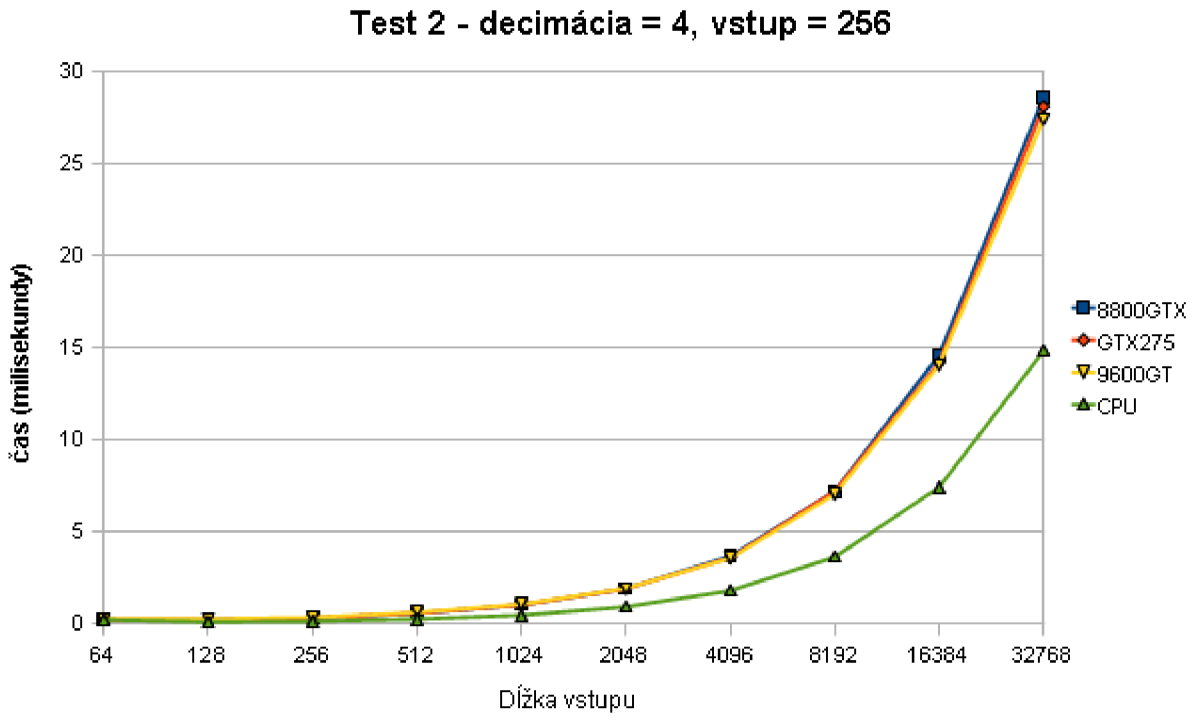
Tabuľka 7.4: Test 2 - interpolácia



Obrázok 7.4: Test 2 - interpolácia

Dĺžka filtra	8800GTX	GTX 275	9600GT	CPU
64	0,185	0,184	0,247	0,185
128	0,195	0,192	0,241	0,072
256	0,269	0,269	0,319	0,112
512	0,540	0,552	0,592	0,223
1024	0,992	0,997	1,032	0,439
2048	1,880	1,880	1,899	0,892
4096	3,650	3,567	3,542	1,773
8192	7,187	7,180	7,015	3,606
16384	14,597	14,193	14,063	7,406
32768	28,627	28,115	27,442	14,825

Tabuľka 7.5: Test 2 - decimácia



Obrázok 7.5: Test 2 - decimácia

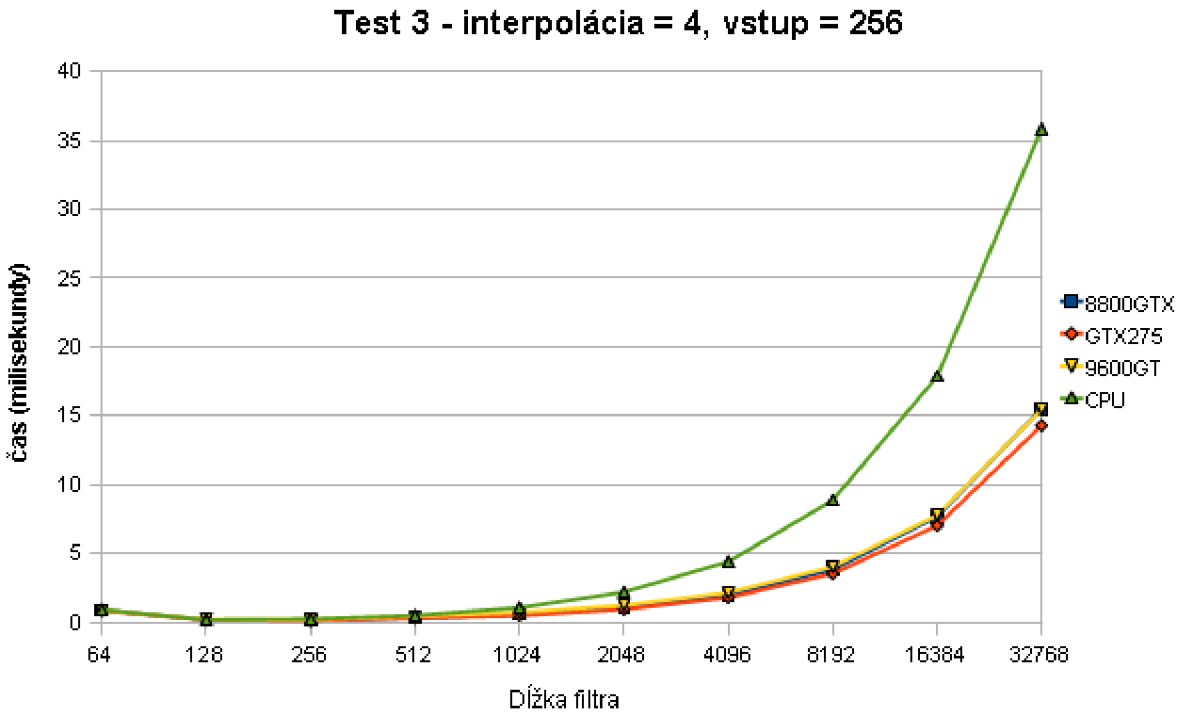
Príliš malá dĺžka vstupu sa podpísala na výkone grafických kariet v teste. Pri interpolácii sa karty ešte držali nad výkonom procesora, ale až od 4096 koeficientov. Decimácia dopadla z pohľadu grafických kariet ešte horšie, CPU výkonnostne dominovalo pri každej dĺžke filtra.

7.4 Test 3 - zväčšovanie dĺžky filtra, najväčší bežný vstup

V tomto teste bol vstup natiahnutý na najväčšiu dĺžku, ktorá sa pri spravovaní real-time audia používa - 256 vzorkov (1024 pri decimácii). Nastavenia ostali zhodné s predošlým testom.

Dĺžka filtra	8800GTX	GTX 275	9600GT	CPU
64	0,930	0,892	0,937	0,957
128	0,233	0,213	0,265	0,222
256	0,196	0,166	0,239	0,280
512	0,352	0,335	0,493	0,548
1024	0,615	0,539	0,770	1,106
2048	1,106	0,987	1,276	2,234
4096	1,996	1,841	2,209	4,448
8192	3,857	3,567	4,075	8,898
16384	7,705	7,057	7,808	17,891
32768	15,532	14,326	15,468	35,857

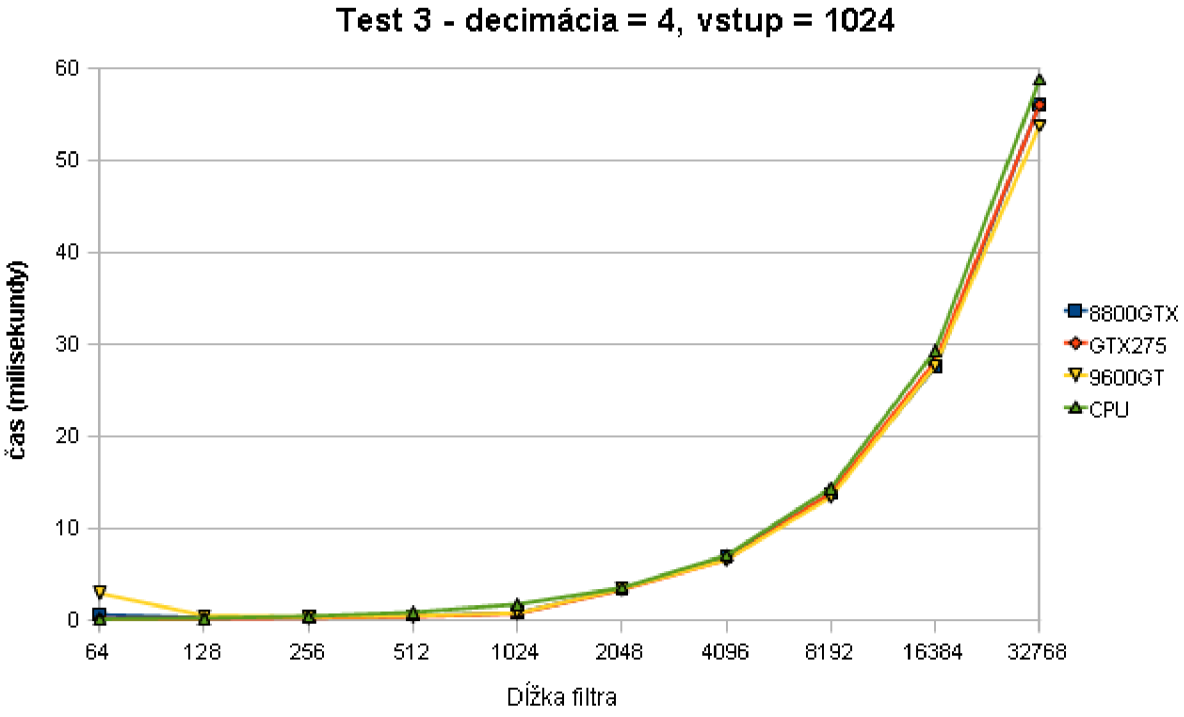
Tabuľka 7.6: Test 3 - interpolácia



Obrázok 7.6: Test 3 - interpolácia

Dĺžka filtra	8800GTX	GTX 275	9600GT	CPU
64	0,605	0,154	2,930	0,106
128	0,271	0,171	0,517	0,219
256	0,318	0,248	0,359	0,434
512	0,472	0,401	0,483	0,876
1024	0,790	0,715	0,785	1,737
2048	3,405	3,324	3,429	3,520
4096	6,854	6,587	6,591	7,055
8192	13,760	13,873	13,401	14,380
16384	27,591	28,152	27,711	29,355
32768	56,181	56,095	53,850	58,803

Tabuľka 7.7: Test 3 - decimácia



Obrázok 7.7: Test 3 - decimácia

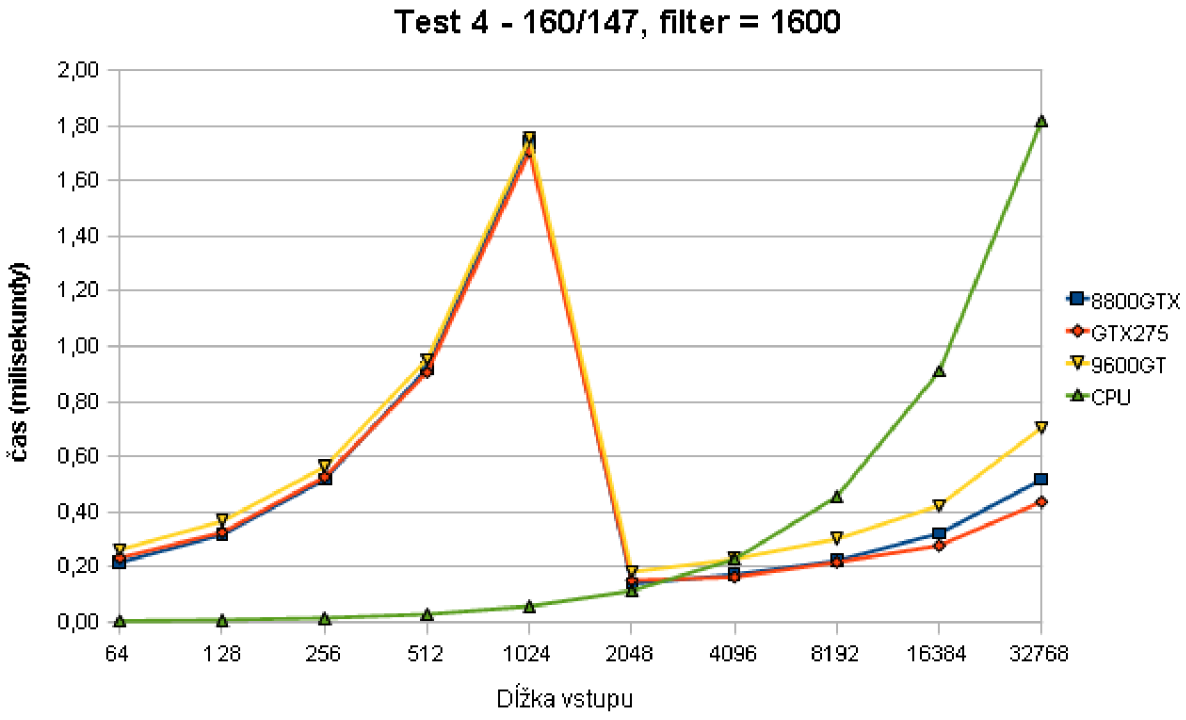
Pri interpolácii bolo dosiahnuté 2-násobné zrýchlenie pri filtri o dĺžke 1024 vzorkov (GTX 275), celkovo boli grafické karty výkonnejšie, až na dva prípady. U decimácie je situácia odlišná - nakoľko sa používa pri filtrácii viac koeficientov (v poli koeficientov sa posúva vždy o interpolačný faktor, ktorý je u decimácie rovný 1), čoho výsledkom je aj častejší prístup do globálnej pamäte grafickej karty, ktorá nie je cache-ovaná. Najväčší rozdiel bol dosiahnutý opäť pri filtri dĺžky 1024, s rastúcou dĺžkou vstupu však odstup procesora a grafických kariet nie je veľmi veľký.

7.5 Test 4 - špeciálny prípad prevzorkovania medzi 44,1 KHz a 48 KHz

Ako bolo spomenuté vyššie, jedným zo špeciálnych prípadov prevzorkovania signálu je prevod medzi vzorkovacou frekvenciou 44,1 KHz a 48 KHz. Vtedy sú totiž oba faktory vysoké - 160 a 147. Testoval som s filtrom o dĺžke 1600 (resp. 1470 pri podvzorkovani), pričom sa menila dĺžka vstupu.

Dĺžka vstupu	8800GTX	GTX 275	9600GT	CPU
64	0,216	0,233	0,263	0,004
128	0,317	0,326	0,366	0,007
256	0,518	0,527	0,563	0,014
512	0,920	0,906	0,954	0,029
1024	1,739	1,707	1,756	0,057
2048	0,140	0,152	0,181	0,114
4096	0,172	0,163	0,229	0,227
8192	0,222	0,217	0,301	0,455
16384	0,321	0,278	0,423	0,909
32768	0,517	0,436	0,705	1,817

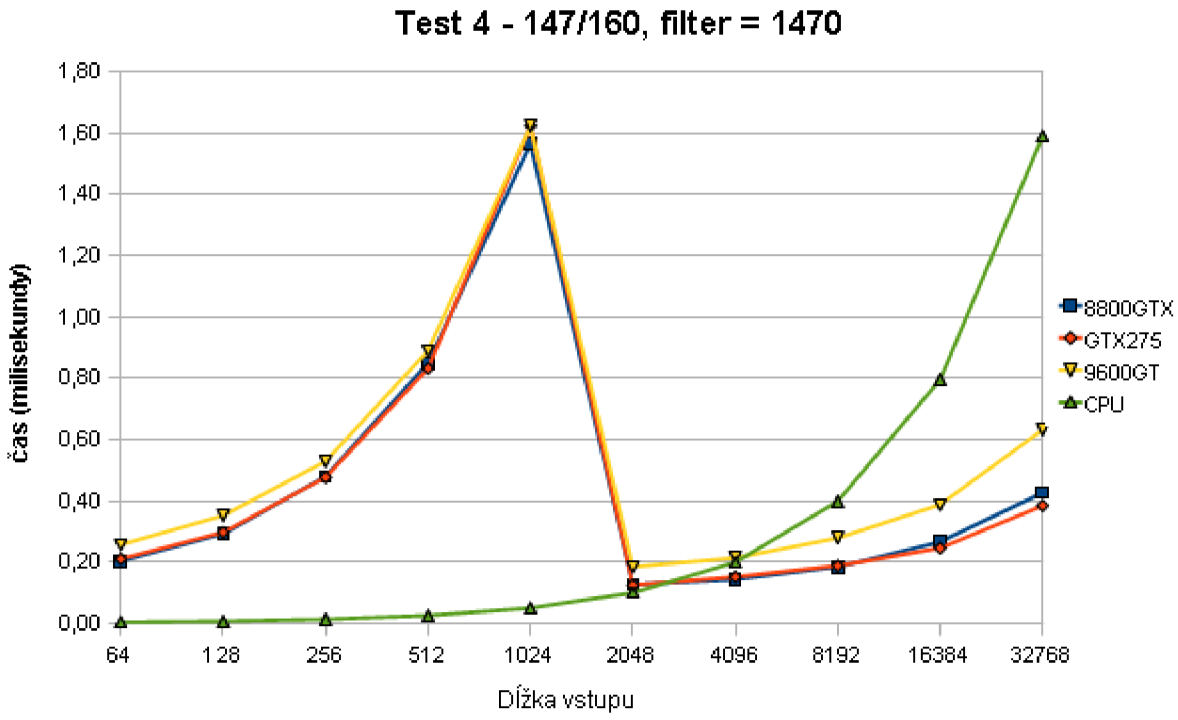
Tabuľka 7.8: Test 4 - interpolácia



Obrázok 7.8: Test 4 - interpolácia pomerom 160/147

Dĺžka vstupu	8800GTX	GTX 275	9600GT	CPU
64	0,203	0,210	0,256	0,003
128	0,292	0,297	0,352	0,006
256	0,479	0,477	0,529	0,013
512	0,849	0,832	0,890	0,025
1024	1,566	1,624	1,624	0,050
2048	0,126	0,125	0,184	0,099
4096	0,143	0,151	0,213	0,199
8192	0,183	0,187	0,278	0,398
16384	0,266	0,245	0,388	0,796
32768	0,426	0,384	0,631	1,591

Tabuľka 7.9: Test 4 - decimácia



Obrázok 7.9: Test 4 - decimácia pomerom 147/160

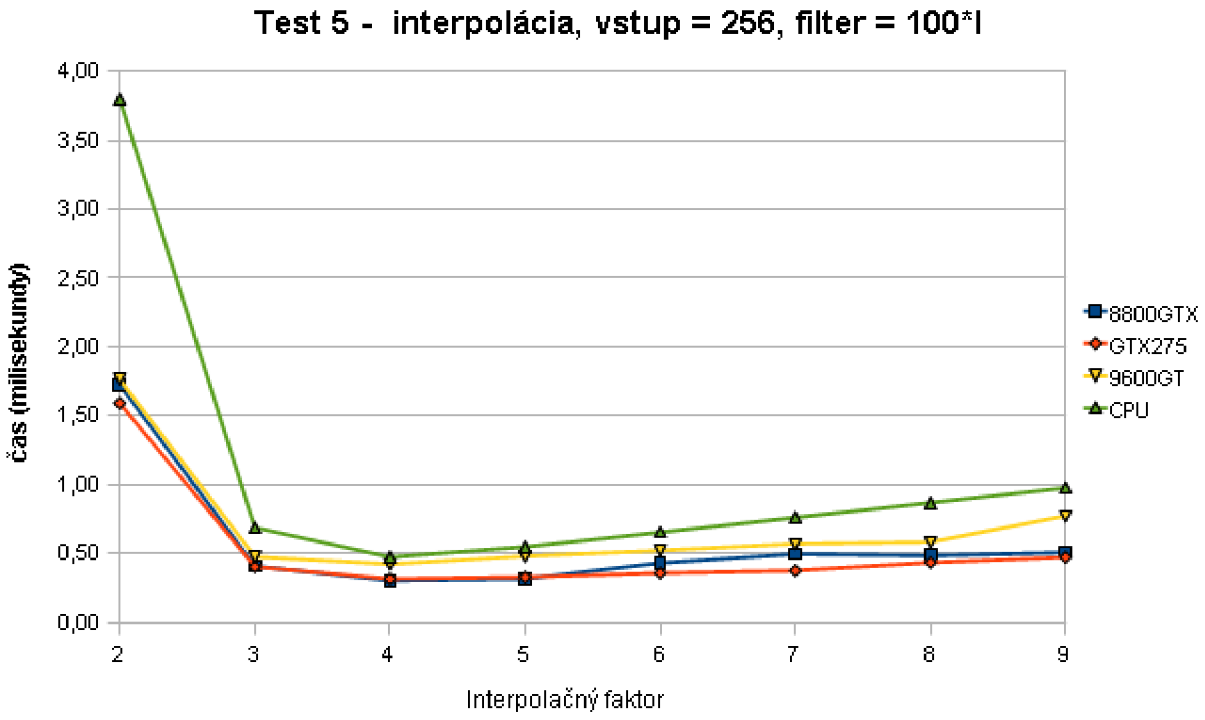
Nakoľko počet vlákien na GPU kvôli nízkemu pomeru rástol pomaly, výhoda oproti CPU sa prejavila až od 4096. Na grafoch 7.8 a 7.9 vidno 10-násobné zrýchlenie medzi dĺžkami 1024 a 2048 (pri ganulárnejšej analýze okolo 1500). Spôsobil ho pravdepodobne veľmi chaotický prístup do pamäte (skoky po 160, resp. 147), s narastajúcim počtom vlákien ho pravdepodobne vyrovnalo viacero prísputpov na miesta s podobnou lokáciou, ktoré bolo možné obslúžiť efektívnejšie - pri prístupe do globálnej pamäte sa totiž prečíta väčší kus, ako je požiadavka.

7.6 Test 5 - Závislosť prevzorkovacieho faktoru na výkone

Úlohou tohto testu bolo zistiť vplyv prevzorkovacieho faktoru na výkon. Dĺžka filtra bola vypočítaná ako 100-násobok faktora prevzorkovania, čo je v praxi viac než postačujúce pre daný faktor. Vstupný buffer mal dĺžku 256 vzorkov v oboch prípadoch.

Upsampling	8800GTX	GTX 275	9600GT	CPU
2	1,726	1,593	1,765	3,799
3	0,410	0,407	0,478	0,690
4	0,306	0,318	0,425	0,476
5	0,319	0,331	0,483	0,551
6	0,434	0,360	0,524	0,656
7	0,499	0,379	0,572	0,763
8	0,487	0,437	0,585	0,871
9	0,510	0,475	0,775	0,979

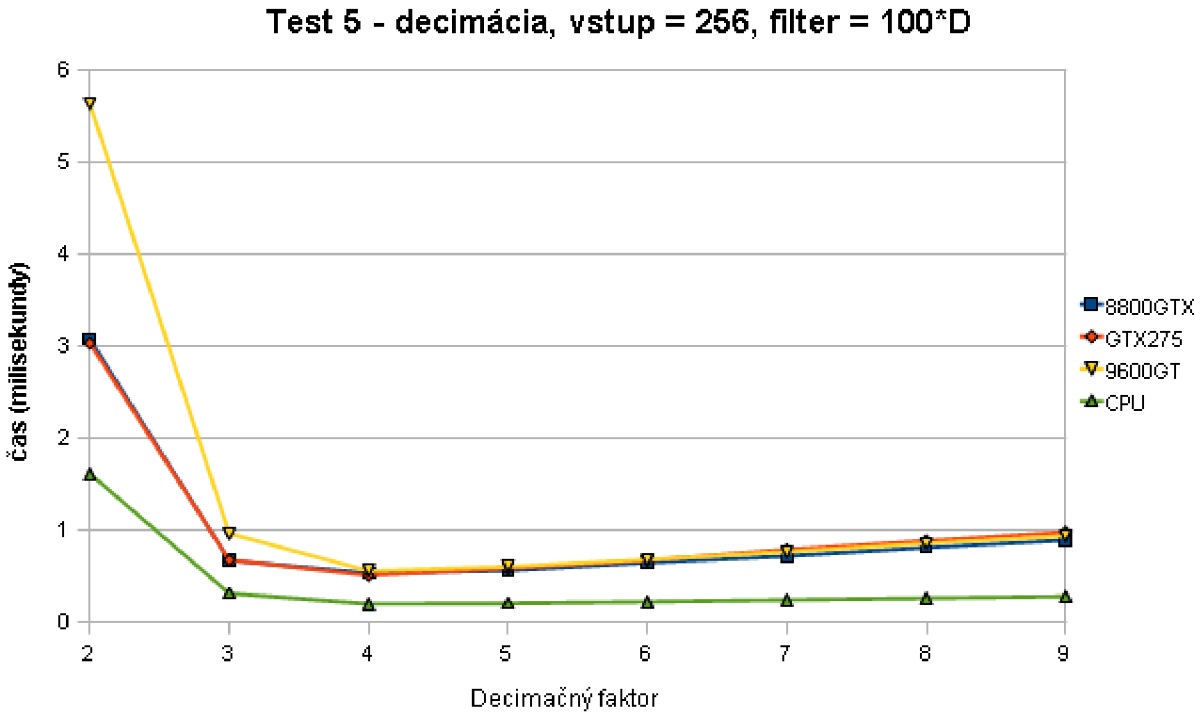
Tabuľka 7.10: Test 5 - interpolácia



Obrázok 7.10: Test 5 - interpolácia

Downsampling	8800GTX	GTX 275	9600GT	CPU
2	3,078	3,035	5,643	1,613
3	0,668	0,673	0,963	0,310
4	0,530	0,506	0,554	0,196
5	0,559	0,583	0,600	0,202
6	0,644	0,671	0,678	0,219
7	0,718	0,786	0,763	0,240
8	0,809	0,883	0,856	0,258
9	0,891	0,975	0,937	0,276

Tabuľka 7.11: Test 5 - decimácia



Obrázok 7.11: Test 5 - decimácia

Testy interpolácie kopírujú trend z testov 2 a 3. Nárast výkonu medzi grafickými kartami a procesorom vidno, ale nie tak výrazný ako v iných testoch. V prípade decimácie sa kvôli malému počtu vlákien (64) výkon grafických kariet prepadol pod procesor. Brzdou v takomto prípade je tiež presun dát do GPU, ktorý sa pri malom počte vlákien na GPU nedá dostatočne vykryť, aby grafická karta predčila procesor.

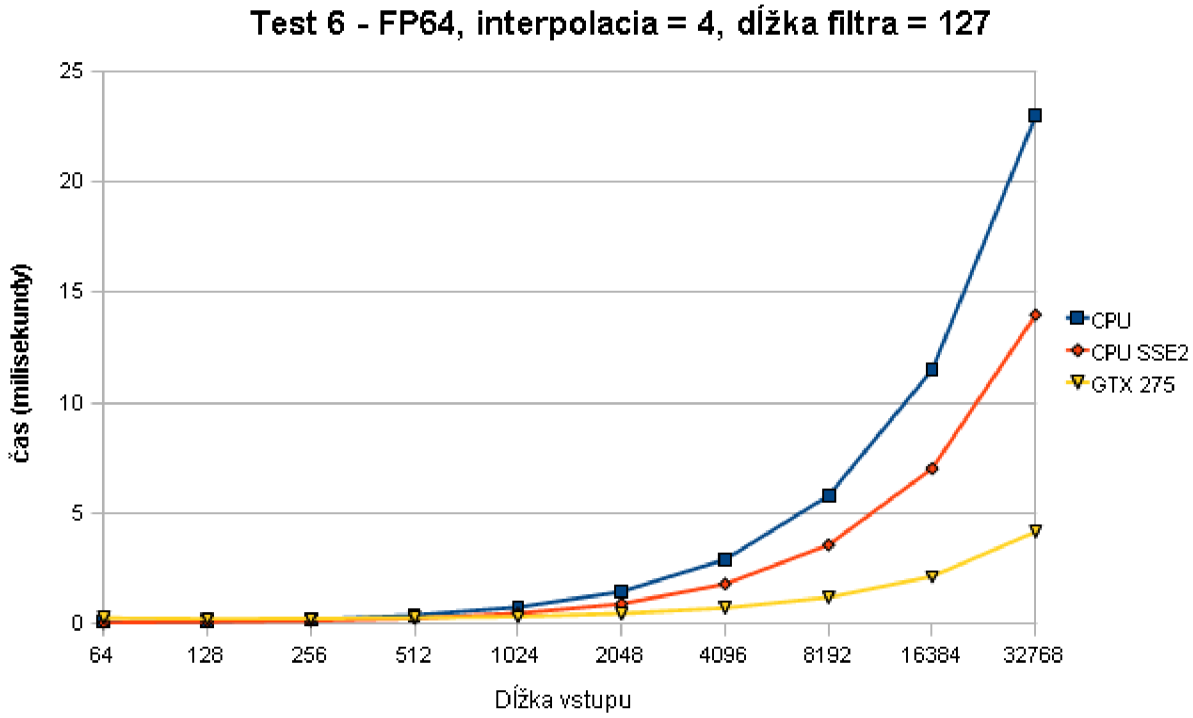
7.7 Test 6 - Dvojnásobná presnosť

Nakoľko som mal k dispozícii kartu podporujúcu výpočty s dvojitou presnosťou, prepísal som CPU aj GPU implementáciu prevzorkovania na dátový typ double. Zvolil som metodiku ako v prvom teste - filter o dĺžke 127, interpolácia o 4. Pre porovnanie som zvolil aj CPU

test skompilovaný s podporou SSE2, ktorý priniesol nemalé zvýšenie výkonu pre procesor.

Dĺžka vstupu	CPU	CPU SSE2	GTX 275
64	0,046	0,029	0,246
128	0,091	0,062	0,207
256	0,180	0,111	0,219
512	0,360	0,221	0,250
1024	0,726	0,445	0,304
2048	1,434	0,878	0,440
4096	2,894	1,781	0,690
8192	5,792	3,561	1,171
16384	11,497	7,019	2,123
32768	23,017	13,990	4,156

Tabuľka 7.12: Test 6



Obrázok 7.12: Test 6

SSE2 pridalo CPU implementácii na výkone, v poslednej iterácii až 65%. Stále to však nestačí na grafickú kartu. Tá si na rozdiel od použitia dátového typu `float` pohoršila. FPU jednotka procesora však natívne pracuje s typom `double`, preto je `float` len úsporou pamäte, nie však výkonnostne, nakoľko musí byť do `double` tak či tak prevedený. Preto je výkon procesora vo výpočtoch s dvojitou presnosťou väčší [1].

Kapitola 8

Záver

Cieľom práce bolo zistiť možnosti využitia grafických kariet a rozhrania OpenCL pre použitie pri spracovaní zvuku v reálnom čase.

Algoritmus prevzorkovania signálu nepatrí medzi najzložitejšie algoritmy používané pri spracovaní zvuku, bez ohľadu na to však grafické karty dokázali prekonať výkon procesora. Pre relatívne malé veľkosti vstupných bufferov je treba cenu za transport dát do grafickej karty kompenzovať napr. zväčšovaním dĺžky filtra na vysoké hodnoty, aby sa výkon dostal pred procesor. Príčinou nižšieho výkonu pri kratších vstupoch je okrem ceny za presun dát do pamäte tiež nižšie vyťaženie grafickej karty, nakoľko sa vytvorí menej vlákien. Problémom je tiež nezarovnaný prístup do pamäte, keď si jednotlivé vlákna pýtajú dáta z rôznych častí globálnej pamäte, ktorá u testovaných grafických kariet nemá vyrovnávaciu pamäť, teda pri požiadavke na dáta sa do hlavnej pamäte grafickej karty pristupuje vždy. Tento problém by sa mal minimalizovať s novou generáciu grafických kariet, počínajúc GeForce GTX radu 400, ktorá už cache pri globálnej pamäti má. Ostrániť tento problém v tejto implementácii sa nepodarilo, nakoľko prostriedky vynaložené na dodatočnú réžiu pamäte sa vzhľadom na zložitosť algoritmu nevyplatili.

Pri omptimalizácii CPU implementácie sa ukázal ako dobrý krok prechod na dvojitú presnosť a využiť podporu SSE2 inštrukčnej sady, pričom nárast výkonu sa pohyboval na úrovni 60%. Výpočty s dvojnásobnou presnosťou podporovala len karta GeForce GTX 275, pričom jej výkon v porovnaní s jednoduchou presnosťou klesol o polovicu. Stále to však stačilo na prekonanie SSE2-optimalizovanej CPU implementácie, pričom výkonnostný rozdiel sa s narastajúcou dĺžkou vstupu zväčšoval, čo odráža výhody extrémnej paralelizácie moderných grafických čipov.

V rámci rozšírenia by bolo možné pridať do pluginu aj samotný processing, prípadne pridať podporu pre viacero zvukových kanálov. Ak by bol algoritmus komplikovanejší, zrýchlenie by sa mohlo dosiahnuť aj využitím napr. niekoľkých grafických kariet. Značná časť kódu triedy `cl_resampler` je znovupoužiteľná, preto je možné vymeniť algoritmus počítaný na grafickej karte za iný.

Literatúra

[1] Doubles outperforming floats. [online].
URL
<http://www.gamedev.net/community/forums/topic.asp?topic_id=506100>

[2] Multiply-accumulate: Fused multiply-add. [online].
URL
<http://en.wikipedia.org/wiki/Fused_multiply-add#Fused_multiply-add>

[3] nVidia GPU Computing Webinars: Introduction to GPU Computing with OpenCL. [online], 2009.
URL <http://developer.download.nvidia.com/CUDA/training/NVIDIA_GPU_Computing_Webinars_Introduction_To_OpenCL.pdf>

[4] OpenCL performance issue. [online], 2009.
URL <http://forums.amd.com/devforum/messageview.cfm?catid=390&threadid=123857>

[5] Virtual Studio Technology. [online], 2009-04-05 [cit. 2010-05-14].
URL <http://en.wikipedia.org/wiki/Virtual_Studio_Technology>

[6] BrookGPU. [online], 2009-24-11 [cit. 2010-02-16].
URL <http://en.wikipedia.org/wiki/BrookGPU>

[7] GPU Caps Viewer. [online], 2010.
URL <http://www.ozone3d.net/gpu_caps_viewer/>

[8] KB71 - Updating your OpenCL code to work with the OpenCL ICD. [online], 2010.
URL <http://developer.amd.com/support/KnowledgeBase/Lists/KnowledgeBase/DispForm.aspx?ID=71>

[9] OpenCL. [online], 2010-02-10 [cit. 2010-02-16].
URL <http://en.wikipedia.org/wiki/Opencl>

[10] ATI Stream Software Development Kit: System Requirements. [online], 2010 [cit. 2010-03-11].
URL <http://developer.amd.com/gpu/ATIStreamSDK/Pages/default.aspx>

[11] Comparison of AMD graphics processing units. [online], 2010 [cit. 2010-04-29].
URL <http://en.wikipedia.org/wiki/Comparison_of_AMD_graphics_processing_units>

[12] Comparison of Nvidia graphics processing units. [online], 2010 [cit. 2010-04-29].
URL <http://en.wikipedia.org/wiki/Comparison_of_Nvidia_graphics_processing_units>

[13] Dual-channel architecture. [online], 2010 [cit. 2010-05-03].
URL <http://en.wikipedia.org/wiki/Dual-channel_architecture>

[14] Python::OpenCL v0.2 documentation. [online], [cit. 2010-02-15].
URL <http://python-opencl.next-touch.com/>

[15] Advanced Micro Devices, Inc.: *ATi Stream SDK Performance Notes*. 10 2009, v2.01.
URL <http://developer.amd.com/gpu/ATIStreamSDK/assets/ATI_Stream_SDK_Performance_Notes.pdf>

[16] Advanced Micro Devices, Inc.: *ATI Stream SDK OpenCL Programming Guide*. 04 2010, v1.0c.
URL <http://developer.amd.com/gpu/ATIStreamSDK/assets/ATI_Stream_SDK_OpenCL_Programming_Guide.pdf>

[17] ANGELINI, C.; ABI-CHAHLA, F.; CHARPENTIER, F.: GeForce GTX 480 And 470: From Fermi And GF100 To Actual Cards! [online], 2010.
URL
<http://www.tomshardware.com/reviews/geforce-gtx-480,2585-18.html>

[18] GÖDDEKE, D.: GPGPU::Basic Math Tutorial. [online], 2007 [cit. 2010-02-17].
URL
<http://www.mathematik.uni-dortmund.de/~goeddeke/gpgpu/tutorial.html>

[19] HAGEDOORN, H.: NVIDIA GF100 (Fermi) Technology preview. [online], 2010.
URL <http://www.guru3d.com/article/nvidia-gf100-fermi-technology-preview/6>

[20] HENSLEY, J.: ATI Stream OpenCL Technical Overview Video Series. [online], 2007.
URL <http://developer.amd.com/documentation/videos/OpenCLTechnicalOverviewVideoSeries/Pages/default.aspx>

[21] JEDEC: *JEDEC JESD79-3D, DDR3 SDRAM Standard*. 2009.
URL <http://www.jedec.org/standards-documents/docs/jesd-79-3d>

[22] Khronos OpenCL Working Group: *The OpenCL Specification*. Rev. 48.
URL <http://www.khronos.org/registry/cl/specs/opencl-1.0.48.pdf>

[23] KIRK, D.; HWU, W.-M.: *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 2010, ISBN 978-0-12-381472-2.

[24] KOLEJÁK, M.: Súboj performance riešení: Radeon HD 4870 vs. GeForce GTX 260. [online], 2008.
URL <http://www.pc.sk/modules.php?name=article&what=read&v=1227038699>

[25] LUEBKE, D.: GPU Architecture: GPU Architecture: Implications and Trends. In *SIGGRAPH 2008*, 2008.
URL <http://s08.idav.ucdavis.edu/luebke-nvidia-gpu-architecture.pdf>

[26] MUNSHI, A.: OpenCL: Parallel Computing on the GPU and CPU. In *SIGGRAPH 2008*, 2008.
URL <http://s08.idav.ucdavis.edu/munshi-opencl.pdf>

[27] nVidia Corporation: *nVidia OpenCL Best Practices Guide*. 08 2009, v2.3.
URL <http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_OpenCL_BestPracticesGuide.pdf>

[28] PROAKIS, J. G.; MANOLAKIS, D. G.: *Digital Signal Processing: Principles, Algorithms, and Applications*. Prentice Hall, třetí vydání, 1996, ISBN 0-13-373762-4.

[29] SCHNITER, P.: Polyphase Resampling with a Rational Factor. [online].
URL <http://cnx.org/content/m10443/latest/>

[30] SELAND, J.: CUDA Programming. [online], 2008.
URL <http://heim.ifi.uio.no/~knutm/geilo2008/seland.pdf>

[31] SMALLEY, T.: GT200: Nvidia GeForce GTX 280 analysis. [online], 2008.
URL <http://www.bit-tech.net/hardware/graphics/2008/06/24/nvidia-geforce-gtx-280-architecture-review/4>

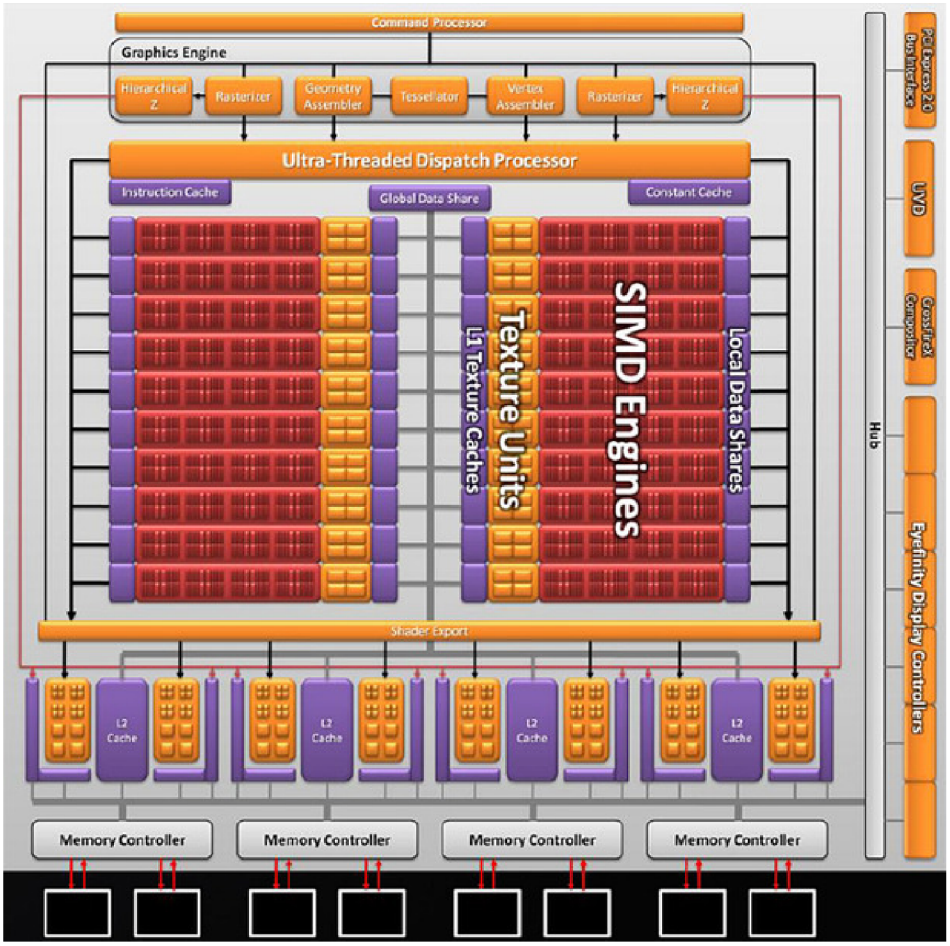
[32] SMALLEY, T.: ATI Radeon HD 5870 Architecture Analysis. [online], 2009.
URL <http://www.bit-tech.net/hardware/graphics/2009/09/30/ati-radeon-hd-5870-architecture-analysis/5>

[33] TREVETT, N.; ZELLER, C.: OpenCL on the GPU. In *GPU Technology Conference*, nVidia Corporation, 09 2009.

Príloha A

Obrázky

Obrázok A.1: Architektúra a hierarchické rozdelenie výpočetných jednotiek grafického čipu GF100 ešte podľa pôvodného návrhu so 16-timi multiprocesormi a 512 výpočtovými jadrami, pre ilustráciu ku kapitole 3.2.2, prevzaté z [17]



Obrázok A.2: Schéma grafického čipu RV870, prevzaté z [32]. SIMD Engines sú výpočtové jednotky. Použité v kapitole 3.3.1, prevzaté z bit-tech.net

Príloha B

Zdrojový kód OpenCL vykonávaný na grafickej karte

```
__kernel void resample( __private int I,
                        __private int D,
                        __private int thread_count,
                        __private int filter_length,
                        __global float * input,
                        __global float * old_input,
                        __global float * output,
                        __constant float * filter,
                        __private int input_length)
{
  int thread_num = get_global_id(0);
  if(thread_num >= thread_count)
    return;

  int start_index = trunc((thread_num*D)/(float)I);
  int filter_index = ((thread_num*D) % I);
  float sum=0;

  while(filter_index<filter_length)
  {

    if(start_index<0)
      sum += old_input[abs(start_index)-1] * filter[filter_index];
    else
      sum += input[start_index] * filter[filter_index];

    filter_index += I;
    --start_index;
  }
  output[thread_num]=sum;
}
```

Príloha C

Použitie knižnice a pluginu

Na priloženom CD sa nachádzajú zdrojové kódy k samotnej knižnici, výkonnostným testom a pluginu vo forme Visual Studio projektov pre jednoduchšiu manipuláciu. Aby bolo možné spustiť OpenCL kód na cieľovom počítači, je nutné vlastniť OpenCL-kompatibilnú grafickú kartu. Medzi podporovanými sú:

- nVidia GeForce radu 8000 a vyššie
- AMD Radeon HD 4000 a vyššie

Grafické karty iných výrobcov (napr. Intel) štadard OpenCL nepodporujú. Je tiež nutné mať nainštalované ovládače, ktoré OpenCL podporujú, čo môže byť problém napr. u notebookových čipov. Funkčnosť či prítomnosť rozhrania OpenCL je možné jednoducho zistiť napr. programom GPU Caps Viewer [7].

V rámci projektu sú priložené súbory `cl.h`, `cl_platform.h` a knižnica `OpenCL.lib`, aby na cieľovom počítači nemusel byť nainštalovaný celý vývojový kit. V prípade zmeny štandardu, napr. doplnenie o nové rozšírenia či odstránenie chýb, je vhodné tieto súbory prepísať aktuálnymi, dostupné sú v SDK nVidie alebo AMD.

CPU implementácia prevzorkovania, `cpu_resampler`, vychádza z knižnice ScopeFIR, pričom som použil identické rozhranie ako na GPU verzii.

Testovací program, nachádzajúci sa v prílohe, obsahuje 10 testov (5 párov), ktorých výsledky boli použité v tejto práci. Test s dvojnásobnou presnosťou vznikol prepísaním dátových typov na double u prvého testu, podobná úprava nastala aj v triedach `cl_resampler` a `cpu_resampler`. Pre automatizáciu testov sú k dispozícii funkcie `gpu_test` a `cpu_test`. Výstupom testovacieho programu je súbor `results.txt`. Celá sada testov trvá približne 45 minút, závisí na použitom hardware.

Pre použitie pluginu je nutné mať VST hostiteľský program, nakoľko sa jedná o DLL knižnicu. Program VST Plugin Analyser, ktorý som tiež použil pri vývoji, je k dispozícii na CD priloženom k práci. Súčasťou jeho zdrojových kódov je aj VST SDK 2.4, nad ktorými je vytvorená šablóna pre tvorbu pluginov od pána Ing. Schimmla. Počas testovania sa ukázali určité problémy so stabilitou pluginu, pričom ladením a testovaním bola vylúčená chyba v algoritme.

Príloha D

Obsah priloženého CD

Obsah priloženého CD je rozdelený do adresárov:

- *bin* - preložená binárka s testovacím programom a VST plugin ako DLL knižnica
- *src* - obsahuje zdrojové kódy jednotlivých knižníc, pluginu a testovacieho programu
  - *cl_resampler* - trieda GPU implementácie spolu OpenCL hlavičkovými súbormi
  - *cpu_resampler* - trieda CPU implementácie prevzorkovania
  - *test* - testovací program
  - *plugin* - zdrojové kódy VST pluginu
- *text* - obsahuje tento dokument vo formáte PDF
- *latex_src* - zdrojové kódy tohto dokumentu a obrázky použité v texte
- *VST Plugin Analyser* - VST mini-host použitý pri vývoji demonštračného pluginu