VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

STANDARD OPENCL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

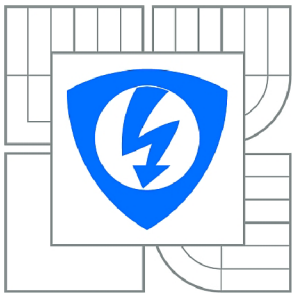
AUTOR PRÁCE
AUTHOR

JAKUB MICHLOVSKÝ

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

STANDARD OPENCL

OPENCL STANDARD

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

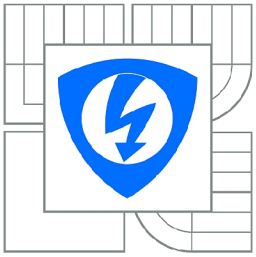
AUTOR PRÁCE — JAKUB MICHLOVSKÝ

AUTHOR

VEDOUCÍ PRÁCE — Ing. ZDENĚK PRŮŠA

SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Jakub Michlovský	***ID:***	72908
Ročník:	3	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Standard OpenCL

POKYNY PRO VYPRACOVÁNÍ:

Standard OpenCL definuje programové rozhraní (API) pro paralelní programování heterogenních systémů. V práci podrobně popište způsob instalace a konfigurace API, architekturu a model OpenCL. Zmapujte úroveň podpory standardu hlavními výrobci hardwaru a existenci rozšiřujících knihoven pro vědecké výpočty. Dále vytvořte program pro operace s barevným obrazem (změna odstínu, převod mezi barevnými modely, převzorkování obrazu), který bude sloužit k porovnání doby výpočtů. Program bude mít dostatečnou úroveň uživatelské interakce včetně vlastních ovládacích prvků a místa pro výpis doby vykonávání programu. Pro tvorbu GUI použijte vhodné knihovny např. OpenGL a freeGLUT.

DOPORUČENÁ LITERATURA:

[1] AMD: OpenCL™ Course: Introduction to OpenCL™ Programming: Training Guide, 2010. [online] Dostupné z:http://developer.amd.com/zones/openclzone/courses/pages/introduction-opencl-programming-may-2010.aspx

[2] Kirk D. B., Hwu W. W.: Programming Massively Parallel Processors: A Hands-On Approach (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA 2010, ISBN 0123814723

Termín zadání:	7.2.2011	***Termín odevzdání:***	2.6.2011

Vedoucí práce: Ing. Zdeněk Průša

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Tato práce pojednává o standardu OpenCL, který umí použít grafickou kartu pro akceleraci náročných výpočtů. V první části práce je podrobně rozebrána architektura standardu a podporova hlavních výrobců hardware a operačních systémů. Ve druhé části je rozepsána tvorba a implementace ukázkové aplikace, která provádí úpravy obrázku. Celá práce je zakončena srovnávacím měřením doby výpočtů na procesoru a na grafické kartě.

KLÍČOVÁ SLOVA

OpenCL, GPGPU, freeGLUT, paralelismus, úprava obrázku, převzorkování, změna odstínu, změna barevného modelu, RGB, HSV

ABSTRACT

This work discussed about the OpenCL standard, which can use video card for acceleration intensice calculations. In first part is an detail analyze of architecture of standard and support by major hardware manufacturers and operating systems. In the second part is described a creation and implementation of the application for adjustment of the picture. Whole work is ended by comparative measurment of the time of recon to processor and on the graphic card.

KEYWORDS

OpenCL, GPGPU, freeGLUT, paralelism, image adjustment, resampling, hue change, color model change, RGB, HSV

MICHLOVSKÝ, Jakub *Standard OpenCL*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2011. 49 s. Vedoucí práce byl Ing. Zdeněk Průša

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Standard OpenCL“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

POĎEKOVÁNÍ

Děkuji vedoucímu práce Ing. Zdeňku Průšovi za velmi užitečnou metodickou pomoc a cenné rady při zpracování bakalářské práce.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Ikdyž se vývoj v oblasti informačních technologií za několik posledních let velmi posunul, existuje stále mnoho oblastí, kde je potřeba obrovského výpočetního výkonu a i nejmodernějším procesorům doslova dochází dech. Typickou disciplínou je zpracování zvuku, obrazu a videa.

Od počátku programování existuje snaha o urychlení těchto výpočtů. Historicky se programátoři snažili intenzivně používat práci s ukazateli a psaní částí kódu nebo celých kódů v assembleru. S příchodem moderních procesorů, které obsahovaly instrukce MMX (MultiMedia Extensions) a SSE (Streaming SIMD Extensions) se výpočetní časy znatelně zkrátily a to díky jakémusi datovému mikroparalelismu. Převod Audio CD (Compact Disk) do formátu MP3 (MPEG-2 Audio Layer III) již netrval stovky minut, ale pouze desítky.

S příchodem vícejádrových procesorů se možnosti paralelizace ukázaly ve větším měřítku. Výkonové možnosti těchto procesorů byly zpočátku obrovské, bohužel jako největší problém se ukázala neschopnost vytížit všechna jádra tak, aby byl skutečně využit jejich potenciál. Dnes představují vícejádrové procesory jednu z nejlepších a nejčastějších cest k úlohovému paralelismu, bohužel datový paralelismus se zdá být stále neřešitelným problémem.

V okamžiku nástupu prvních 3D akcelerátorů snad nikoho nenapadlo, že by se tato velmi specializovaná technologie někdy dala použít i jako univerzální výpočetní jednotka. Nicméně dnes se tak již stalo a výkonné grafické karty jsou schopny provádět paralelizované výpočty. Tyto karty se dnes označují jako GPGPU (General-purpose computing on graphics processing units). Tato technologie je relativně mladá, ale díky ne příliš přesvědčivému výkonovému posunu CPU (Central Processing Unit), se zdá být velmi slibnou alternativou do budoucna.

V současné době existuje několik způsobů, jak využít grafické karty pro výpočty. Jedná se o NVIDIA CUDA (Compute Unified Device Architecture), ATI Stream nebo MS DirectCompute. Tyto technologie jsou buď vázané na karty daného výrobce, nebo operační systém.

Jako velmi univerzální řešení se zdá být relativně mladý standard OpenCL (Open Computing Language), kterým se zabývá tato práce. Aktuální specifikace přijatá 11. Června 2010 má označení 1.1. Jedná se o otevřený standard, který je od začátku myšlen jako platformě nezávislý. Česká literatura se tomuto tématu zatím vůbec nevěnuje, proto bylo nutné čerpat ze zahraničních serverů a za samotné specifikace OpenCL.

První část této práce se podrobně věnuje architektuře tohoto standardu. Jsou popsány čtyři základní modely a paměťové objekty, kterým je také třeba věnovat pozornost. Nakonec je jen stručně zmíněn OpenCL framework a jeho základní část.

Navazuje podrobný výpis tvůrců hardware a jejich postoji k OpenCL. Hlavní důraz je kladen na podporu a nástroje pro vývojáře. Dále je nezbytně nutné prozkoumat podporu hlavních operačních systémů.

Druhá část se věnuje vývoji ukázkové aplikace, která má umět změnu odstínu obrázku, převzorkování a změnu bareveného modelu. Tyto operace jsou nejdříve pojaty teoreticky a jejich princip je podrobně rozebrán. V příloze je několik ukázkových kódů, ty jsou v práci rezebrány a je na nich názorně vysvětleno programovaní v OpenCL. Poslední částí je měření časů, které jsou nutné pro vykonání té či oné operace za pomoci OpenCL zařízení a procesoru. Pro vývoj grafického rozhranní je použita technologie OpenGL (Open Graphics Library) a knihovna freeGLUT (Free OpenGL Utility Toolkit).

1 STANDARD OPENCL

1.1 O standartu obecně

Moderní architektury procesorů považují paralelismus jako důležitou cestu k vyššímu výkonu. V současné době se vyskytují technologické bariery v takování CPU. Vyšší taktování přináší neúměrně vysokou spotřebu vzhledem k výpočetnímu výkonu. Navíc je nutno řešit i náročné chlazení. Za velmi efektivní metodu zvyšování výkonu je považováno zvýšení počtu výpočetních jader. Mezi tím se GPU (Graphic Processing Unit) vyvinul z funkce renderovacího zařízení do přístroje, který může být naprogramován na zpracování paralelních procesorů. Protože dnešní počítačové systémy často obsahují vysoce paralelní CPU, GPU a dalších mnoho typů procesorů, je důležité umožnit vývojářům plně využít všechna výpočetní jádra.

Vytváření aplikací pro platformu heterogenního paralelního zpracování je výzvou, tradiční programovací přístupy pro multi-core CPU a GPU jsou velmi odlišné. Paralelní programovací modely pro procesory jsou obvykle založeny na normách, většinou požadují sdílený adresní prostor a nezahrnují vektorové operace. Obecně jsou modely pro programování GPU složité hierarchie pamětí a vektorových operací. Navíc jsou závislé na konkrétní platformě, výrobci nebo prodejci. Tato omezení ztěžují vývojáři přístup k výpočetnímu výkonu různorodých CPU, GPU a dalších procesorů. Více než kdy jindy, je třeba umožnit vývojářům softwaru, aby mohly účinně a plnou měrou využívat heterogenní výpočetní platformy. Od vysoce výkonných výpočetních serverů, přes stolní počítače až po mobilní přístroje. Běžně totiž počítače obsahují různorodou směs paralelních procesorů, grafických karet a dalších.

OpenCL je otevřený standard pro obecné účely programování paralelních procesů napříč klasickými procesory, grafickými procesory a dalšími speciálními procesory, což vývojářům softwaru dopřává přenosný a efektivní přístup k výkonu.

OpenCL podporuje širokou škálu aplikací, od software určeného pro spotřebitele, který je implementován do různých domácích řešení, až po high-power aplikace. Vytvořením efektivního programovacího rozhraní, se z OpenCL stává základní vrstva paralelních výpočtů pro ekosystém platformě nezávislých nástrojů a aplikací. OpenCL hraje stále významnější roli, hlavně co se týče mohutně rozmáhajících se interaktivních grafických aplikací, které kombinují obecné paralelní výpočetní algoritmy s vykreslováním grafiky.

OpenCL se skládá z API (Application Programming Interface) pro koordinaci paralerních výpočtů v heterogenních procesech a platformě nazávislého programovacího jazyka.

Rozhraní OpenCL bylo původně navrženo firmou Apple v roce 2008. První ucelený návrh byl předán organizaci Khronos, která má ve svém portfoliu i další stan-

dardy, jako je např. OpenGL. Khronos ve spolupráci s dalšími výrobci hardware a software došel ke konečné specifikaci OpenCL 1.0, dostupné o rok později. Dne 11.6.2010 bylo přijata verze OpenCL 1.1, která přináší tři nové datové typy, podporu pro řídící příkazy a mnoho dalšího.

1.1.1 Diagram tříd OpenCL

Obr. 1.1 popisuje OpenCL specifikaci jako diagram tříd jako UML (Unified Modeling Language) notaci. Tento diagram ukazuje třídy a jejich vzájemné vztahy. Pro zjednodušení jsou vyobrazeny pouze třídy než atributů či operací. Abstraktní třídy jsou zaznamenány jako {abstraktni}. Pokud jde o vztahy, zobrazena je agragace (plný diamand s popiskem), sdružení (bez popisku) a dědičnost (otevřená šipka). Hutnost vztahu je uvedena na konci každé anotace. V případě znaku "*"je myšlena velká hutnost, "1"znamená jeden jediný, "0 .. 1"reprezentuje jeden volitelný a "1 ..*"představuje jeden a více. [1]

1.2 Architektura OpenCL

OpenCL je otevřený průmyslový standard pro programování na CPU, GPU a dalších samostatných výpočetních zařízeních, které jsou uspořádány do jediné platformy. Tento standard je více než jen programovací jazyk. OpenCL je kompletní framework pro paralelní programování. Zahrnuje programovací jazyk, API, knihovny a runtime systém pro podporu vývoje softwaru. Při použití OpenCL může například programátor napsat univerzální programy, které jsou prováděny na GPU, aniž by bylo nutné zmapovat jejich algoritmy na 3D grafiku jako u OpenGL nebo DirectX.

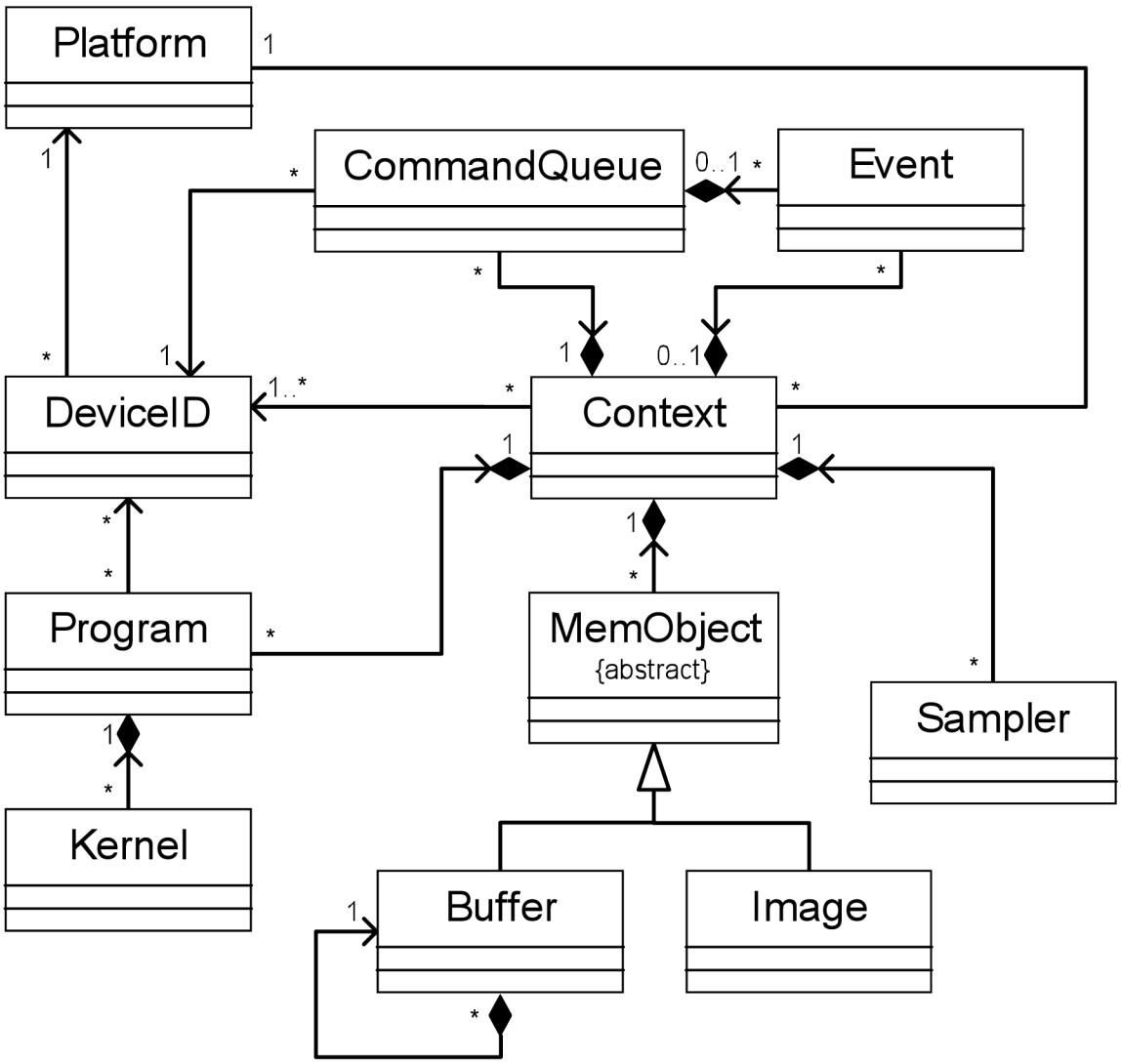
Cílem jsou zejména programátoři, kteří dosud nemohli napsat přenositelný a zároveň účinný kód. Jde hlavně o tvůrce knihoven, výrobců takzvaného *middleware* a programátory, kteří již vyvíjejí graficky orientované aplikace.

Standard OpenCL se sklá z těchto hlavních modelů:

- Platformní model
- Paměťový model
- Vykonávací model
- Programovací model

1.2.1 Platformní model

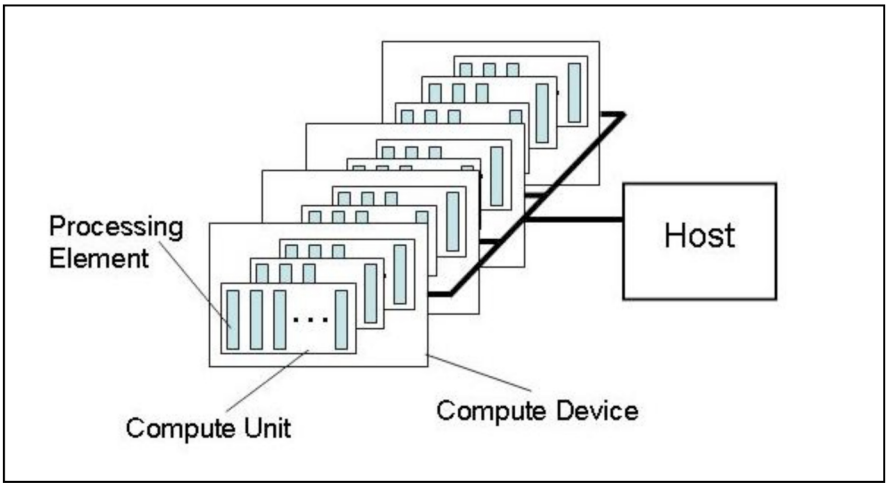
Platformní model OpenCL je definován na obr. 1.2 . Tento model se skládá z hostitelského systému **(host)**, který obsahuje jedno nebo více OpenCL zařízení **(compute**



Obr. 1.1: OpenCL UML diagram tříd [1]

device. Velmi obvyklým případem je systém, kde se mimo podporované grafické karty vyskytuje i podporovaný x86_64 procesor. Tato zařízení obsahují jednu nebo více vypočetních jednotek **(compute unit)**, které jsou dále rozděleny do jednoho nebo více prvků zpracování dat **(processing elements)**, to jsou například SSE jednotky procesoru anebo jednotlivé jádra multiprocesorů grafické karty.

OpenCL aplikace běží na hostitelském systému, protože je závyslá na modelech, které fungují na hostiteli nativně. Následně tato aplikace odesílá příkazy od hostitele, ty inicializují provedení výpočtů na prvcích pro zpracování v rámci zařízení. Prvky pro zpracování, které jsou v rámci jedné výpočetní jednotky, prochází pouze jediný proud informací. Jednotky se chovají jako SIMD (Single Intruction, Multiple Data - jednotky se spustí synchronně s tokem instrukcí) nebo jako SPMD (Single Process, Multiple Data - každý prvek si udržuje své vlastní počitadlo).[2]



Obr. 1.2: Platformní model s jedním hostitelem plus jedno nebo více výpočetních zařízení, každé s jednou nebo více výpočetními jednotekami, každá s jedním nebo více prvků zpracování dat. [1]

Podpora pro smíšené verze platforem

OpenCL je určeno pro podporu zařízení s různými vlastnostmi v rámci jedné platformy. To zahrnuje zařízení, které vyhovují různým verzím specifikací OpenCL. Zde jsou tři důležité verze identifikátoru, které jsou zasadní pro systém OpenCL: verze platformy, verze přístroje, a verze jazyka OpenCL C, kterou zařízení podporuje.[1]

Verze platformy označuje jaké verze runtime OpenCL jsou podporovány. To zahrnuje všechny API a co dalšího hostitel může využít k interakci s runtime OpenCL, například to jsou kontexty, paměti objektů, zařízení a velení fronty.[2]

Verze zařízení je údaj o schopnostech zařízení, který je reprezentován informací **clGetDeviceInfo**. Tuto informaci zařízení vrací při inicializaci. Příklady atributů, které jsou spojeny s verzí zařízení, jsou zdroje omezení nebo rozšíření funkcionality. Navrácená verze odpovídá nejvyšší specifikaci OpenCL, se kterou umí zařízení komfortně pracovat, ale zároveň není vyšší než verze platformy.[1]

Verze jazyka OpenCL C zařízení určuje s jakou verzí může developer pracovat. Vždy je uvedena nejvyšší podporovaná verze.

OpecCL C ja navržen tak, aby byl zpětně kompatibilní. Potom zařízení nemusí podporovat více než jednu verzi jazyka. Pokud je podporováno více jazykových verzí, vybere kompilátor jako výchozí tu nejvyšší verzi, kterou podporuje zařízení. Jazyková verze verze není vyšší než verze platformy, ale může být vyšší než verze zařízení.

1.2.2 Vykonávací model

Provádění programu OpenCL se objevuje ve dvou částech: **kernely**, které se provádějí na jednom nebo více zařízeních a **hostitelský program**, který běží na hostitelském systému. Hostitelský program definuje kontext kernelů a řídí jejich realizaci.

Jádro vykonávacího modelu OpenCL je definováno tím, jak jsou vykonávany kernely. Pokud je kernel prováděn na hostitelském systému, je třeba definovat indexovaný prostor. Instance kernelu spuštěná na jednom výpočetním prvku se nazývá **pracovní položka**, přičemž jsou tyto instance (vlákna) jednoznačně identifikovány v rámci indexovaného prostoru. Každá instance má v tomto prostoru přiděleno jednoznačné ID.[3]

Pracovní položky jsou organizovány do **pracovních skupin**. Tyto skupiny poskytují lepší pohled na využití celého indexované prostoru. Mají též svoje unikátní skupinové ID, které má stejný rozměr jako ID pro jednotlivé pracovní položky. Každé vlákno má i identifikátor v rámci pracovní skupiny. Tyto vlákna v rámci jedné pracovní skupiny mohou komunikovat prostřednictvím sdílené paměti.[1]

Celý indexovaný prostor OpenCL se nazývá **NDRange**. To je N-rozměrný prostor, kde N může nabývat hodnot 1,2 nebo 3. NDRange je definován jako celočíselné pole o délce N, které ja navíc upřesněno o rozsah indexu v každém prostoru, počínaje offsetním indexem F (defaultně nula). Každé globální a lokální ID pracovní položky je N-rozměrnou n-ticí. Globální ID má rozsah od F až od F plus počet elementů v daném rozměru minus jedna.[3]

Pracovní skupině jsou přiřazovány identifikátory podobným způsobem, jako globální ID u pracovních položek. Pole o délce N určuje počet pracovních skupin v každém rozměru. Pracovní položky jsou přiřazeny do pracovních skupin v závislosti k lokálnímu ID. Rozmezí je od nuly až po velikost pracovní skupiny v rámci jednoho rozměru minus jedna. Proto je kombinace ID pracovních skupin s lokálním

ID v ramci jedné skupiny jednoznačným identifikátorem pracovní položky. Každou pracovní položku je možné zjistit dvěma způsoby, buď globálním indexem, nebo indexem pracovní položky plus index v rámci pracovní skupiny.

Například, je uvažováno dvourozměrné indexované pole z obr. 1.3. Do indexovaného prostoru vstupují pracovní položky (G_x, G_y), velikost každé pracovní skupiny je (S_x, S_y) a globální ID offset (F_x, F_y). Globální indexy jsou definovány mezi G_x a G_y, kdy celkový počet pracovnívh ploţek je produktem G_x a G_y. Lokální indexy definují indexovaný prostor od S_x do S_y, kde počet pracovních položek v jedné pracovní skupině je produktem S_x, S_y. Vzhledem k velikosti jednotlivých pracovních skupin a celkovému počtu pracovních položek můžeme spočítat počet pracovních skupin. Dvourozměrné indexování prostoru se používá k jednoznačné identifikaci pracovních skupin. Každý pracovní bod může být identifikován jeho globálním ID (g_x, g_y) nebo kombinací ID pracovní skupiny (w_x, w_y), velikostí každé pracovní skupiny (S_x, S_y) a místním ID (s_x, s_y) v rámci pracovní skupiny. Celý princip je demonstován na vztahu1.1.[1]

$$(g_x, g_y) = (w_x S_x + s_x + F_x, w_y S_y + s_y + F_y) \tag{1.1}$$

Počet pracovních skupin může být vypočítán dle vztahu 1.2.

$$(W_x, W_y) = (G_x/S_x, G_y/S_y) \tag{1.2}$$

Dostáváme globální ID a velikost pracovní skupiny, ID pracovní skupiny pro pracovní položku se vypočítá podle vzahu 1.3.

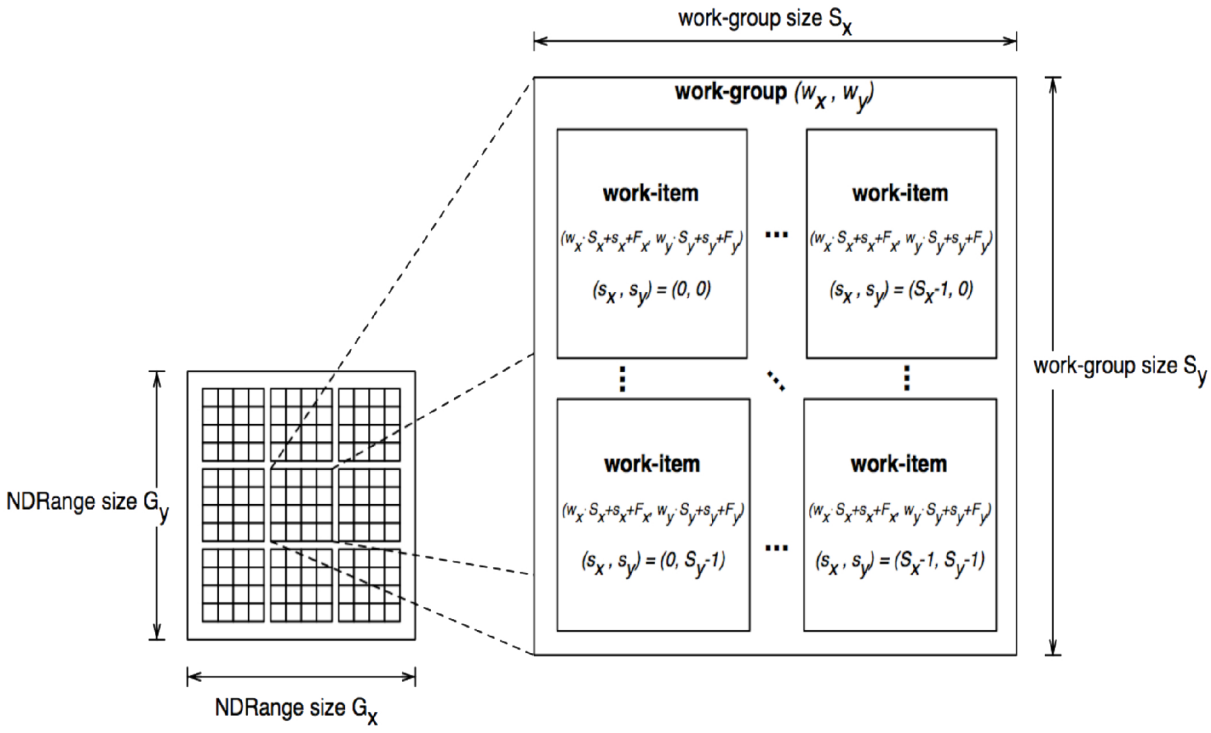
$$(w_x, w_y) = ((g_x - s_x - F_x)/S_x, (g_y - s_y - F_y)/S_y) \tag{1.3}$$

Široké programové spektrum modelů může být mapováno právě tímto spouštěcím modulem. Explicitně OpenCL podporuje oba modely: **datový paralelní programovací model** a také **úkolový paralelní programovací model**. [2]

Kontext a fronty příkazů

Hostitelský systém definuje kontext pro vykonání kernelů. Kontext zahrnuje následující zdroje:

1. **Zařízení:** Souhrn všech OpenCL zařízení, které jsou použitelná na hostitelském systému.



Obr. 1.3: Příklad NDRange indexovaného prostoru, který ukazuje pracovní položky, jejich globální identifikátory a jejich mapování na páru pracovní skupina a lokální ID. [1]

2. **Kernely:** OpenCL funkce, které běží na OpenCL zařízeních.
3. **Programové objekty:** Programový kód a spustitélný kód, který je implementován do kernelu.
4. **Paměťové objekty:** Sada paměťových objektů viditelných pro hostitele a OpenCL zařízení. Paměťové objekty obsahují hodnoty, které mohou být provozovány na instancích kernelu.

Hostitel vytváří a manipuluje s kontextem pomocí funkce z OpenCL API. Hostitel vytváří datovou strukturu tzv. **fronty příkazů** pro koordinaci provádění kernelů na zařízeních. Hostitel umisťuje příkazy do těchto front, které jsou poté plánované v rámci zařízení. Mezi ně patří:

- **Prováděcí příkazy kernelu:** Spuštění kernelu na prvcích zpracování na zařízení.
- **Paměťové příkazy:** Přesouvání dat do, z a mezi paměťovými objekty nebo mapování a odmapovávání paměťových objektů z hostitelského adresného pro-

storu.

- **Synchronizační příkazy:** Zachování pořadí prováděných příkazů.

Fronta příkazů plánuje příkazy pro spuštění na zařízení. Ty se spouští asynchronně mezi hostitelem a zařízením. Příkazy se mohou vzhledem k sobě samým spouštět ve dvou režiměch:

- **Vykonávání v pořadí**: Příkazy jsou prováděny v pořadí, v jakém se objeví ve frontě a dokončovány jsou přesně v tomto pořadí. Jinými slovy je prioritnější příkaz z fronty dokončen dříve než je spuštěn ten následující. Takto se serializuje fronta spoštěcích příkazů.
- **Vykonávání mimo pořadí:** Příkazy jsou také spouštějí v pořadí, v jakém se objeví ve frontě, ale nečekají na dokončení předchozího příkazu. Omezení libovolného provádění příkazů záleží jen na programátorovi, který dodržování pořadí řeší přes příkazy explicitní synchronizace.

Provádění kernelu a paměťové příkazy posílájí do fronty příkaz pro generování událostí. Toto se používá na kontrolu provádění mezi příkazy a koordinaci provádění mezi hostitelem a zařízením.

Je možné spojit více front s jedním kontextem. Tyto fronty mohou běžet souběžně a nezávisle bez explicitního mechanismu, synchronizace mezi nimi je řešena v rámci OpenCL.

Kategorie kernelů

Vykonávací model OpenCL podporuje dvě kategorie kernelů:

- **OpenCL kernely** jsou napsány OpenCL C programovacím jazykem a jsou zkompilovány pomocí OpenCL kompilátoru. Všechny OpenCL implementace podporují OpenCL kernely. Implementace může stanovit další mechanismy pro vytváření OpenCL kernelu.
- **Nativní kernely** jsou přístupné skrze hostitelskou funkci pointer. Nativní kernely jsou zařazovány do fronty spolu s OpenCL kernely na zařízení a objekty se sdílenou pamětí obsahující OpenCL kernely. Například mohou být tyto nativní kernely funkcně definovány v kódu aplikace nebo být exportovány z knihovny. Schopnost spouštět nativní jádra je funkce volitelná.[1]

1.2.3 Paměťový model

Pracovní položky provádějící kernel mají přístup do čtyř různých typů pamětí:

- **Globální paměť:** Tato oblast umožňuje čtení / zápis všem pracovním položkám ve všech pracovních skupinách. Pracovní položky mohou číst nebo zapisovat z libovolné části paměťového objektu. Čtení a zápis globální paměti může být cachováno v závislosti na možnostech zařízení.
- **Konstantní paměť:** Tento region globální paměti zůstává konstantní po celou dobu provádění kernelu. Hostitelský systém přiděluje a inicializuje paměť objektů umístěných do konstantní paměti.
- **Lokální paměť:** Oblast paměti pro pracovní skupiny, může být použita pro alokování proměnných a sdílena pro všechny pracovní prvky této pracovní skupiny. Může být implementována jako vyhrazená část paměti OpenCL zařízení. Alternativně může být namapována na část nebo části globální paměti.
- **Privátní paměť:** Je určena výhradně pro pracovní prvky. Zde definované proměnné jsou viditelné pouze pro prvek kde byly definována. Ostatní pracovní prvky nemají k této paměti přístup.

Tab. 1.1 popisuje jak kernel nebo hostitelský systém může alokovat oblast paměti (statické tj. kompilace vs. dynamické tj. runtime) a jaká práva mají pro čtení / zápis.

Tab. 1.1: Paměťový region - Alokace a přístup k paměti

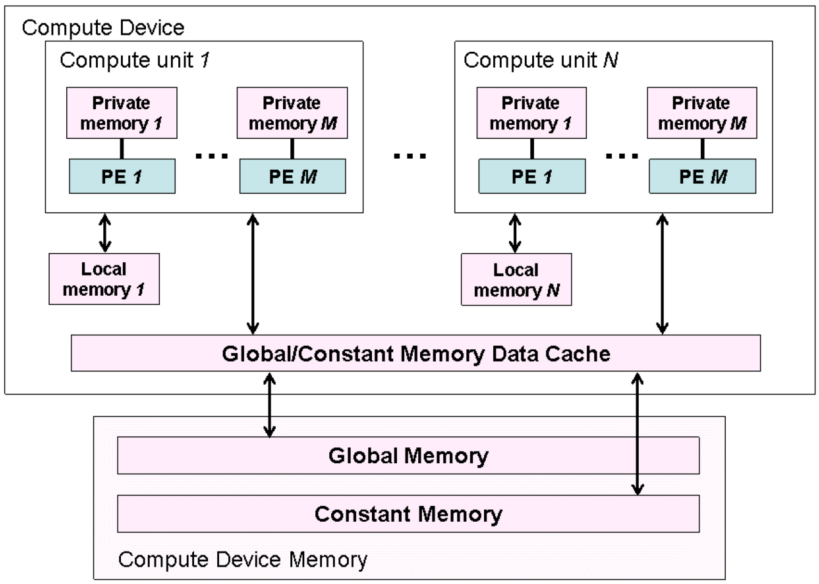
	Globalní paměť	Konstantní paměť	Lokální paměť	Privátní paměť
Hostitel	Dyn. alokace Práva: Čtení / zápis	Dyn. alokace Práva: Čtení / zápis	Dyn. alokace Práva: Bez práv	Bez alokace Práva: Bez práv
Kernel	Bez alokace Práva: Čtení / zápis	Stat. alokace Práva: Pouze čtení	Stat. alokace Práva: Čtení / zápis	Stat. alokace Práva: Čtení / zápis

Oblasti pamětí a jejich vzájemné relace ilustruje obr. 1.4.

Aplikace, která běží na hostitelském systému používá OpenCL API, vytváří paměťové objekty v globální paměti a tvoří frontu paměťových příkazů, které pak operují s těmito objekty.[1]

Paměťové moduly hostitele a OpenCL zařízení jsou co nejvíce rozdělené a na sobě co nejvíce nezávislé. Jedná se o nutnost, neboť hostitel je definován mimo OpenCL. Občas je ale potřeba interakce, která se vyskytuje v jednom ze dvou způsobů: explicitní kopírování dat nebo mapování a odmapování regionů v paměti objektu.[2]

Mají-li být data kopírována explicitně, hostitel zařadí příkazy pro přenos dat mezi paměti objektů a paměť hostitele. Tato paměť může nebo nemusí být pro přenos



Obr. 1.4: Koncepce architektury OpenCL zařízení s prvky pro spravování dat (PE) a výpočetními jednotkami. [1]

příkázů blokována. Pokud OpenCL volá funkci blokování mohou být ostatní přidružené zdroje bezpečně použity znovu. Pro neblokování používá OpenCL zařazení do fronty bez ohledu na to, jestli je hostitelova paměť bezpečně použitelná. [1]

Mapování a odmapování je způsob interakce mezi hostitelským systémem a OpenCL paměťovými objekty, umožňující hostitely použít určitý region z paměťového objektu do vlastního adresovaného prostoru. Mapa pamětí může nebo nemusí příkaz blokovat. Jakmile je region z paměťového objektu namapován, může hostitel číst ne zapisovat do této oblasti. Hostitel region odmapuje pokud je s přístupem (se čtením a / nebo zápisem) kompletně hotov.

Konzistence paměti

OpenCL používá uvolněnou konzistenci paměti, tj. pro stav paměti viditelné pro pracovní položku není garantována konzistence skrz všechny pracovní položky po celou dobu.

Lokální paměť je konzistentní skrz pracovní položku a jednu pracovní skupinu až k bariéře pracovní skupiny. Globální paměť je také konzistentní skrz pracovní položku a jednu pracovní skupinu až k bariéře pracovní skupiny, ale nejsou zde žádné jistoty konzistence mezi různými pracovními ploškami, které provádějí kernel.[1]

Konzistence paměti pro paměťové objekty, která je sdílená mezi příkazy zařazenými ve frontě, je vynucena synchronizacním ukazatelem.

1.2.4 Programovací model

Vykonávací model podporuje datově paralelní i úkolově paralelní programovací modely, stejně jako podporuje hybridní modely složené z těchto dvou. Primární model řízení návrhu je datově paralelní.

Datově paralelní programovací model

Datově paralelní programovací model definuje vypočet z hlediska posloupnosti instrukcí použitých na více prvcích paměťových objektů. Indexovaný prostor spojený s OpenCL vykonávacím modelem definuje pracovní položky a jak jsou data zmapována uvnitř pracovní položky. Ve striktně datově paralelním modelu je mapování mezi pracovní položkou a prvkem v paměťovém objektu, ve které může být paralelně vykonáván kernel, jedna k jedné. OpenCL implementuje volnější verzi, takže poměr jedna k jedné není nutný.[1]

OpenCL poskytuje hierarchický datově paralelní programovací model. Existují dva způsoby, jak určit hierarchické dělení. V explicitním modelu programátor definuje celkový počet pracovních položek prováděných současně a také jak jsou tyto položky rozděleny do pracovních skupin. V implicitním modelu programátor specifikuje pouze celkový počet pracovních položek, které se provadí součásně. Rozdělení do skupin je režii OpenCL.

Úkolově paralelní programovací model

Je to model, ve kterém je provedena jedna instance kernelu nezávisle na indexovaném prostoru. To je logický ekvivalent k vykonávání kernelu na výpočetní jednotce, kde je v pracovní skupině jen jediná pracovní položka. Pod tímto modelem mohou uživatelé vyjádřit paralelismus takto:

- pomocí vektorových datových typů implementovaných na zařízení,
- zavedením do fronty více úkolů,
- a / nebo zařezení do fronty nativní jádra vytvořená pomocí programovacího modelu ortogonálně k OpenCL.

Synchronizace

Existují dvě domény synchronizace v OpenCL:

- Pracovní položky v jedné pracovní skupině

- Příkazy zařezené do fronty v jedno kontextu

Synchronizace mezi pracovními položkami v jedné pracovní skupině se provádí pomocí bariéry pracovní skupiny. Všechny pracovní položky v pracovní skupně musí nejdříve překonat bariéru, teprve potom je jim dovoleno pokračovat ve vykonávání. Bariéra je jakýsi záchytný bod, kde se běh pracovních položek srovná. U bariéry pracovní skupiny se setkají všechny pracovní položky, které vykonávájí kernel. Tento nechanismus synchronizuje pouze pracovní položky nikoliv celé skupiny.[1]

Synchronizační body mezi příkazy a frontou příkazů:

- Bariéra fronty příkazů zajištuje, že všechni dříve do fronty zařazené příkazy mají vykonávání dokončeno a všechny následné aktualizace paměťových objektů jsou viditelné, pro následně zařazené příkazy, které ještě nezačaly s vykonáváním. Tato bariéra synchonizuje pouze příkazy v rámci jedné fronty příkazů.[2]
- Čekání na údálost. Všechny funkce OpenCL API, které zařazují příkazy do front, vracejí informace o identifikaci přízaku a aktualizaci paměťových objektů. Následující příkaz čeká na návrat těchto hodnot, tím je zaručena viditelnost paměťových objektů před tím, než se tento příkaz začne vykonávat.[2]

1.2.5 Paměťové objekty

Paměťové objekty se dělí na dva typy: *buffer* objekty a *image* objekty. *Buffer* objekt bývá naplněn jednorozměrným polem prvků zatímco objekt *image* se používá pro uchování dvou čí třírozměrných polí či textur, obrázků nebo frame-buffrů.

Části objektu *buffer* mohou být skalární data (jako int, float), vektorová data nebo uživatelsky definovaná struktura. Objekty *image* se používají na reprezentaci zásobníku, který může být použit jako textura nebo frame-buffer. Části obrázku jsou vybírány z řady předem definovaných formátů. Minimální počet těchto částí je jedna.

Základní rozdíly mezi *buffer* a *image* objektem jsou:

- Prvky objektu *buffer* jsou řazeny sekvenčně, takže k nim může kernel vykonávaný na zařízení přistupovat pomucí ukazatele (pointeru). Naopak prvky v objektu *image* jsou řazeny ve formátu, který je takříkajíc neprůhledný, proto je použití pointeru vyloučeno. Zabuduvané funkce OpenCL C jazyku umí povolit kernelu čtení a zápis do tohoto objektu.

- Pro *buffer* objekt jsou data uložena ve stejném formátu, v jákém k němu přistupuje kernel, ale v případě objektu *image* může být formát pro ukládání obrazových prvků stejný jako formát použitý v kernelu. Obrazová data jsou vždy uložena ve čtyř komponentním vektoru (každá komponenta může být typu float nebo signed / unsigned char) v kernelu. Vestavětá funkce na čtení z obrazu konvertuje části z formátu, ve které je obraz nahrán, do formátu čtyř komponentního vektoru. Podobně převádí vestavěná funkce pro zápis. Ze čtyř komponentního vektoru do odpovídajících formátu např. čtyři osmibitové prvky.[3]

Paměťové objekty jsou popsány **cl_mem** objektem.

1.2.6 OpenCL framework

OpenCL framework umožňuje aplikacím využít hostitelský systém a jedno nebo více OpenCL zařízení jako jeden heterogenní paralelní výpočetní systém. Framework obsahuje následující komponenty:

- **OpenCL platformní vrstva:** umožňuje hostiteli prozkoumat OpenCL zařízení, porovnat jejich vlastnosti a vytvořit kontext.[2]
- **OpenCL runtime:** umožňuje hostiteli manipulovat s kontexty, které byly vytvořeny.
- **OpenCL kompiler:** vytváří spustitelné programy, které obsahují OpenCL kernely. Objekt *buffer* se používá výhradně na jedno rozměrné záležitosti a *image* objekt se používá na dvou či tří rozměrné textury, frame-buffer nebo obrázky.

1.3 Podpora výrobců hardware

Standard OpenCL začíná být velmi žádaný, protože umožňuje využít stávající zdrojový kód a s minimálnimi změnami jej efektivněji využít. Výrobci hardware musí s tímto trendem držet krok, a proto je téměř nutností poskytnout programátorům podporu pro tvorbu v OpenCL. Snad všichni větší výrobci hardware se snaží angažovat a dodat pořebné nástroje pro vývoj. Všechny údaje jsou platné ke květnu 2011. Mezi ty hlavní výrobce patří:

1.3.1 NVIDIA

Známá kalifornská firma se již od roku 1993 věnuje výrobě grafických chipů. V posledních letech vyrábí i specializované výpočetní stanice a chipy platformy ARM (Advanced RISC Machine).

Nutnost paralelizace výpočtu si v NVIDII uvědomili již dříve a tak na konci roku 2006 přinesli vlastní technologii CUDA. V prvních verzích byly podporovány jen ty nejvyšší modely grafických karet, které byly velmi drahé a tudíž pro mnoho vývojářů nedostupné. V pozdějších specifikacích se počet kompatibilních karet značně rozrostl, protože přibyli i levnější karty, notebookové dedikované karty i karty integrované v chipsetech a to jako mobilní tak stolní verze. CUDA je dnes velmi populární a existuje pro ni mnoho předpřipravených knihoven pro specializované výpočty a pomáhá ji velká komunita vývojářů.[8]

Tato firma stála u zrodu OpenCL a proto se celý framework velmi podobá technologii CUDA. V současné době nabízí NVIDIA několik manuálů jak začít s OpenCL a na stránkách pro vývojáře je dostatek ukázkových kódů. Bohužel se zde neobjevují žádné knihovny pro specializované výpočty. Prim jednoznačně drží mateřská technologie, OpenCL je zmiňována spíše okrajově a je zařazena do CUDA balíku.

Vývojářům jsou k dispozici speciální vývojářské ovladače pro grafické karty a balík **CUDA toolkit**, který obsahuje vše potřebné pro vývoj v CUDA A OpenCL, včetně kompilátoru, hlavičkových souborů, zmíněných manuálů a ukázkových kódů. Dále je možné využít nástrojů pro debugovaní a analýzu, které umožní lépe využít paralelismu karty. Velmi zajímavý program je CUDA profiler, který pomahá optimalizovat kernely, pro co nejvyšší výkon. [8]

Seznam karet[1], které podporují OpenCL uvádí výrobce ve svých specifikacích, jedná se převážně o některé modely z řady 8800 a 9800. Z moderních řad 200, 300, 400 a 500 jsou podporovány všechny karty a to i moderní mobilní dedikované i integrované karty a profesionální karty Quatro.

ARM chip Tegra2, který je aktuálně používán v tabletech a mobilných telefonech v kombinaci s operačním systémem Android od Google, je také podporován, takže je možno použít OpenCL kód i na mobilních zařízeních.

1.3.2 Intel

Jeden z největších výrobců integrovaných obvodů sice pomáhal s vytvářením standardu, ale do podpory se zapojil až se specifikací 1.1. Nyní nabízí OpenCL SDK (Software Development Kit) ve verzi beta, která je ale velmi stabilní a pro vývoj

[1] http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/OpenCL_Programming_Guide.pdf. strana 39

může být mez problémů použita. Finální verze je očekávána ve velmi krátké době. Pro vývojáře je dostupných několik ukázkových kódů a návody jak začít s OpenCL a tipy a triky. Podobně jako u NVIDIE se neobjevuje žádná knihovna pro specializované matematické výpočty, ale podporu zahrnují i nástroje na debugování a analýzu s velmi obsáhlým manuálem. [9]

Intel uvádí i seznam podporovaných procesorů[2], jsou to modely obsahují instrukce Intel SSE 4.1 nebo SSE 4.2 případně AVX (Advanced Vector Extensions). Kompletní podpora je zaručena u všech modelů Intel i3, i5 a i7. Modely Core 2 nejsou podporovány všechny, ale pouze určité série. Naopak vůbec podporovány nejsou integrované grafické karty a to ani moderní modely chipů, které jsou součástí posledních procesorů řady Sandy Bridge. To je Intelu vytýkáno a je velmi pravděpodobné, že podpora bude o tyto zařízení rozšířena, protože jejich výpočetní výkon není rozhodně zanedbatelný. Celé této spekulaci přidává na důvěryhodnosti nedávno vydaný nástroj pro optimalizaci OpenCL kódu pro grafické karty Intel, který byl nedopatřením zmíněn na sociální síti Twitter jedním ze zaměstnanců Intelu.

1.3.3 AMD/ATI

Původně bylo AMD jen výrobcem procesorů, ale v půlce roku 2006 koupilo i velkého výrobce grafických karet ATI a před několika měsíci přejmenovalo všechny nové grafické karty na AMD.

Firma ATI vytvořila také svou vlastní technologii pro paralelní výpočty. Technologie ATI Firestream nebyla zdaleka tak úspěšná jako konkurenční CUDA. Možná i proto, že vychází z FireGL (Fire Graphics Library), která ze začátku nebyla veřejně přístupná a používala speciální programovací jazyk. Firestrem (někdy označovaný jako ATI APP - Ati Accelerated Parallel Processing) je u AMD spíše na druhé místě. První místo obsadila techlogie OpenCL, takže situace je přesně opačná jako u NVIDIE.[10]

AMD poskytuje vývojářům zdaleka nejvyšší komfort. Kromě klasického SDK, které obsahuje v podstatě totéž co konkurence, přidává seriál o vývoji, mnoho tipů a triků, knihovny pro speciální výpočty (BLAS - Basic Linear Algebra Subprograms i FFT - Fast Fourier transform), vývojářské ovladače, profiler a optimalizer kernelů a obsáhlé diskuzní fórum na podporu vývojářů.

AMD podporuje jak své procesory tak své grafické karty. Jedná se o všechny procesory s instrukcemi SSE 2.x a vyššími a všechny moderní grafické karty v mobilní a desktopové variantě. Kompletní přehled uvádí výrobce na svých stránkách[3].

[2] http://software.intel.com/en-us/articles/opencl-release-notes/

[3] http://developer.amd.com/gpu/AMDAPPSDK/pages/DriverCompatibility.aspx

1.3.4 Ostatní výrobci

OpenCL si našlo své zastánce i na poli mobilních ARM procesorů, je podporována již zmíněná Tegra2 od NVIDIE. Apple podporuje OpenCL na svých A4 a A5 chipech, které se používají v iPhonech a iPadech. Další výrobci jsou Qualcomm nebo Texas Instruments.

Na poli desktopových grafik je třeba zmínit firmu S3, její grafiky mají plnou podporu OpenCL 1.1, DirectX 10.1 a OpenGL 3.0, přesto jsou u nás spíše exotickou záležitostí. Firma S3 navíc nedodává vývojářům žádné nastroje.

Ostatní firmy, které se v OpenCL angažují jsou: Ericsson, Nokia, Broadcom, Sony a další.

1.4 Podpora operačních systémů

1.4.1 Microsoft Windows

Nejrozšířenější operační systém neposkytuje podporu pro vývoj přímo, ale je nutno doinstalovat podporu od výrobce hardware, na kterém bude vývoj probíhat. To většinou znamená vyvojářské ovladače a SDK. Pro samotné programováni je třeba ještě vývojové prostředí.

Vývojových prostředí je pro tento systém velké množství, ale jako velmi výhodné se jeví Microsoft Visual Studio 2010, ve kterém je možné OpenCL aplikaci naprogramovat. Je také možné použít free variantu s přídomkem Express.

Hlavní výrobci hardware (Intel, AMD, NVIDIA) podporují systém Windows XP a vyšší. A to jak 32bitovou tak 64bitovou variantu. Jedinou vyjímkou jsou grafické karty AMD, kterou už nejdou programovat pod systémem XP.

Microsoft také vyvinul svou vlastní technologii DirectCompute. Tato technologie ja vázaná systémem Windows Vista nebo 7 a kompatibilní grafickou kartou s DirectX 10 a dnes i 11. Microsoft sice uveřejnil vývojové nástroje, ale dnes tuto technologii využívá hlavně sám Microsoft na akceleraci součástí systému.

1.4.2 MAC OS X

Firma Apple je jedním z iniciátorů a hlavních tůrců standartu OpenCL. Od verze 10.6 Snow Leopard je podpora OpenCL přímo implementována do systému. Vzhledem k uzavřenosti a omezenému portfoliu použitých grafických karet a procesorů může Apple garantovat použitelnost OpenCL na každém počítači, který byl dodán se systémem 10.6. Apple používá jen procesory Intel a grafiky NVIDIA a AMD. Bohužel podpora starších strojů s PowerPC procesorem G5 není poskytována.

Jako vývojé prostředí je možno využít Xcode4, které vyvíjí přímo Apple a je k němu dostatek materiálů o programování obecně a o programování v OpenCL. Apple hodně tlačí vývojáře do programování v OpenCL a to nejen pro desktopový operační systém, ale i pro mobilní iOS variantu a nabízí ohromné množství materiálů buď formou videí a přednášek na iTunes nebo v textové podobě na svých stránkách pro vývojáře.

1.4.3 GNU/Linux

Tento systém je na tom s podporou podobně jako systém Windows, přímá podpora pro vývoj neexistuje a je potřeba ji doinstalovat. Naštěstí se hlavní výrobci hardware podpory linuxu nestrání a nabízí vše potřebné pro vývoj. Pro nejrozšířenější distribuce jsou nachystány přímo instalační balíky. Vývojové prostředí bývá obvykle součástí systému, pokud ne je možno vybírat z nepřeberného množství.

1.4.4 Virtualizované systémy pod VMware

Tento výrobce virtualizačního software umožňuje využít OpenCL na virtualizované grafické kartě. Takže je možné například vyvíjet OpenCL kód na Windows, které jsou virtualizovány na MAC OS X. Výkon se nemůže s fyzickou kartou kartou rovnat, ale pro vývoj je to zpravidla dostačující.

1.4.5 Mobilní systémy

Hodně mladou a rychle rostoucí skupinou jsou mobilní systémy, které se za posledních několik let posunuly od velmi jednoúčelových aplikací až do univerzálních sytémů, které v mnohém konkurují klasickým počítačovím systémům. Jak rostly nároky uživatelů, rostli také nároky aplikací a OpenCL si našlo své místo i zde. Google Android ve verzi pro mobilní telefony a i ve verzi pro tablety umožňuje programovat a využívat OpenCL. Podobně se chová i Apple, jeho iOS je také schopen maximálně využít tuto technologii. Mladý mobilní systém Windows Phone 7 zatím s OpenCL nespolupracuje, ale je jen otázkou času, kdy se objeví náročnější aplikace, které bude akceleraci potřebovat.

2 VÝVOJ UKÁZKOVÉHO PROGRAMU

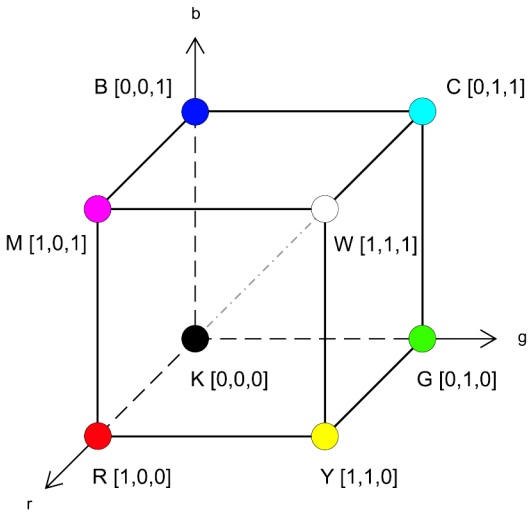
Cílem práce je navrhnout aplikaci, která bude umět změnit odstín obrázku, převod do jiného barevného modelu a převzorkování obrázku tak, aby operace bylo možné provádět na CPU i akcelerovaně pomocí grafické karty a OpenCL. Z aplikace bude možné odečíst časy provádění jednotlivých kroků.

2.1 Návrh

Před tvorbou samotné aplikace bylo nutné objasnit si, jak potřebné úpravy fungují teoreticky, pak bude možné vytvořit zdrojový kód. Teoretická část všech úkonů je podrobně popsána v následujících podkapitolách.

2.1.1 Změna odstínu a převod do jiného barevného modelu

Tyto dva úkoly je velmi výhodné spojit. Z obrázku je možné získat informace o barevných složkách každého pixelu. Ty mohou být interpretovány buď osmibově nebo celočíselným rozsahem 0 - 255. Přímo ze složek červené, zelená a modré není možné zjistit odstín a případně jej změnit. Proto je nutné RGB (Red, Green, Blue) model převést na jiný barevný model, kde je možné odstín přímo odečíst a modifikovat. RGB model je zobrazen na obr. 2.2.



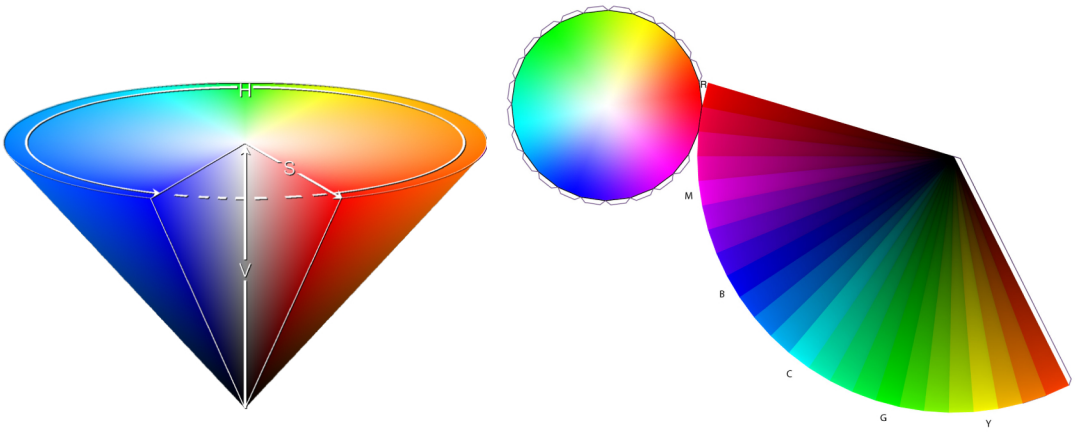
Obr. 2.1: Ukázka RGB modelu [5]

Barevný model RGB je aditivní způsob míchání barev, technicky jde o vyzařování těchto barev, což se reálně děje na monitorech či projektorech. Čím více barev se

sečítá tím světlejší je výsledek. Sečtení všech tří složek s maximální hutností vytváří bílou. [5]

Modelů ze kterých je možno získat odstín je více, ale nesnadnější převod z RGB a zpět nabízí model HSV (Hue, Saturation, Value).

Z modelu HSV je možné přímo odečíst odstín, ten je interpretován přímo jednou z hlavních složek. Změna odstínu probíhá ve třech krocích: převod z RGB do HSV, samodná změna odstínu, převod zpět na RGB. H (Hue) reprezentuje odstín, což je hlavní spektrální složka, S (Saturation) reprezentuje sytost, která je označována také jako čistota či živost barvy. V (Value) reprezentuje světlost nebo-li jas. Někdy bývá místo písmena V použito písmeno B (Brightness) a model je označován jako HSB (Hue, Saturation, Brightness). Jde nejen o rozdílné označení, hodnota H je sice stejná v obou případech, ale zbylé dvě hodnoty se liší a mají odlišné algoritmy výpočtu. Model HSV je zobrazen na obr. 2.2. [5]



Obr. 2.2: Ukázka HSV modelu [5]

Převod z RGB je poměrně jednoduchý, všechny tři barevné složky se porovnají. Ta s nejvyšší hodnotou je označena jako *max* a naopak ta s nejnižší hodnotou je označena jako *min*. Rozdíl *max* a *min* je možno označit jako *delta*.

Pomocí těchto pomocných proměnných je možno spočítat hodnotu H. Výpočet je uveden v tab. 2.1. Proměnná S se spočte jako $1 - \frac{min}{max}$, pokud je $max = 0$ je i hodnota S rovna nule. Proměnná V je rovna *max*. Hodnota H je v rozsahu od 0 do 360 stupňů. Takže je možné k ní přičíst libovolný počet stupňů a tím bude provedena změna odstínu. Pokud přesáhne proměnná H přičítáním 360 stupňů, je třeba od této hodnoty 360 stupňů zase odečíst.

Změněnou hodnotu H a nezměněné hodnoty S a V je třeba zpět přepočítat na model RGB. Do proměnné i je nutné uložit zbytek po dělení odstínu H číslem 60. Pro dokončení výpočtu je třeba použít dalších proměnných f,p,q,t. Jejichž hodnoty

Tab. 2.1: Výpočet parametru H při přepočítávání HSV z RGB

Hodnota H	Podmínka
0	$max = min$
$60° * \frac{\mathrm{G}-\mathrm{B}}{delta} + 0°$	$max = \mathrm{R}; \mathrm{G} \geq \mathrm{B}$
$60° * \frac{\mathrm{G}-\mathrm{B}}{delta} + 360°$	$max = \mathrm{R}; \mathrm{G} < \mathrm{B}$
$60° * \frac{\mathrm{B}-\mathrm{R}}{delta} + 120°$	$max = \mathrm{G}$
$60° * \frac{\mathrm{R}-\mathrm{G}}{delta} + 240°$	$max = \mathrm{B}$

budou potřeba pro vyčíslení hodnot RGB.
$f = h - i$
$p = \mathrm{V} * (1 - \mathrm{S})$
$q = \mathrm{V} * (1 - f * \mathrm{S})$
$t = \mathrm{V} * (1 - (1 - f) * \mathrm{S})$

Z těchto proměnných je potom možné odečíst hodnoty barevných kanálů podle tab. 2.2

Tab. 2.2: Určení parametrů RGB při přepočítávání z HSV

R, G, B	Podmínka
V, t, p	$i = 0$
q, V, p	$i = 1$
p, V, t	$i = 2$
p, q, V	$i = 3$
t, p, V	$i = 4$
V, p, q	$i = 5$

Celý výpočet není nijak složitý a zdá se, že by jej mohly moderní procesory bez problémů zvládnout. Bohužel, při klasickém programování musí procesor procházet obrázek pixel po pixelu a u každého provést výpočet. Pokud má obrázek velmi vysoké rozlišení, může přepočítání všech pixelů trvat nepříjemně dlouho. Právě tato disciplína je v hodná pro zpracování s využitím možnosti maximální paralelizace.

2.1.2 Převzorkování obrázku

Je dána základní obrazová matice o šířce X a výšce Y. Každý pixel je určen souřadnicí (x,y) Z tohoto obrázku je možno vytvořit obrázek menší o šířce X' a výšce

Y'. Poměry mezi původní a novou šírkou a výškou reprezentují poměry mx a my. Pro příklad je uvažován nový obrázek menší s koeficienty mx a my. Logicky musí dojít k tomu, že některé pixely budou v menším obrázku vypuštěny. Je možné určit pro každý bod o souřadnicích (x,y) jeho nové odpovídající souřadnice (x',y') podle rovnice 2.1

$$\mathrm{M}\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.1}$$

Matice M má tvar uvedený ve vztahu 2.2 protože dochází jen ke změně stran a nedochází k rotaci obrázku.

$$\mathrm{M} = \begin{pmatrix} mx & 0 \\ 0 & my \end{pmatrix} \tag{2.2}$$

Tento způsob ale není úplně vhodný, protože se hledají souřadnice i pro pixely, které budou vypuštěny. Ty také dostanou souřadnice, proto se musí u každé souřadnice ověřit, jestli patří do matice nového obrázku či nikoli.

Proto je vhodnější přistupovat postupně k pixelům nového obrázku a k nim hledat odpovídajíci souřadnice v původním obrázku, což ilustruje vzatah 2.3

$$\mathrm{M}^{-1}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix} \tag{2.3}$$

Matice je pak maticí inverzní a má tvar, který je uveden ve vztahu 2.4

$$\mathrm{M}^{-1} = \begin{pmatrix} \frac{1}{mx} & 0 \\ 0 & \frac{1}{my} \end{pmatrix} \tag{2.4}$$

Protože souřadnice získané x,y nejsou celá čísla, musí se zaokrouhlit. Tím jsou známy souřadnice bodu v původním obrázku jehož vlastnosti se přenesou na souřadnice x',y' nového obrázku. Toto metoda se nazývá nearest neighbor.

Pro lepší viditelnost převzorkování, je lepší zmenšený obrázek zobrazit na stejné ploše, jako ten původní.

I zde je výhodné využít paralelní výpočet, protože podobně jako u přepočítání barevného modelu, musí výpočet probíhát pixel a pixelu a to může u velkého obrázku prodloužit výpočet.

2.2 Implementace, instalace a konfigurace API

Vlastní aplikace byla vyvíjena na sytému Windows 7 Professional SP1 32 bit. Tento systém byl virtualizován na MAC OS X 10.6.7. Snow Leopard, který nativně běžel na Apple MacBook Pro model 2010. Jako virtualizační program byl využit software Paralles Desktop ve verzi 6, jehož virtualizovaná grafika neumí OpenCL. Takže jako OpenCL zařízení musel být použit procesor Intel Core 2 Duo 2,4 Ghz (P8600), který podporuje virtualizaci a bylo tudíž možné využít Intel SDK.

Jako vývojové prostředí bylo použito Microsoft Visual Studio 2010 Professional. V projektu je používána knihovna freeGLUT[1], jejíž obsah bylo nutné zkopírovat do příslušných míst. Soubor **freeglut.dll** bylo nutno nahrát složky windows/system32. Knihovna dále obsahuje lib/**freeglut.lib**, tento soubor bylo nutné nahrát do Program Files/Microsoft Visual Studio 10/VC/lib. Poslední součástí byla složka /include/**GL**, která se celá musí nahrát do Program Files/Microsoft Visual Studio 10/VC/include. Tímto byla zavedena knihovna freeGLUT a je možné ji v projektu používat.

V projektu je také nutné odkázat se na OpenCL soubory. V tomto případě jsou sobory umístěné ve složce, kam se nainstalovalo SDK od Intelu. Obecně se všechny SDK instalují do Program Files. Pravým kliknutím na projekt ve Visual Studiu se zobrazí nabídka, kde je důležitá až poslední položka Properties. V záložce Linker ⇒General ⇒Additional Library Directories je nutné odkázat se soubor **OpenCL.lib**. Z další záložce C/C++⇒General⇒Additional Include Directories je třeba se odkázat na složku **CL**, která obsahuje mimo jiné i soubor **OpenCL.h**. [4] [6]

Tímto krokem je nakonfigurováno vývojové prostředí je možné přistoupit k vývoji samotné aplikace.

2.3 Konfigurace OpenCL

Do těla hlavní aplikace bylo nutné umístit připojení hlavičkových souborů knihovny freeGLUT a knihovny OpenCL. Dále je připojen hlavičkový soubor imageLoad.h, což je program, který se stará o korektní načtení obrázku. Podporovány jsou nekomprimované formáty BMP a TARGA. Umí také z obrázky vytáhnou potřebná data

[1] http://www.martinpayne.me.uk/software/development/GLUT/freeglut-MSVC.zip

jako počet barevných kanálů, výšku a šířku. Tyto hodnoty jsou stěžejní pro další výpočty.

Dále je nutno definovat OpenCL objekty. Ty se definují jako proměnné a jejich seznam, který byl v programu použit, je uveden v příloze A.1. Tyto objekty jsou nutné, protože se používají při konfiguraci OpenCL zařízení a pracují s nimi i další komponenty hostitelského programu.

Další nevyhnutelnou částí je inicializace vlastního OpenCL zařízení. Jako OpenCL zařízení je možné použít jen podporovaný procesor, případně jen podporovanou grafickou kartu, či jiné specializované zařízení. Pokud je potřeba co nejvyšší výpočetní výkon, je možné inicializovat všechny kompatibilní OpenCL zařízení. V praxi se výhodné využít jen grafickou kartu, protože je celá konfigurace mnohem jednodušší a je menší pravděpodobnost, že program zhavaruje. Toto se občas stává u procesorů Intel u kterých ještě není podpora vyladěna na 100%. U inicializovaných zařízení je nutné provést kontrolu, zařízení musí být v platformě, která byla zvolena, a pokud je odkazováno na přesný typ zařízení (CPU, GPU) musí být těmto odkazům vyhověno. [1] [7]

V příloze A.2 je ukázková konfigurace, kde je jako jediné OpenCL zařízení použit procesor, který je sám v jediné ptatformě, a je ověřena jeho možnost použití, pokud je vybrána platforma, které neobsahuje OpenCL kompatibilní zařízení, nebo pokud je odkázáno na jiné zařízení, program je ukončen.

Dále je v této příloze ukazaná inicializace kontextu, fronty příkazů a kompilace kernelu. Kompilace kernelu při každém startu určitou dobu trvá. Nakonec je zkompilovaný kernel přezkoušen, pokud není kompilování v pořádku, je celý program ukončen.

Je možné vytvořit několik kontextů a každý z nich může využít jiné zařízení v různých platformách. Tyto kontexty je možno řadit do několika front příkazů podle potřebné důležitosti. Také vykonávaných kernelů může být více, každý může obsahovat jiný kontext a tudíž může být v jiné frontě příkazů. Tímto způsobem je možné nastavit zásadní vlastnosti paralelního zpracování. [6]

Pro tuto modelovou aplikace je možné použít jen jeden kompilovaný program, jednu frontu a jediný kontext.

2.4 Hostitelská část programu

Tato část programu běží přímo na hostitelském systému. Je vytvořena jako funkce, které je možné předat parametry. V aplikaci je tak jeden program pro přepočítaní mobelu a změnu odstnínu a jeden program pro převzorkování obrazu.Jako příklad je v příloze A.3 uveden program pro převzorkování obrazu.

Na základě zvolených parametrů mx a my je třeba spočítat nová výška a šířka převzorkovaného obrázku.

Následuje určení k jakému programu patří kernel (**clCreateKernel**), který je spojen s tímto hostitelským programem. Další nutnou záležitostí je deklarace proměnné, kam je uložená velikost zpracovávaného obrázku v Bytech (**size_t imageSizeInBytes**). A proměnné **newImageSizeInBytes**, která určuje velikost zmenšeného obrázku. Taty velikosti jsou zásadní, protože určují jak velký bude vstupní (**inputMem - imageSizeInBytes**) a výstupní (**outputMem - newImageSizeInBytes**) buffery, které jsou třeba alokovat na OpenCL zařízení. Vstupní paměť je určena jen pro čtení a výstupní jen pro zápis.

Tyto paměti jsou zařazeny do front. Fronta pro zápis obsahuje vstupní paměť a fronta pro čtení obsahuje výstupní paměť. V tomto hostitelském programu je také možné vytvořit další proměnné, které pak mohou být pomocí funkce **clSetKernelArg** předány kernelu. Pořadí, které je zvoleno v tomto předávání argumentů, je zásadní. Přesně v tomto pořadí je nutno argumenty použít i v kernelu. Nakonec je pomocí **clReleaseMemObject** uvoněna vstupní a výstupní paměť. Poslední řádek je už jen pro vykreslení obrázku pomocí OpenGL a freeGLUT.

Hostitelský program pro převod barevného modelu a změnu odstínu je vytvořen velmi podobně. Jsou předávány jiné argumenty a výstupní buffer je stejně velký jako vstupní, protože velikost obrázku v bytech se nemění.

2.5 Kernel

Tato část kódu je běží na OpenCL zařízení. Hostitelská část programu předá kernelu potřebné argumenty, kernel je zpracuje a pak předá vypočtené hodnoty zpět. Ukázkový kód kernelu je v příloze A.4. Jedná se o kernel, který má na starosti zmíněné převzorkování.

Kernel je nutné zapsat jako textový řetězec a je zde použit kód v jazyce C, jde se o stejný kód, jaký je použit při převzorkování obrazu pomocí procesoru, zasadní orzdíl je, že pro zpracování na procesoru je použit cyklus **for**, který není v kernelu použit. Toto je velké plus pro OpenCL, poměrně jednoduchou úpravou je možné efektivněji využít stávající kódy. [4]

Zápis kernelu začíná vždy **__kernel**. Následuje název kernelu, který musí korespondovat s názvem, který byl použit u hostitelského programu. V kulatých závorkách jsou pojmenovány předané argumenty. Argumenty jsou přesně v takovém pořadí, v jákem je předal hostitelský program. Například. pokud je třetí předávaný paramater šírka obrázku, bude jakkoliv pojmenovaný třetí argument v kernelu obsahovat tuto hodnotu. Je proto vhodné jej pojmenovat **__global width**. Je možné

používat tyto proměnné i lokálně pokud je potřeba.

Ve složených závorkách je poté uveden kód, který se stará o samotný výpočet. Je možné zde definovat další proměnné a používat předané argumenty. Také je možné předané argumenty měnit odevzdávat hodnoty do hostitelského programu. V ukázce kernelu je vidět kód pro výpočet převzorkování.

Druhý kernel obsahuje jiné argumenty a uvnitř je kód pro přepočet na HSV model a zpět se změnou odstínu. Zápis kernelu je totožný.

2.6 Ovládání ukázkové aplikace

Aplikace je koncipována tak, aby se po zkompilování otevřelo okno s testovacím obrázkem a druhé okno, kde se vypisují informace o kompilaci a načítaní obrázku. Kompilace trvá několik vteřin protože se kompiluje kernel, který je prováděn na OpenCL zařízení. Aplikace reaguje na stisk předem definovaných kláves. Po stisknutí přislušné klávesy se vypíše prováděná operace a doba trvání dané operace v milisekundách. Program také umí zoomovat obrázek, je k tomu potřeba kolečko myši.

Klávesa **Z** spouští funkci pro změnu odstínu o 30 stupňů. Dalším stiskem se odstím posune o dalších 30 stupňů. První krok je nastaven v globální proměnné i a další krok je nastaven cyklem ve funkci, která vyhodnocuje stist kláves. Cyklus je ošetřen, ale stupně nepřesáhly 360.

Klávesa **U** provádí také změnu odstínu, ale k obrázku je přistupováno přes dva cykly for a pro výpočet je použit jen procesor. První krok je nastaven v globální proměnné j a další krok je také nastaven cyklem.

Klávesa **I** spouští funkci pro převzorkování obrazu s parametry $mx = 0,2$ a $my = 0,2$. Tyto parametry je možno ve zdrojovém kódu libovolně změnit.

Klávesa **O** provádí převzorkování pomocí procesoru se stejnými defaultními parametry, které je možné nazávisle měnit.

Klávesa **E** načte zpět původní obrázek.

2.7 Výsledky měření času

Modelová aplikace byla otestována na třech různých počítačových sestavách. Jedná se o jeden notebook s integrovanou grafickou kartou a dvě počítačové sestavy s dedikovanými kartami.

Testovací sestava 1: Macbook Pro 2010

- Procesor: Intel Core 2 Duo 2,4 GHz (P8600) (2 jádra)

- Grafická karta: NVIDIA GeForce 320M až 256 MB sdílené paměti (48 výpočetních jader)
- Windows 7 Professional SP1 32 bit

Testovací sestava 2:
- Procesor: Intel Core i3-530 2,93 GHz (2 jádra podporující HT, 4 vlákna)
- Grafická karta: NVIDIA GeForce GT 240, 512 MB vlastní paměti (96 výpočetních jader)
- Windows 7 Professional SP1 64 bit

Testovací sestava 3:
- Procesor: Intel Core i3-540 3,06 GHz (2 jádra podporující HT, 4 vlákna)
- Grafická karta: NVIDIA GeForce GTX 480, 1536 MB vlastní paměti (480 výpočetních jader)
- Windows 7 Professional SP1 64 bit

Měření probíhalo tak, že se každý úkon provedl desetkrát na každé testovací sestavě. Testovaný obrázek má rozměry 1024 x 1024 pixelů. Aritmetickým průměrem byla určena hodnota, která je porovnávána. Měření může být zatíženo chybou, u zpracování procesorem není možné ovlivnit procesy systému, které běží na pozadí a procesor vytěžují. U OpenCL je problém podobný, vzhledem k akcelerovanému desktupu Aero není možné ovlivnit vytížení grafiky. Doba konání OpenCL nepočítá jen čistý čas provádění kernelu, ale také alokaci pamětí na zařízení a nahrávání dat do a ze zařízení. Výsledky tedy odpovídají běženému použití aplikace.

Tab. 2.3: Výsledky měření grafických úprav pomocí CPU a OpenCL zařízení

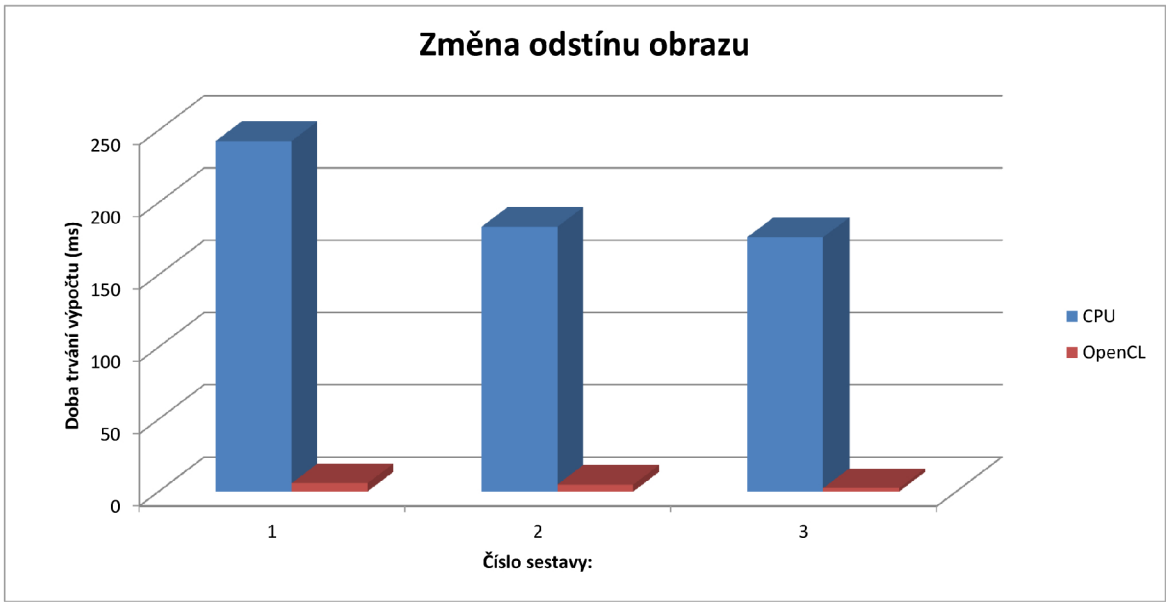
Typ úpravy	Převzorkování ($mx = 0,2, my = 0,2$)		Změna odstínu (posun o 10 stupňů)	
Výpočet pomocí	CPU	OpenCL	CPU	OpenCL
Sestava 1	1,2 ms	0,35 ms	242,5 ms	5,7 ms
Sestava 2	0,91 ms	0,32 ms	183,1 ms	4,7 ms
Sestava 3	0,87 ms	0,21 ms	175,9 ms	2,3 ms

Výsledky uvedené v tab. 2.3 odpovídají teoretickému předpokladu. Díky dobré možnosti paralelizace jsou OpenCL zařízení vždy rychlejší. Rozhodně ale naplatí přímá úměra mezi počtem výpočetních jader a časem potřebným k dokončení výpočtu. Vzhledem k velikosti obrázku a metodě použité k výpočtu není u převzorkování

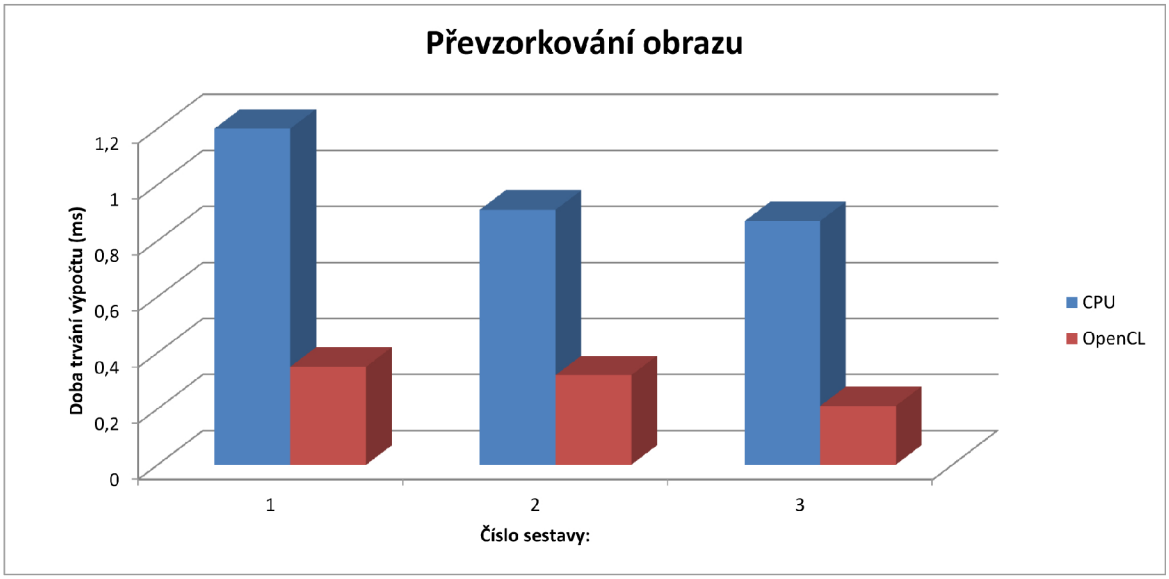
rozdíl mezi procesory a ani rozdíl mezi grafickými kartami nikterak zásadní. Pokud by byl ale použit obrázek s velmi vysokým rozlišením, rozdíly by byly určitě patrnější.

U změny odstínu je rozdíl velmi viditelný. Slabší procesor Core 2 duo je citelně pomalejší než nejrychlejší Core i3 a to o téměř 80 ms. Rozdíly mezi jednotlivými procesory Core i3 jsou velmi malé a to jen díky vyšší taktování jednoho z nich. Nejslabší grafická karta GeForce 320M je o 170 ms rychlejší, než výnější Core i3. Čas 5,7 ms je na integrovanou kartu se sdílenou pamětí opravdu velmi dobrý. Druhá grafická karta GeForce GT 240 sice obsahuje dvojnásobek výpočetních jader a má vlastní paměť, ale je rychlejší jen o 1 ms. Nejrychlejší karta dasáhla času 2,3 ms což potvrzuje, že s počtem jader neroste výpočetní výkon přímo.

Pro větší přehlednost jsou výsledky měření vyneseny do grafů 2.3 a 2.4.



Obr. 2.3: Graf závislosti doby výpočtu změny odstínu na použitém zařízení



Obr. 2.4: Graf závislosti doby výpočtu převzorkování obrázku na použitém zařízení

3 ZÁVĚR

Standard OpenCL je relativně mladou technologií. Díky dobře a přehledně sepsané specifikaci není tak obtížné ji pochopit. Rozdělení do jednotlivých modelů je jasnou a srozumitelnou kostrou. Mezi hlavní výhody patří nezávislost na operačním systému a nezávislost na zařízení jednoho výrobce. Velmi jednoduše se dá vzít stávající kód a použít ho v OpenCL. Pokud je úloha dobře paralelizovatelná, bude celá aplikace mnohem výkonnější.

Velmi pozitivní je také přístup hlavních výrobců hardware, kteří vývojářům nabízí dostatek pomůcek pro vývoj. Rozrůstá se komunita vývojářů, přibývají ukázky kódů a různé specializované knihovny. Obzvlášt hlavní iniciátor vytvoření tohoto standardu, firma Apple, se velmi angažuje a tlačí vývojáře k používání OpenCL.

Konfigurace vývojového prostředí není nikterak složitá. Je vždy potřebné stáhnout podporu od výrobce hardware, který má být pro vývoj použit, a nasměrovat vývojové prostředí k souborům OpenCL.

Vyvíjená aplikace splnila teoretické předpoklady. Zpracování obrázku je velmi dobře paralelizovatelný úkol. Proto byly výsledky OpenCL o mnoho lepší než pro procesor, tím se v práci podařilo splnit všechny zadané úkoly v plném rozsahu.

LITERATURA

[1] MUNSHI A. *Khronos OpenCL API Registry* [online]. 9/30/10 [cit. 2011-05-30]. The OpenCL Specification. Dostupné z WWW: <http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf>.

[2] *Mac OS X Developer Library* [online]. 2009 [cit. 2011-05-30]. OpenCL Programming Guide for Mac OS X. Dostupné z WWW: <http://developer.apple.com/library/mac/#documentation/Performance /Conceptual/OpenCL_MacProgGuide/Introduction/Introduction.html>.

[3] KIRK D. B., HWU W. W. *Programming Massively Parallel Processors* A Hands-On Approach (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA 2010, ISBN 0123814723

[4] *AMD Developer Central* [online]. 10/13/2009 [cit. 2011-05-30]. Image Convolution Using OpenCL™ - A Step-by-step Tutorial. Dostupné z WWW: <http://developer.amd.com/sdks/AMDAPPSDK/documentation /ImageConvolutionOpenCL/Pages/ImageConvolutionUsingOpenCL.aspx>.

[5] RAJMIC P. *Základy počítačové sazby a grafiky* Brno: Fakulta elektrotechniky a komunikačních technologií VUT v Brně, 2011. 121 s.

[6] MUNSHI, A. *OpenCL: Parallel Computing on the GPU and CPU*[online]. 2008 [cit. 2011-05-30]. SIGGRAPH2008. Dostupné z WWW: <http://s08.idav.ucdavis.edu/munshi-opencl.pdf>.

[7] *NVIDIA: Cuda zone* [online]. 2010 [cit. 2011-05-30] Dostupné z WWW: <http://developer.download.nvidia.com/OpenCL /NVIDIA_OpenCL_JumpStart_Guide.pdf>.

[8] *NVIDIA: Cuda zone* [online]. 2010 [cit. 2011-05-30]. OpenCL. Dostupné z WWW: <http://www.nvidia.com/object/cuda_opencl_new.html>.

[9] *Intel Software Network* [online] 2011 [cit. 2011-05-30] Intel OpenCL SDK. Dostupné z WWW: <http://software.intel.com/en-us/articles/opencl-sdk/>.

[10] *AMD network* [online] 2011 [cit. 2011-05-30] OpenCL: The Open Standard for Parallel Programming of GPUs and Multi-core CPUs . Dostupné z WWW: Dostupné z WWW: <http://www.amd.com/US/PRODUCTS/TECHNOLOGIES/STREAM-TECHNOLOGY/OPENCL/Pages/opencl.aspx>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

CPU	Central Processing Unit
GPU	Graphic Processing Unit
OpenCL	Open Computing Language
API	Application Programming Interface
OpenGL	Open Graphics Library
UML	Unified Modeling Language
SIMD	Single Intruction, Multiple Data
SPMD	Single Process, Multiple Data
ARM	Advanced RISC Machine
CUDA	Compute Unified Device Architecture
SDK	Software Development Kit
AVX	Advanced Vector Extensions
fireGL	Fire Graphics Library
Ati APP	Ati Accelerated Parallel Processing
BLAS	Basic Linear Algebra Subprograms
FFT	Fast Fourier transform
RGB	Red, Green, Blue
HSV	Hue, Saturation, Value
HSB	Hue, Saturation, Brightness
freeGLUT	Free OpenGL Utility Toolkit
SSE	Streaming SIMD Extensions
MMX	MultiMedia Extensions
GPGPU	General-purpose computing on graphics processing units
CD	Compact Disk
MP3	MPEG-2 Audio Layer III

SEZNAM PŘÍLOH

A ZDROJOVÉ KÓDY

A.1 Použité OpenCL objekty

```
cl_context context;
cl_context_properties properties[3];
cl_kernel kernel;
cl_command_queue command_queue;
cl_program program;
cl_int err;
cl_uint num_of_platforms=0;
cl_platform_id platform_id[3];
cl_device_id device_id;
cl_uint num_of_devices=0;
cl_mem inputMem, outputMem;
```

A.2 Inicializace OpenCL

```
if (clGetPlatformIDs(3, platform_id, &num_of_platforms)!= CL_SUCCESS)
//ověření zvolené platformy
 {
  printf("Unable to get platform_id\n");
  return 1;
 }
if (clGetDeviceIDs(platform_id[0], CL_DEVICE_TYPE_CPU, 1, &device_id,
&num_of_devices) != CL_SUCCESS)
//ověření zvoleného OpenCL zařízení
 {
  printf("Unable to get device_id\n");
  return 1;
 }
properties[0]= CL_CONTEXT_PLATFORM;
properties[1]= (cl_context_properties) platform_id;
properties[2]= 0;
context = clCreateContext(properties,1,&device_id,NULL,NULL,&err);
command_queue = clCreateCommandQueue(context, device_id, 0, &err);
//vytvoření fronty příkazů a kontextu
printf("Compiling kernel\n");
```

```
program = clCreateProgramWithSource(context,1,(const char **) &ProgramSource,
NULL, &err);
if (clBuildProgram(program, 0, NULL, NULL, NULL, NULL) != CL_SUCCESS)
 {
  printf("Error building program\n");
  return 1;
 }
//kompilace kernelu a následné ověření jeho funkčnosti
```

A.3 Hostitelská část programu pro převzorkování obrazu

```
void runResamCL(float mx, float my)
 {
  novy.height = input.height*my;
  novy.width = input.width*mx;
  //vytvoření nových rozměrů obrázku
  if (novy.data == 0)
   novy.data = createEmptyImage(novy.width, novy.height, input.channels);
  else {
   _aligned_free(novy.data);
   novy.data = createEmptyImage(novy.width, novy.height, input.channels);
  }
  kernel = clCreateKernel(program, "resam", &err);
  size_t newImageSizeInBytes = novy.width*novy.height*3;
  size_t ImageSizeInBytes = input.width*input.height*3;
  //provázání s kernelem vytvoření důležitých proměnných,
  které informují o velikosti původního a nového obrázku
  inputMem = clCreateBuffer(context, CL_MEM_READ_ONLY,
  sizeof(unsigned char) * ImageSizeInBytes, NULL, NULL);
  outputMem = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
  sizeof(unsigned char) * newImageSizeInBytes, NULL, NULL);
  //vytvoření vstupního a výstupního bufferu
  clEnqueueWriteBuffer(command_queue, inputMem, CL_TRUE, 0,
  sizeof(unsigned char) * ImageSizeInBytes, input.data, 0, NULL, NULL);
  cl_int clWidth_novy = novy.width;
  cl_int clChannels = 3;
  cl_float clmx = mx;
```

```
 cl_float clmy = my;
 cl_int clWidth_puvodni = input.width;
 //vstupní fronta bufferu a deklarace několika potřebných proměnných
 clSetKernelArg(kernel, 0, sizeof(cl_mem), &inputMem);
 clSetKernelArg(kernel, 1, sizeof(cl_mem), &outputMem);
 clSetKernelArg(kernel, 2, sizeof(cl_int), &clWidth_novy );
 clSetKernelArg(kernel, 3, sizeof(cl_int), &clChannels);
 clSetKernelArg(kernel, 4, sizeof(cl_float), &clmx);
 clSetKernelArg(kernel, 5, sizeof(cl_float), &clmy);
 clSetKernelArg(kernel, 6, sizeof(cl_int), &clWidth_puvodni);
 //předávání argumentů kernelu v přesném pořadí
 size_t indexCL[2];
 indexCL[0] = novy.width;
 indexCL[1] = novy.height;

 clEnqueueNDRangeKernel(command_queue, kernel, 2, NULL, indexCL,
 NULL, 0, NULL, NULL);
 clFinish(command_queue);
 clEnqueueReadBuffer(command_queue, outputMem, CL_TRUE, 0,
 sizeof(unsigned char) * newImageSizeInBytes, novy.data, 0, NULL, NULL);
 //výstupní fronta bufferu
 clReleaseMemObject(inputMem);
 clReleaseMemObject(outputMem);
 //uvolnění paměti použitých bufferů
 glTexImage2D(GL_TEXTURE_2D, 0,input.channels, novy.width, novy.height,
 0, input.channels, GL_UNSIGNED_BYTE, novy.data);
 //OpenGL kód pro správné vykreslení výskaných dat
}
```

A.4 Kernel pro převzorkování obrazu

```
__kernel void resam(__global uchar *input, __global uchar *output,
__global int width_novy, __global int channels, __global float mx,
__global float my, __global int width_puvodni)
//definice kernelu s jeho jménem a předávanými argumenty v přesném pořadí.
 {
  size_t idx = get_global_id(0);
  size_t idy = get_global_id(1);
```

```
 int x1 = (int)(idx*(1/mx));
 int y1 = (int)(idy*(1/my));
 size_t index_novy = (idx + width_novy*idy)*channels;
 size_t index_puvodni = (x1 + width_puvodni*y1)*channels;
 output[index_novy] = input[index_puvodni];
 output[index_novy+1] = input[index_puvodni+1];
 output[index_novy+2] = input[index_puvodni+2];
 //vlastní kód pro převzorkování
}
```

B OBSAH PŘILOŽENÉHO DVD

Složky a jejich obsah:

- **OpenCL source** - vytvořený projekt pro Microsoft Visual Studio 2010 obsahující všechny zdrojové kódy a původní testovací obrázek.
- **LaTeX source** - zdrojové kódy této práce včetně všech použitých obrázků
- **Vlastní práce** - obsahuje soubor xmichl01.pdf, což je elektronická verze této práce
- **freeGLUT** - soubory knihovny freeGLUT, která je použita v této práci