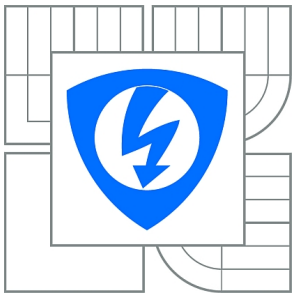
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

VYUŽITIE GRAFICKÝCH PROCESOROV PRE UNIVERZÁLNE VÝPOČTY V PRIEMYSELNÝCH SYSTÉMOCH

GENERAL PROCESSING ON GRAPHICS PROCESSING UNITS FOR INDUSTRIAL SYSTEMS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

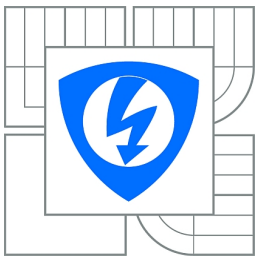
AUTOR PRÁCE — MARTIN LUKAČOVIČ

AUTHOR

VEDOUCÍ PRÁCE — Ing. RADKO KRKOŠ

SUPERVISOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Martin Lukačovič	***ID:***	146894
Ročník:	3	***Akademický rok:***	2013/2014

NÁZEV TÉMATU:

Využitie grafických procesorov pre univerzálne výpočty v priemyselných systémoch

POKYNY PRO VYPRACOVÁNÍ:

Popíšte problematiku univerzálnych výpočtov na grafických čipoch a porovnajte toto riešenie z pohľadu časovej zložitosti, realizačnej náročnosti, ceny HW a energetickej efektivity s riešeniami používajúcimi CPU, DSP či FPGA, analyzujte typické skupiny problémov vhodné pre konkrétne riešenie. Diskutujte architektúru GPU z pohľadu vykonávania vedeckých výpočtov a zamerajte sa najmä na vplyv využitia jednoduchej a dvojitej presnosti na výslednú chybu a výkon výpočtu, možnosti komunikácie GPU so špeciálnym HW ako A/D a D/A karty, podporu priameho prístupu do pamäte DMA. Popíšte framework OpenCL, analyzujte najmä fázy vykonávania programu z pohľadu ich vplyvu na celkovú dobu výpočtu a rozdiely oproti natívnej implementácii. Diskutujte architektúru heterogénnych výpočtových systémov, pamäťové modely NUMA a hUMA.

DOPORUČENÁ LITERATURA:

[1] GASTER, Benedict, Lee HOWES, David R. KAELI, Perhaad MISTRY a Dana SCHAA. Heterogeneous Computing with OpenCL. Amsterdam: Morgan Kaufmann, 2011, xvi, 277 s. Parallel Programming/Computer Architecture. ISBN 978-0-12-387766-6.
[2] KIRK, David B. a Wen-mei W. HWU. Programming Massively Parallel Processors: A Hands-on Approach. 2nd ed. San Francisco, California: Morgan Kaufmann, 2012. ISBN 01-241-5992-3.

Termín zadání:	10.2.2014	***Termín odevzdání:***	4.6.2014

Vedoucí práce: Ing. Radko Krkoš
Konzultanti bakalářské práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

ABSTRAKT

Práca sa zaoberá možnosťami grafických procesorov v oblasti GPGPU. Obsahuje historické riešenia až po súčasné architektúry. Rovnako sú popísané grafické procesory od najväčších súčasných výrobcov, ich zameranie a ciele v budúcnosti. Pre implementáciu algoritmov pomocou GPU sú potrebné API rozhrania, ktoré ponúkajú rôzne možnosti prevedenia. Okrem CPU a GPU sa pre univerzálne paralélne výpočty využívajú i alternatívy ako FPGA a DSP, kedy je potrebné zvážiť cenovú a energetickú náročnosť. V práci je venovaná časť spôsobu komunikácie s hardwarom a moderným pamäťovým prístupom. Pre demonštráciu paralelného výpočtu bola uskutočnená implementácia násobenia matíc v OpenCL.

KĽÚČOVÉ SLOVÁ

GPGPU, GPU, CPU, NVIDIA, Intel, AMD, OpenCL, paralelizácia

ABSTRACT

The thesis deals with the abilities of graphics processors for GPGPU. It contains historical solutions to contemporary design. There are also described graphics processors from the largest manufacturers of this time, their focus and goals in the future. For algorithms implementation using GPU, there are necessary APIs that offer various possibilities of execution. In addition to the CPU and GPU universal heterogeneous computing, there are alternatives such as FPGA and DSP so it is necessary to consider the price and energy cost. Part of the work is devoted to the communication possibilities with the hardware and advanced memory approaches. For demonstrating parallel computing an implementation of matrix multiplication in OpenCL was realized.

KEYWORDS

GPGPU, GPU, CPU, NVIDIA, AMD, Intel, OpenCL, parallelization

LUKAČOVIČ, Martin *Využitie grafických procesorov pre univerzálne výpočty v priemyselných systémoch*: bakalárska práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2014. 59 s. Vedúci práce bol Ing. Radko Krkoš

PREHLÁSENIE

Prehlasujem, že som svoju bakalársku prácu na tému „Využitie grafických procesorov pre univerzálne výpočty v priemyselných systémoch“ vypracoval samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/nebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona č. 121/2000 Sb., o právu autorském, o právoch súvisejúcich s právom autorským a o zmeně niektorých zákonov (autorský zákon), vo znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhé, hlavy VI. diel 4 Trestného zákoníka č. 40/2009 Sb.

Brno ....

(podpis autora)

POĎAKOVANIE

Ďakujem vedúcemu bakalárskej práce Ing. Radkovi Krkošovi za cenné pripomienky a rady pri vypracovávaní bakalárskej práce.

Brno ....

(podpis autora)

OBSAH

Úvod **9**

1 Možnosti grafických procesorov v oblasti GPGPU **10**
1.1 Historické riešenia a súčasnosť 10
1.1.1 NVIDIA GPU 11
1.1.2 Intel GPU 14
1.1.3 AMD APU a GPU 17

2 Aplikačné programové rozhrania **19**
2.1 CUDA 19
2.2 OpenCL 20
2.3 DirectCompute 25
2.4 Stream 26

3 GPGPU a alternatívy pre priemyselné systémy **27**
3.1 CPU 27
3.2 GPU 28
3.3 FPGA 28
3.4 DSP 30

4 Paralelizácia algoritmov **33**

5 Heterogénna architektúra a komunikácia s hardwarom **35**
5.1 Heterogeneous Systems Architecture 35
5.2 Pamäťové modely NUMA a hUMA 37
5.3 Komunikácia s hardwarom a pamäťový prístup 40

6 Praktické meranie **42**

7 Záver **46**

Literatúra **47**

Zoznam symbolov, veličín a skratiek **53**

Zoznam príloh **55**

A Zdrojové kódy **56**

B Obsah priloženého DVD **59**

ZOZNAM OBRÁZKOV

1.1 Štruktúra NVIDIA GeForce 8800 [2] 12
1.2 Vývoj výkonu na zobrazených platformách pri operácii s desatinnou rádovou čiarkou pri jednoduchej a dvojnásobnej presnosti [6] 13
1.3 Vývoj priepustnosti dát pamäte na zobrazených platformách [6] . . . 13
1.4 Systém Sub-slice a common-slice v generácii Haswell [11] 16
2.1 CUDA hierarchia [6] 20
2.2 Organizácia pamäte v OpenCL [24] 25
3.1 Štruktúra FPGA, programovateľné bloky pospájané cestičkami, ktoré umožňujú komunikáciu. I/O (input/output) bloky pre spojenie čipu s vonkajším prostredím [34] 29
3.2 Zjednodušená verzia použitia DSP pri spracovaní analógového signálu [36] 31
3.3 Príklad konkrétneho využitia DSP (MP3 audio prehrávač) [37] 31
4.1 Príklad paralelného počítania, dekompozícia na celky až inštrukcie s využitím viacerých zdrojov 34
5.1 HSA APU platforma zložená z viacjadrového CPU, GPU s viacnásobnými výpočtovými jednotkami H–CUs a pamäťovou riadiacou jednotkou HMMU. Tieto komponenty komunikujú s koherentnou i nekoherentnou pamäťou systému [45] 36
5.2 Porovnanie jednotlivých pamäťových modelov [48] 38
5.3 hUMA model odstraňuje potrebu kopírovania dát a výsledkov z pamäte do pamäte, GPU má možnosť sledovať vložené odkazy [48] . . . 39
5.4 Výhody dosiahnuté hUMA modelom, napr. stránkovateľná pamäť – GPU má bezproblémový prístup k adresám virtuálnej pamäte, ktoré (zatiaľ) nie sú obsiahnuté vo fyzickej pamäti [48] 39
5.5 Periférna operácia 40
5.6 Využitie radiča DMA 41
6.1 Porovnanie časovej náročnosti násobenia dvoch matíc o rôznej veľkosti na uvedených typoch CPU 43
6.2 Porovnanie časovej náročnosti násobenia dvoch matíc o rôznej veľkosti na uvedených typoch GPU 43
6.3 Porovnanie časovej náročnosti prenosu dát do/zo zariadenia pri maticiach o rôznej veľkosti na najrýchlejšom a najpomalšom CPU a GPU 45
6.4 Porovnanie časovej náročnosti násobenia dvoch matíc o rôznej veľkosti na najrýchlejšom CPU a GPU a natívnej funkcie na CPU 45

ZOZNAM TABULIEK

1.1 Porovnanie Intel HD Graphics v počte EU a podpory API [11] 16
1.2 Porovnanie generácií grafických kariet od AMD [21], [22], [23] 18
3.1 Porovnanie medzi architektúrou Trinity firmy AMD a architektúrou Haswell firmy Intel [29], [30] 28
3.2 Porovnanie tzv. diskrétnych GPU od firmy NVIDIA a AMD [31], [32] 28
6.1 Popis použitých CPU na testovanie 42
6.2 Popis použitých GPU na testovanie 42

ÚVOD

Grafické procesory poskytujú okrem bežných úloh ako spracovanie videí, fotografií či akceleráciu 3D grafiky aj širšie využitie ako napríklad rôzne komprimácie, prelamovanie hesiel a teda využívajú sa i v iných oblastiach ako len v oblasti grafických výpočtov. Čím náročnejšie sú úlohy, tým prepracovanejšie aplikácie vznikajú. Práve vykonávaním týchto úloh sa stali grafické procesory neoddeliteľnou súčasťou dnešných výpočtových systémov. Cieľom tejto práce je popis grafického procesora v oblasti paralelných výpočtov, zoznámenie sa s nástrojmi pre implementáciu algoritmov na tomto procesore a jeho porovnanie s inými platformami.

Prvá kapitola sa venuje historickému vývoju, prvému zavedeniu pojmu GPU (Graphics Processing Unit) „grafický procesor“ a stručným porovnaním s CPU (Central Processing Unit) „hlavný procesor počítača“ až po súčasný dizajn. Ďalej je časť textu venovaná variáciám prevedenia grafických procesorov firmami, ktoré v súčasnosti najvýraznejšie ovládajú trh v tejto oblasti.

Druhá kapitola sa venuje API (Application Programming Interface) „rozhranie pre programovanie aplikácií“ vyvinutých pre implementáciu algoritmov na požadovanú platformu. Opisuje sa spôsob akým pracujú, štruktúra a ich výhody / nevýhody.

Tretia kapitola sa zaoberá alternatívami pre priemyselné systémy, menovite CPU, FPGA (Field Programmable Gate Array) „programovateľné hradlové polia“, DSP (Digital Signal Processor) „digitálne signálové procesory“ a ich porovnaním s GPU v oblasti univerzálnych výpočtov, ďalej odlišnosťami pri ich riešení ako je cena, energetická náročnosť ai.

Štvrtá kapitola je venovaná algoritmom, paralelizácii algoritmov, problematike a postupom pre ich správnu efektívnosť a masívnej paralelizácii, ktorá je charakteristická pre grafické procesory.

Piata kapitola predstavuje heterogénnu architektúru, v ktorom je okrem iného zjednotený pohľad na pamäť so zámerom rýchlejšieho prístupu. Súčasťou kapitoly sú opísané pamäťové prístupy NUMA (Non-Uniform Memory Access) „nejednotný prístup do pamäte“ a hUMA (heterogeneous-Uniform Memory Access) „yjednotný prístup do pamäte“, rozdiely medzi nimi a ich využitie, podpora hardwarovej komunikácie s priamym prístupom do pamäte DMA (Direct Memory Access) a činnosť jej radiča.

Po teoretickej časti nasleduje experimentálna časť, kedy bolo pre porovnanie výkonu medzi CPU a GPU, rýchlosti prenosu dát a využitia API implementované násobenie dvoch matíc. Táto časť zahŕňa výpis zariadení, na ktorých bolo meranie uskutočnené, výsledky meraní vynesené do grafov a ich popis.

1 MOŽNOSTI GRAFICKÝCH PROCESOROV V OBLASTI GPGPU

Dôsledkom rozsiahlych výskumov a následného vývoja sa prejavil výrazný nárast výkonu a schopností grafických procesorov. Samotní vývojári sa začali zaujímať najmä o univerzálne výpočty, neskôr začali vznikať rôzne implementácie a vývoj podpory v oblasti API pre GPU.

Ďalším podstatným faktom je, že ich vývoj je rýchlejši ako vývoj samotných CPU. Pokrok technológie vo výrobe či konštrukcie zariadenia pomáha obom platformám, rozdiel sa prejavuje u architektonických častí, kedy vzhľadom na možnosť výpočtu veľkého množstva dát v paralelnéj podobe dokáže GPU využiť dostupné tranzistory efektívnejšie pre dosiahnutie vyššej aritmetickej intenzity pri rovnakom počte tranzistorov ako u CPU. Sú to teda paralelné procesory poskytujúce vysoký výkon. Práve vďaka vývoju či už z pohľadu programovateľnosti alebo zväčšujúcej sa kapacity, obsadzuje pozíciu ako alternatíva ku tradičným mikroprocesorom vo vysokovýkonných výpočtových systémoch súčasnosti i blízkej budúcnosti.

GPU možno označiť za špeciálny procesor poskytujúci 3D grafiku v reálnom čase, vyvinutý na vysokoparalelný systém s množstvom jadier, pracujúcich s veľkým množstvom dát. Vo všeobecnosti platí že čím viac jadier, tým lepší výkon. Neexistujú žiadne obmedzenia v počte jadier, prípadné obmedzenia predstavuje úroveň vedy výrobnej technológie, potreba chladenia, veľkosť čipu či potrebný výkon na jeho prevádzku. Vzhľadom na prácu, ktorú GPU vykonávajú, sú rozdielne vo veľkosti, výkone a taktiež cene. Siahajú od malých jadier vnútri ARM procesoru (napr. NVIDIA Tegra alebo Qualcomm Snapdragon) cez integrované grafické karty v rámci x86 procesoru (napr. AMD Fusion, Intel Sandy Bridge) po samostatne stojace zariadenia alebo dGPU (napr. AMD Radeon, NVIDIA GeForce) [1].

1.1 Historické riešenia a súčasnosť

Pred samotným využitím GPU, bola grafika u klasických stolných počítačov ovládaná pomocou VGA (Video Graphics Array) ovládača. Jeho prácou bolo v podstate príjem dát obrazu, ich správne usporiadanie a zobrazenie na zobrazovacie zariadenie, štandardne monitor počítača. Po roku 1990 boli ku VGA pridávané rôzne komponenty pre zrýchlenie činnosti, až v roku 1999 bol na trh predstavený prvý GPU firmou NVIDIA s názvom GeForce 256. Prvé GPU boli navrhnuté pre sadu jednoduchých funkcií ako trojuholníkové súradnice, nastavenie farby, textúry či veľkosť presvietenia, za účelom efektívneho vytvorenia obrazu. Postupom času sa stali GPU

programovateľnými, čím môžu byť menšie programy spustené pre každé trojuholníkové, vertexové a pixelové spracovanie a to zrýchli generovanie množstva vizuálnych efektov. Vzrastajúcim počtom logických jednotiek určených pre špeciálne účely ako vertexové alebo pixelové spracovanie malo za následok vytvorenie univerzálnych jednotiek, na ktorých by boli vykonávané obe inštrukcie. Taktiež výpočty sa stali presnejšie v čase, v poslednej dobe najčastejšie pridaním double precision floating point – „dvojnásobnej presnosti pri operácii s rádovou desatinnou čiarkou“, ktorá vyjadruje metódu aproximácie reprezentovania reálnych čísiel. V neposlednom rade boli vyvinuté inštrukčné sady a pamäťové nástroje pre podporu univerzálnych programovacíh jazykov ako C a C++. V súčasnosti sú to firmy NVIDIA, AMD a Intel, ktoré sa podielajú na vývoji v oblasti GPU [2].

1.1.1 NVIDIA GPU

Ako už bolo spomenuté vyššie, bola to firma NVIDIA, ktorá priniesla na trh prvé GPU s označením „GeForce 256“. Prvé GPU vykonávali tzv. fixované funkcie a teda z pohľadu programátora, po zadaní a zaslaní dát do zreťazeného spracovania (rúry) GPU, tieto dáta už nemohli byť zmenené. NVIDIA GeForce 7800 bol jeden z prvých GPU, ktoré mali vertexový i pixelový shader. No i v tomto prípade boli jadrá procesoru stalé oddelené ako vertexové procesory a fragmentové (pixelové) procesory. Až NVIDIA GeForce 8800 GTX, patriaci do architektúry G80, bol prvý GPU model, ktorý zjednotil vertexové, fragmentové a geometrické shadery do individuálnych stream procesorov. Odlišnosti medzi procesormi boli odstránené a nahradené jednotným SM (Streaming Multriprocessor), viď obr. 1.1.

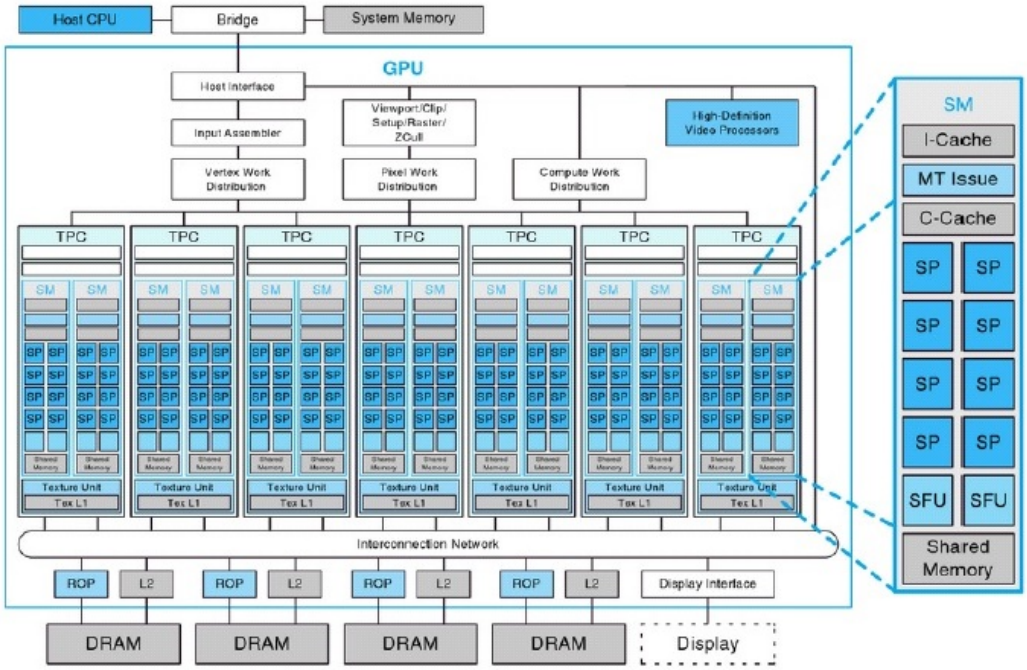
Ďalšou dôležitou generáciou bola architektúra *Fermi*, ktorá predstavovala najväčší skok od generácie G80. Prvé GPU na architektúre Fermi obsahovali 3 miliardy tranzistorov, 512 CUDA jadier, ktoré boli umiestnené v 16 SM. Spomínané SM obsahovali 32 jadier. Vďaka všetkým inováciám je architektúra Fermi 4,2krát rýchlejšia pri double precision formáte v porovnaní s predchádzajúcou 10-sériou s označním G200 firmy NVIDIA [3].

Nástupcom architektúry Fermi je architektúra *Kepler*, u ktorej sa uskutočnili opäť zmeny, najmä v počte tranzistorov (7,1 miliardy), nástupom ďalšej generácie SM pod názvom SMX, dynamickými paralelizáciami, aplikovaním HYPER-Q, čo umožňuje viacerým jadrám CPU pracovať na práve jednom GPU [4].

V roku 2014 bola uvedená architektúra *Maxwell*. Niektoré vylepšenia, ktoré boli navrhnuté už v tejto architektúre boli posunuté na ďalšiu generáciu s názvom *Pascal*. Uvedená bola ako architektúra medzi Maxwell a Volta, plánovaná je na rok 2016. Medzi vylepšenia patrí zjednotený virtuálny pamäťový priestor – čiže rýchlejší prístup k dátam pre CPU i GPU, 3D pamäť – naskladanie DRAM čipov do mo-

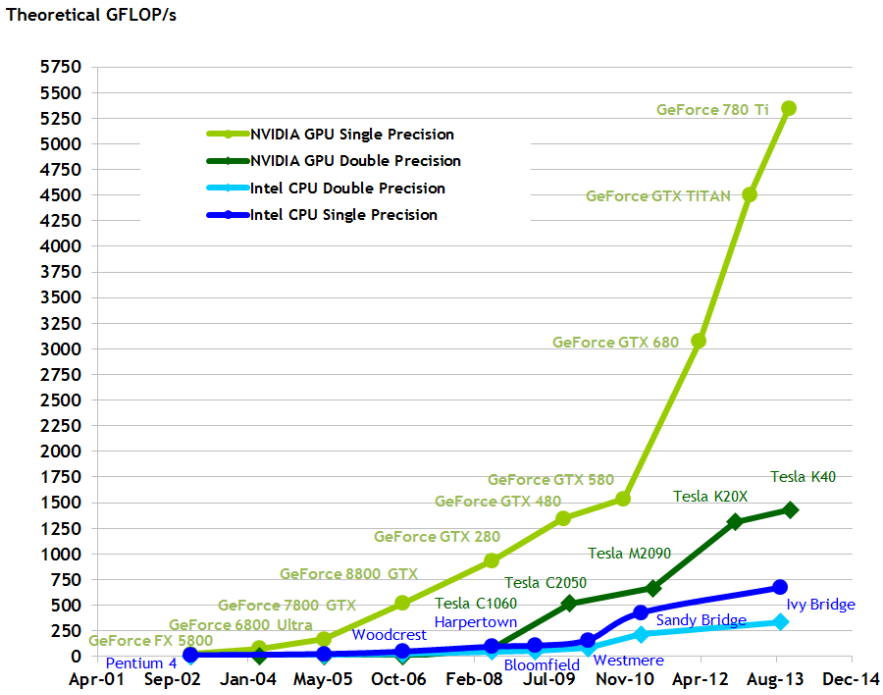
dulu, ktorý bude umiestnený priamo so sadou GPU, čo prinesie väčšiu priepustnosť, efektivitu, veľkosť pamäte a rovnako lepšiu energetickú účinnosť.

Zaujímavou novinkou je, že NVIDIA má snahu nahradiť PCI-Expres zbernicu svojou vlastnou rýchlejšou zbernicou s názvom NVLink, pre dosiahnutie požadovaného a najmä efektívneho využitia šírky pásma pamäte. PCIe zbernica však bude stále využívaná a aj NVLink zbernica bude využívať pre posielanie niektorých príkazov zbernicu PCIe. Okrem zefekívnenia komunikácie medzi viacerými GPU je snaha zo strany NVIDIA dosiahnuť rovnaké podmienky pri komunikácii medzi CPU a GPU. Využitie tejto zbernice sa predpokladá skôr na serveroch ako u bežných užívateľov. Realitou sa však stane najskôr až pri generácii Pascal. Do budúcnosti je ohlásená architektúra Volta, dátum jej vydania však nie je známy. [5].

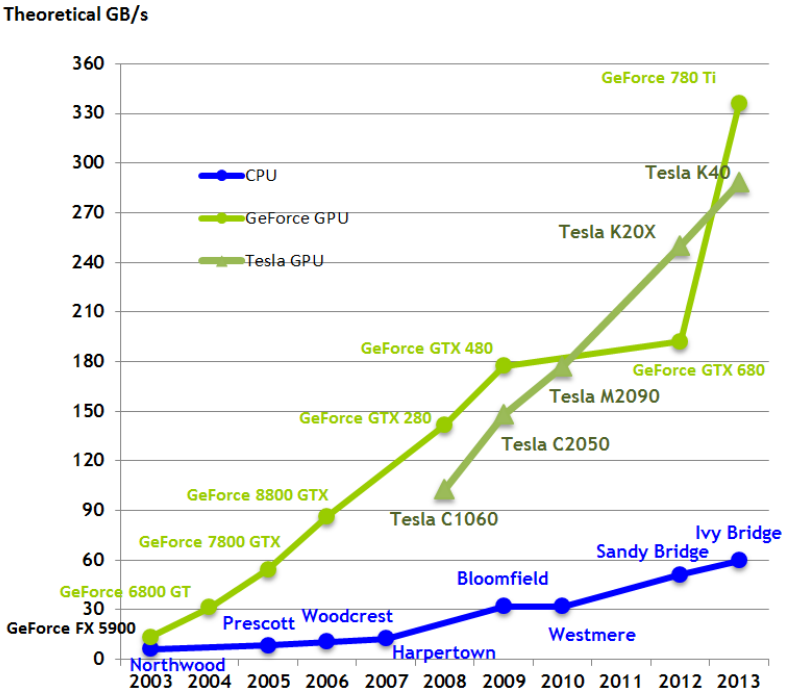


Obr. 1.1: Štruktúra NVIDIA GeForce 8800 [2]

Na obr. 1.2 a obr. 1.3 sú zobrazené rozdiely vo výkone a dátovej priepustnosti medzi CPU a NVIDIA GPU.



Obr. 1.2: Vývoj výkonu na zobrazených platformách pri operácii s desatinnou rádovou čiarkou pri jednoduchej a dvojnásobnej presnosti [6]



Obr. 1.3: Vývoj priepustnosti dát pamäte na zobrazených platformách [6]

1.1.2 Intel GPU

Po vynájdení procesora x86 firmou Intel, boli samotné CPU nazývané „mozgami počítačov“. Po vývoji GPU sa však u CPU ukázali slabšie stránky a preto sa niektoré firmy rozhodli spojiť tieto dve architektúry do jednej. APU (Accelerated Processing Unit) je teda označenie pre jednotku, ktorá zlučuje CPU a GPU do jedného čipu za zámerom väčšej efektivity, výkonu a zrýchlenia vykonávania výpočtov a úloh. APU rovnako dokáže dodať rovnaký výkon ako diskrétne grafické karty nižších tried pri spotrebe omnoho menšej energie. Prvé APU bolo na trh uvedené firmou Intel a AMD, jednoduchým pridaním jadier grafického procesora do CPU architektúry. Aj napriek využívaniu APU jednotiek inými firmami, v súčasnosti termín APU využíva na trhu iba firma AMD pre svoje produkty [7].

Intel HD Graphics 2000 a 3000

Intel HD Graphics 2000 a 3000 je názov pre integrované grafické procesory firmy Intel. Architektúra grafického jadra je veľmi podobná prvej generácii Ironlake, preto pomenovanie Intel HD Graphics ostalo a pridali sa číselné označenia 2000 a 3000. Predošlé architektúry:

Clarkdale, obsahuje 32nm procesor a 45nm grafický ovládač s pamäťovým rozhraním, využívaný v počítačoch, známy pod názvom Intel Core i5, Core i3 a Pentium.

Arrandale rovnako obsahuje 32nm procesor a 45nm Intel HD Graphics, využívaný v mobilných zariadeniach, známy pod názvom Intel Core i7, Core i5, Core i3 alebo Celeron a Pentium [8].

Precíznejšia integrácia viedla k umiestneniu grafického jadra na rovnaký digitálny integrovaný obvod k ostatným funkčným jednotkám CPU, vďaka čomu môžu priamo komunikovať. Tieto nové grafické jadrá sú výkonnejšie a zároveň veľmi energeticky úsporné, preto sa využívajú v notebookoch a v niektorých energeticky úsporných počítačoch. Popisovaná druhá generácia je známa pod názvom *Sandy Bridge*, zlučuje jadrá Intel HD Graphics 2000 alebo Intel HD Graphics 3000 s niektorým z radu Core procesorov [9].

Významné zlepšenie výkonu vyplýva zo zmien vykonaných vo vnútornej štruktúre CPU. Zavedením 32nm procesora a umiestnením výpočtových jadier, grafického jadra, pamäte cache a pamäťového ovládača v rámci rovnakého obvodu pomohlo ku relatívne rýchlejšej práci s pamäťovým podsystémom. Všetky vnútorné procesorové jednotky sú spojené cez kruhovú zbernicovú topológiou vrátane integrovaného grafického jadra, vďaka čomu už nepracuje toto jadro priamo so systémovou pamäťou ale rovnako ako výpočtové jadrá, pomocou vysokorýchlostnej L3 pamäte (6–8 MB), vďaka čomu sa zrýchli napríklad textúrovanie.

Vylepšené boli aj EU (Execution Units), ktoré majú takmer dvojnásobok šírky pásma a preto aj lepšie paralelné výkony. Akcelerácia grafického jadra narastá v prípade že jadrá procesoru nie sú plne vyťažené a naopak klesá ak by hrozilo prekročenie príkonového limitu z dôvodu veľkého zaťaženia. Rozdiel medzi Intel HD Graphics 2000 a 3000 je, že HD grafika 2000 má 6 vykonávacích jednotiek oproti HD grafike 3000, ktorá ich má 12 [9].

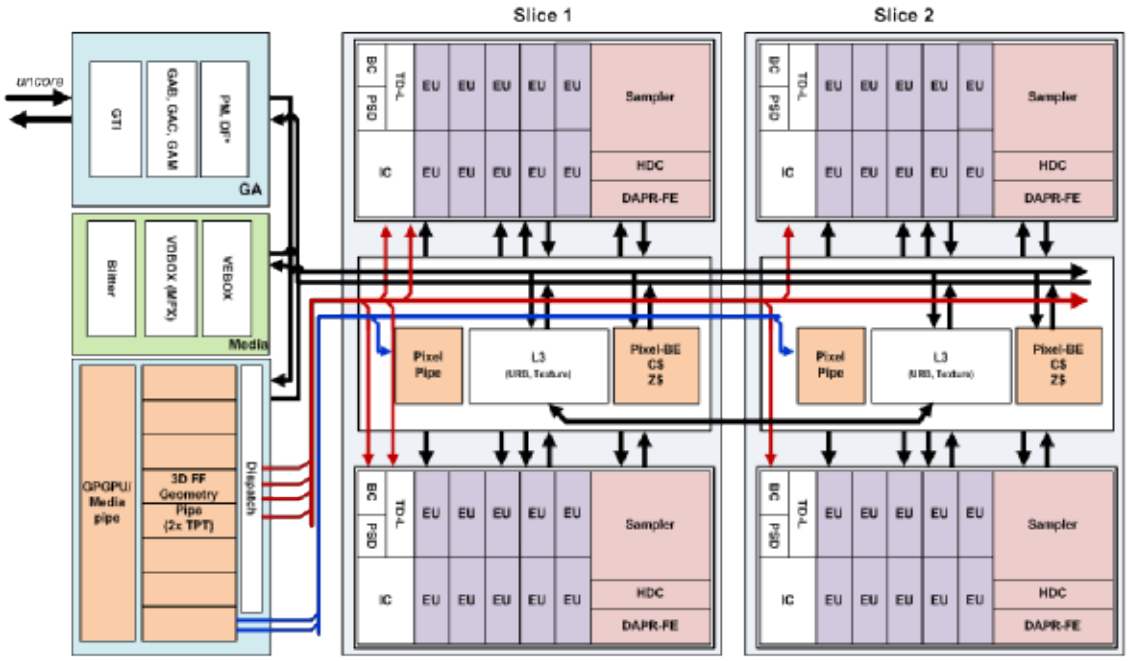
Intel HD Graphics 2500 a 4000

Ivy Bridge procesory predstavujú tretiu generáciu 22nm procesorov, ktoré nastupujú po generácii Sandy Bridge. Prvé rozdiely možno utvrdiť už v samotnom dizajne architektúry, kde sa zvýšil počet tranzistorov (1,4 miliardy) a tým sa zväčšila aj plocha, ktorú grafické jadro zaberá. Firme Intel v tomto prípade nešlo o vylepšenie jadra z pohľadu širšieho využitia výpočtov ale najmä na všeobecné zlepšenie výkonu tak aby nebolo potrebné využitie samostatne stojacích grafických kariet, najmä v mobilných zariadeniach. Rovnako ako v Sandy Bridge, komponenty procesora vrátane grafického jadra sú spojené kruhovou zbernicou, čo zrýchli komunikáciu s pamäťou systému pomocou L3 pamäte a zredukuje sa oneskorenie, spôsobené čakaním na doručenie dát. Jadro Intel HD Graphics 4000 rovnako ako HD Graphics 2500 dokáže pracovať nezávisle s 3 displejmi pri implementácii čipových sád 7 série. HD Graphics 4000 obsahuje 16 vykonávacích jednotiek, zatiaľ čo HD Graphics 2500 len 6 a je dva krát rýchlejší ako jeho predchodca HD Graphics 3000 v oblasti 3D výkonu. Tieto silnejšie jadrá sú požadované v mobilných zariadeniach, u stolových počítačov sa používa HD Graphics 4000 iba v kombinácii s procesormi od rady Core i7, prípadne taktované modely procesorov rady Core i5. Je zrejmé, že tieto jadrá sú viac zamerané na prácu v mobilných zariadeniach, čo dokazuje aj vyššia frekvencia v porovnaní s použitím u stolových počítačov či notebookov [10].

Intel HD Graphics 4200,4400,4600,5000 a rada Iris

Nástupcom Ivy Bridge je štvrtá generácia s názvom *Haswell*, so zameraním na grafické jadro pre novú generáciu procesorov. Kľúčovou výhodou je dostupnosť pre univerzálne výpočtové zariadenia. Svoje miesto si nájde od tabletov až po výkonné počítače, navrhnuté napríklad pre vedecké účely. Vzhľadom na podobnosť s predošlou generáciou, boli grafické jadrá rozdelené pod niekoľko označení. GT2 predstavuje základnú verziu s 20 vykonávacími jednotkami, na trhu pod názvom HD Graphics 4600, zároveň predstavuje jedinú verziu grafického jadra implementovaného u rady Haswell, využívanej v stolných počítačoch. GT3 a GT3e poskytujú väčší 3D výkon, počet vykonávacích jednotiek sa zdvojnásobil na 40. Na trhu sú známe pod označením Iris 5100 a Iris Pro 5200 avšak do bežných počítačových procesorov nebudú

implementované. Je zachovaná snaha Intelu o vyšší výkon a zároveň menšiu spotrebu, čo rovnako predstavuje časť zmien v oblasti ovládania výkonu. Zmenšil sa čas potrebný na prechod do a zo „spánkového“ režimu. V porovnaní s predchádzajúcou generáciou bola pridaná ďalšia ALU (Aritmetic-Logic Unit), ktorá vykonáva základné a logické operácie s celými číslami (predošlá generácia obsahovala len 3). Okrem toho obsahujú GT3e/Iris Pro 128 MB eDRAM, ktorá tvorí samostatnú časť avšak v jednom balíku s ostatnými komponentami procesora a pracuje ako zdieľaná L4 vyrovnávacia pamäť [11].



Obr. 1.4: Systém Sub-slice a common-slice v generácii Haswell [11]

Typ	**Intel HD Graphics 4000**
API / EU	DirectX 11.0, OpenGL 3.x, OpenCL 1.1 / 16
Typ	**Intel HD Graphics 4200/4400/4600**
API / EU	DirectX 11.1, OpenGL 4.0, OpenCL 1.2 / 20
Typ	**Intel HD Graphics 5000/Iris Pro 5200/Iris 5100**
API / EU	DirectX 11.1, OpenGL 4.0, OpenCL 1.2 / 40

Tab. 1.1: Porovnanie Intel HD Graphics v počte EU a podpory API [11]

1.1.3 AMD APU a GPU

Firma AMD sa zaoberá vývojom APU jednotiek od roku 2006, po tom čo odkúpila ATI. Prvá generácia bola predstavená v roku 2011 pod názvom *Llano* (32nm), zameraná pre počítače a notebooky. Táto generácia predstavuje spojenie procesoru K10 a GPU série Radeon HD 6000 v FM1 sokete. Rovnako ako procesory firmy Intel či iné pokročilejšie procesory využívajú K10 „Out of order“ metódu pre efektívnejšie využitie dostupného výpočtového výkonu, teda spracovanie inštrukcií v inom poradí ako uvádza program uložený v operačnej pamäti. Procesor sa sám rozhodne a poskladá inštrukcie pre ich vykonanie v čo najkratšom čase [7].

Pre zariadenia s menšou náročnosťou napájania ako netbooky alebo ultra tenké notebooky či spotrebnú elektroniku boli vyvinuté APU pod názvom *Brazos*, ktoré zlučujú Bobcat procesor a GPU série Radeon HD 6000 umiestnených na rovnakom bloku obvodu, v najčastejšej variante v podobe dvoch jadier Bobcat procesoru a kompatibilného grafického DirectX 11 jadra mobilnej Radeon HD 6000. AMD sa snaží najmä o procesory, ktoré sú veľmi úsporné z hľadiska priestoru i energetickej náročnosti, do ktorých začleňujú grafické jadrá a výsledná jednotka dostáva názov APU. Grafické jadrá sú kompatibilné s API ako OpenCL, DirectCompute, ako aj hardvérovým HD-video dekóderom UVD3, vďaka čomu grafické jadrá zbavia CPU niektorých aplikácií a zlepší sa celkový výkon systému [12].

Druhá generácia bola predstavená o rok neskôr pod názvom *Trinity* (32nm) pre počítače a notebooky. Dosahuje dvojnásobný výkon ako prvá generácia Llano. Zlučuje procesor Piledriver a GPU série Radeon HD 7000 v FM2 sokete. Podporuje API DirectCompute, OpenCL a OpenGL. Pre zariadenia s menšou náročnosťou napájania boli vyvinuté APU pod názvom Brazos 2,0 , ktoré zlučujú opäť Bobcat procesor ale s GPU série Radeon HD 7000 umiestnených na rovnakom bloku obvodu [13].

Tretia generácia pre vysokovýkonné zariadenia bola sprístupnená v roku 2014 pod názvom *Kaveri* (28nm), založeného na architektúre Steamroller, ktorá je zameraná na lepšiu paralelizáciu – tri ALU jednotky na každé jadro, dynamická 4 MB L2 vyrovnávacia pamäť ai. Okrem toho zahŕňa 128-bitový pamäťový radič pre podporu DDR3 a GDDR5 pamätí, čím sa zväčší šírka pásma pamäte [15]. *Temash* (pre tablety) a *Kabini* (pre notebooky a mini PC) boli vydané v roku 2013 – druhé menované Kabini G-série zlučujú 2–4 jadrové procesory a GPU série Radeon HD 8000. Tieto APU sú využivané na zlepšenie 3D grafiky a iných grafických úloh a pre univerzálne výpočty, môžeme ich nájsť v Sony PlayStation 4 či Microsoft Xbox One [14].

U rady *Trinity, Kaveri* a plánovanej *Carizzo* na rok 2015, sa využíva AMD HSA (Heterogeneous System Architecture), čo je systémová architektúra, ktorá povoľuje akcelerátorom, ako je napríklad grafický procesor, pracovať na rovnakej úrovni pra-

covania s dátami ako systémové CPU, teda poskytuje jednotný pohľad na základné výpočtové prvky. K dispozícii je jednotný adresový priestor prístupný pre CPU i GPU, čím sa odstraňuje nutnosť alokácie zdrojov pre prenos a redukuje sa oneskorenie pri ich komunikácii. Bližší popis HSA je v kapitole 5.1.

Doterajší vstupno/výstupný radič známy pod názvom Southbridge sa v APU označuje ako FCH (Fusion Control Hub). Od architektúry Carizzo je umiestnený na čipe, čím sa uštrí niekoľko wattov na spotrebe, katalógové TDP je 65 W, pričom táto hodnota bude nastaviteľná. [15].

AMD Radeon

Po odkúpení výrobcu grafických kariet ATI Technologies, ktorý dovtedy vyvíjal grafické karty pod názvom ATI Radeon bola v roku 2010 predstavená rodina grafických kariet *Northern Islands*, ktorá bola predávaná výhradne pod názvom AMD Radeon. Zlučovali šiestu generáciu grafických kariet Radeon HD [17].

Začiatkom roka 2012 bola vydaná grafická karta rodiny *Southern Islands*. Zlučuje siedmu a prvé typy z ôsmej generácie grafických kariet Radeon HD.

Rodina *Sea Islands* zlučuje ôsmu generáciu grafických kariet Radeon HD a Radeon HD 7790 [18].

Volcanic Islands zlučuje generáciu Radeon R7 2xx a R9 2xx. Rovnako ako u siedmej generácie Radeon HD je tu využívaná architektúra Graphics Core Next, ktorá je zameraná na GPGPU, za zámerom lepšieho využitia zdrojov GPU pre maximálny výkon, zároveň obsahuje väčší počet tranzistorov [19].

Na rok 2015 je plánovaná rodina *Pirates Islands* [20]. GPU uvedené v tab. 1.2 podporujú API ako DirectX, OpenGL, OpenCL i operácie s plávajúcou desatinnou čiarkou s dvojnásobnou presnosťou [21].

GPU	**Príslušná rodina**	**Výkon pri Single Precision**
AMD Radeon HD 6970	Northern Islands	2,7 TFLOPS
AMD Radeon HD 7950	Southern Islands	2,87 TFLOPS
AMD Radeon HD 8950	Sea Islands	3,31 TFLOPS
AMD Radeon R7 260	Volcanic Islands	1,54 TFLOPS
AMD Radeon R9 290	Volcanic Islands	4,85 TFLOPS

Tab. 1.2: Porovnanie generácií grafických kariet od AMD [21], [22], [23]

2 APLIKAČNÉ PROGRAMOVÉ ROZHRANIA

API predstavuje sadu inštrukcií, protokolov a nástrojov ktoré má programátor k dispozícii a špecifikuje ako jednotlivé softvérové komponenty majú medzi sebou komunikovať.

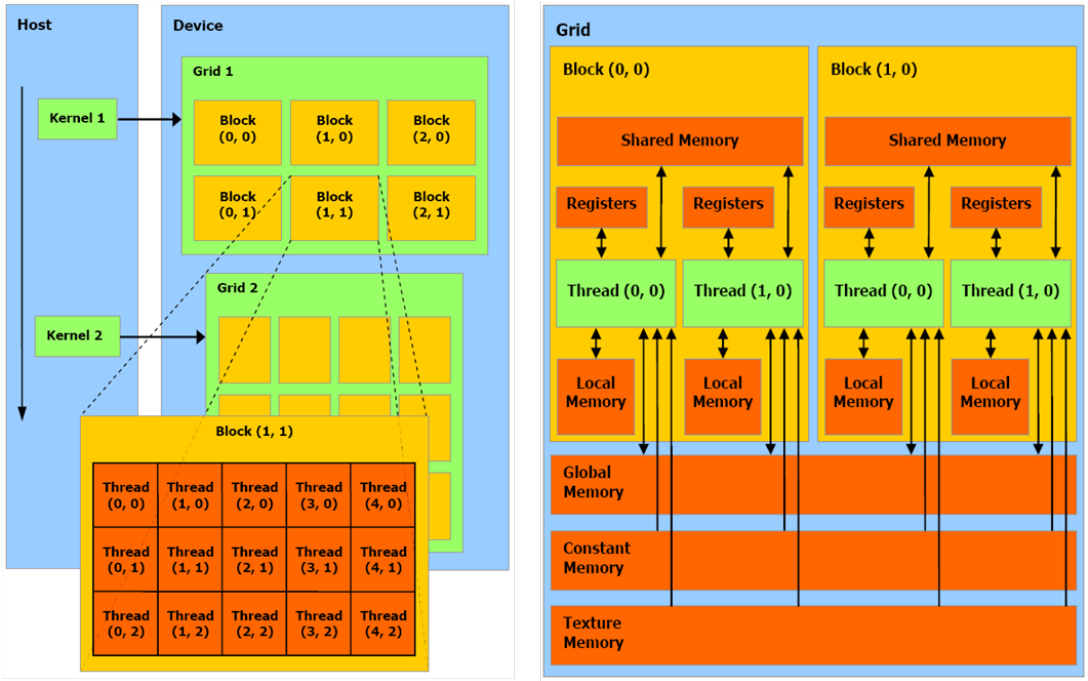
2.1 CUDA

CUDA (Compute Unified Device Architecture) je paralelná výpočtová platforma a programovací model firmy NVIDIA, ktorý v roku 2006 konečne eliminoval potrebu grafického API pre výpočtové aplikácie. Podporuje heterogénne výpočty, kedy dané aplikácie využívajú CPU i GPU. Sériové časti aplikácie pracujú na CPU a paralelné časti sú prevedené na GPU. S CPU a GPU sa zaobchádza ako so samostatnými rozdelenými jednotkami, ktoré majú vlastný pamäťový priestor. Takéto rozdelenie poskytuje súčasné výpočty na oboch platformách bez rozporu o pamäťové zdroje. Rozšírením štandardného jazyka C vznikol CUDA C, ktorý povoluje priamo implementáciu paralelných algoritmov. CUDA tak umožňuje GPU vlastniť stovky jadier, na ktorých sú spoločne vykonávané tisícky vlákien.

Hlavný program sa vykonáva na CPU (hostiteľ) a CUDA program na GPU akcelerátore. Hlavný alebo aj „hostiteľský“ program volá kernel (výpočtové jadro), ktoré sa paralelne vykonáva na GPU. Pre menšie oneskorenie sa používajú bloky, ktoré obsahujú viac vlákien. Samotný počet vlákien v jednom bloku je v kerneli označovaný ako „M“ a počet blokov ako „N“. Bloky vlákna môžu byť jedno/dvoj/troj–zložkové a ID vlákna charakterizuje premenná *threatldx*. Samotné bloky sú umiestňované do tzv. matíc alebo mriežok, vďaka čomu sú bloky spracovávané súčasne [6].

Vlákna počas vykonávania pristupujú k rôznym pamätiam pričom každé vlákno má vlastnú pamäť a register. Každý blok má zdieľanú pamäť, viditeľnú práve pre vlákna daného bloku, vďaka čomu môžu spolupracovať. Globálna pamäť je prístupná všetkým vláknam rovnako ako CPU. Pamäte Constant a Texture sú určené len na čítanie a optimalizované pre rôžne charakteristické pamäťové využitie, sú viditeľné všetkým vláknam, viď obr. 2.1.

CUDA obsahuje rozsiahlu sadu inštrukcií, zároveň NVIDIA zachováva kompatibilitu tým, že programy vytvorené v starších verziách CUDA budú opäť spustiteľné v novších verziách. Najväčšou nevýhodou je, že NVIDIA CUDA je navrhnutá len pre zariadenia od firmy NVIDIA. V roku 2014 bola vydaná najnovšia verzia CUDA 6.0 [6].



Obr. 2.1: CUDA hierarchia [6]

2.2 OpenCL

OpenCL (Open Computing Language) je štandard pre univerzálne výpočty pre rôzne heterogénne platformy ako CPU, GPU a iné procesory. Obsahuje jazyk (využíva podmnožinu C99 ISO s rozšírením pre paralelizmus), API a knižnice, poskytuje možnosť spolupráce s grafickými API ako napr. OpenGL alebo DirectX. OpenCL vyvíja neziskové konzorcium s názvom Khronos Group, ktoré financujú samotní členovia z rôznych firiem. Ich cieľom je vyvinúť API otvorený štandard pre paralelné výpočty spustiteľný na rôznych zariadeniach.

OpenCL architektúru možno rozdeliť na tri modely [24]:

1. Platformový model
2. Pamäťový model
3. Vykonávací model

Platformový model

Tento model pozostáva z „hostiteľa“ pripojeného na jeden alebo viac OpenCL zariadení. OpenCL zariadenie sa rozdeľuje na jedno alebo viac výpočtových jednotiek (compute units), ktoré sa ďalej rozdeľujú do jednej alebo viacerých častí spracovania (processing elements). OpenCL aplikácia je implementovaná ako hostiteľský kód a kód zariadenia (kernel). Hostiteľský kód aplikácie sa vykonáva na procesore, podľa

určeného modelu platformy. Kód hostiteľa aktivuje kernely, ktoré sa vykonávajú paralelne na príslušnom výpočtovom zariadení.

Vykonávací model

Tento model opäť pozostáva z dvoch odlišných jednotiek: kernely, ktoré sa vykonávajú na jednom alebo viacerých OpenCL zariadeniach a hostiteľského programu, ktorý sa vykonáva na hostiteľovej strane. Hostiteľ využíva OpenCL API na vytvorenie a spravovanie kontextu (kontext určuje prostredie, v ktorom budú kernely vykonávané). Funkcie z OpenCL API povoľujú hostiteľovi vzájomne spolupracovať so zariadením pomocou príkazovej fronty. Ak príde príkaz na vykonanie kernelu je definovaný index a funkcia kernelu bude vykonaná pre každý bod určený v indexovom priestore. Tieto funkcie kernelu sú nazývané pracovné prvky a sú umiestnené v pracovnej skupine. OpenCL využíva indexovanie nazývané NDRange, čo je N-dimenzionálny indexový priestor, kde N môže byť jedna, dva alebo tri.

Pamäťový model

Pamäť hostiteľa – je pamäť určená priamo pre hostiteľa, detailnejšie správanie tejto pamäte je určené mimo OpenCL. Pamäťové objekty sa pohybujú medzi hostiteľom a zariadením prostredníctvom funkcií vnútri OpenCL API alebo cez rozhranie zdieľanej pamäte. Pamäť zariadenia – je pamäť určená priamo pre kernely vykonávajúce sa na zariadení, člení sa na viac druhov:

- Globálna pamäť – poskytuje read/write práva, teda „čítať/zapisovať“ pre všetky prvky v akejkoľvek skupine zariadení uvedených v kontexte.
- Konštantná pamäť – vrstva globálnej pamäte, ktorá zostáva konštantná v prípade vykonávania kernelu.
- Lokálna pamäť – pamäť pre pracovné skupiny, používa sa napríklad pre alokovanie premenných, zdieľaných všetkými pracovnými prvkami v rámci skupiny. Považuje sa za najužitočnejšiu pamäť vďaka efektívnemu spôsobomu komunikácie pracovných prvkov v skupine.
- Privátna pamäť – privátna pamäť pracovného prvku, premenné definované práve jedným prvkom nie sú viditeľné pre iný prvok [24].

OpenCL aplikácie často pracujú s poľami veľkých rozmerov alebo multidimenzionálnymi maticami. Skôr ako bude možná realizácia nejakého výpočtu, musia byť tieto dáta fyzicky prítomné na zariadení. K tomu, aby mohli byť prenesené do zariadenia musia byť najskôr zapúzdrené ako tzv. pamäťový objekt. Rozdeľujú sa do troch skupín.

- Buffer

Je typ univerzálneho pamäťového objektu, používaného ako blok súvislej pamäte kde sú uchovávané dáta použité v OpenCL programe. Hodnoty uložené v tejto vyrovnávacej pamäti môžu byť štandardného skalárneho typu dát (napr. integer alebo float), vektorového typu, prípadne užívateľom definovanej štruktúry a sú sprístupnené pomocou ukazovateľov. Vyrovnávacia pamäť je k dispozícii (zdieľaná) medzi všetky kernely. Buffer (zásobník) môže byť deklarovaný v globálnej pamäti (_global) alebo v konštantnej pamäti (_constant). Vytvára sa pomocou funkcie:

```
cl_mem clCreateBuffer(   cl_context context,
                         cl_mem_flags flags,
                            size_t size,
                          void *host_ptr,
                        cl_int *errcode_ret)
```

context – určuje konkrétny kontext správy, v ktorom je buffer vytvorený.
flags – je bitové pole, ktoré sa používa pre určenie alokácie, definovanie vlastností bufferu (napr. iba čítanie, iba zápis alebo ich kombinácia) a pre určenie ďalších možností pri jeho vytváraní. Príznaky (flags) môžu nadobúdať rôzne stavy, ak je však hodnota rovná 0, používa sa štandardne `CL_MEM_READ_WRITE`.
size – je veľkosť pamäte bufferu, ktorá má byť alokovaná (pridelená). Udáva sa v bytoch.
host_ptr – je ukazovateľ na dáta bufferu, ktoré už môžu byť pridelené aplikácii. Veľkosť pamäťového priestoru, na ktorý tento pointer ukazuje, musí byť minimálne rovnako veľký ako hodnota predchádzajúceho parametru „size“.
errcode_ret – vracia chybový kód označujúci chybu pri volaní funkcie. Ak je nastavený na hodnotu „NULL“ nevracia žiadnu chybu kódu.
OpenCL zároveň poskytuje príkazy pre prenos dát z a do zariadenia:
`clEnqueueWriteBuffer` – kopírovanie dát od hostiteľa k zariadeniu.
`clEnqueueReadBuffer` – kopírovanie dát zo zariadenia k hostiteľovi.

Pri spustení dvoch kernelov na dvoch rôznych zariadeniach (napríklad dve GPU) nezávisle na sebe, sú požadované oddelené buffre pretože paralelné spracovanie môžu vykonávať len ak tieto buffre nezdieľajú. OpenCL podporuje vytváranie sub-bufferov (tzv. podružnej vyrovnávacej pamäte) pomocou funkcie `clCreateSubBuffer`. Používa sa pre rozdelenie jednotlivých bufferov do niekoľkých menších bufferov, ktoré sa môžu prekrývať a rovnako môžu byť používané podobným spôsobom ako ich rodičovské pamäťové objekty [24].

- Image

Je typ pamäťového objektu, určený pre uloženie jedno-dvoj-trojrozmerných textúr, polí či obrazov. Image predstavuje nepriehľadnú štruktúru dát, preto sa nepoužívajú ukazovatele ale štruktúra je riadená funkciami definovanými v OpenCL API. Minimálny počet častí v objekte image je jedna a sú vyberané z predefinovaných formátov. Pre lepšie zaobchádzanie s obrazmi, uloženými v textúrovej pamäti, ktoré sú súčasťou mnohých GPU, má programátor pomocou synchronizácie a bariérových operácií možnosť zosúladiť kód tak, aby mohol kernel čítať/zapisovať jeden obraz zároveň. Takáto možnosť je až od verzie OpenCL 2.0. Image nesie informácie ako rozmery obrazu, popis každého prvku v obraze, samotné dáta v podobe obrazu a sú pridelené globálnej pamäti. Pamäťový objekt image je vytvorený pomocou funkcie:

```
cl_mem clCreateImage(              cl_context context,
                                   cl_mem_flags flags,
                          const cl_image_format *image_format,
                            const cl_image_desc *image_desc,
                                     void *host_ptr,
                                  cl_int *errcode_ret)
```

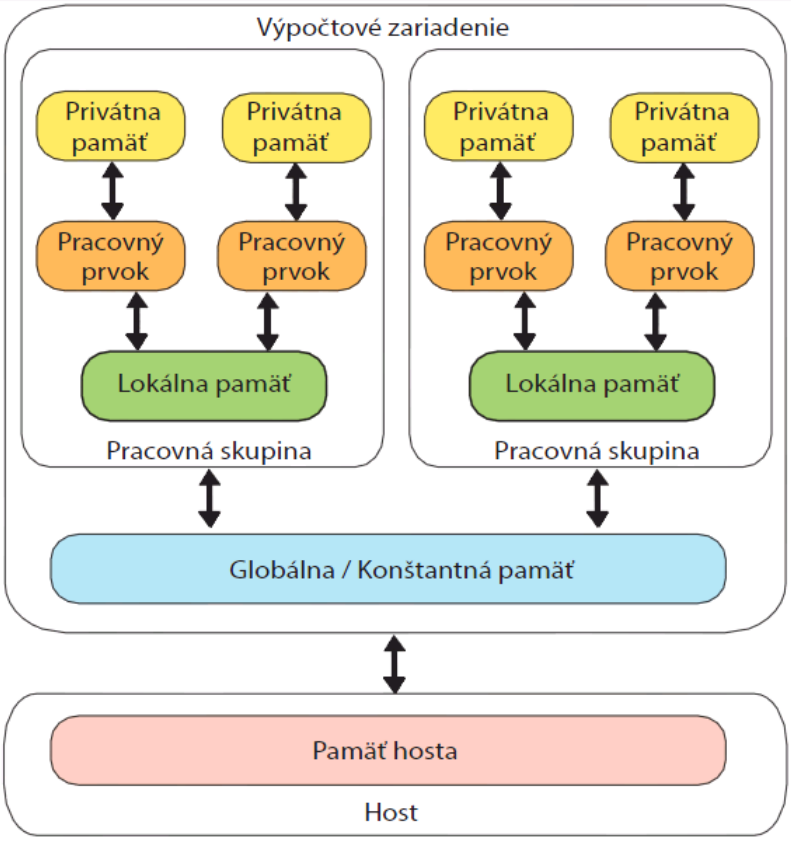
context – určuje konkrétny kontext správy, v ktorom je objekt image vytvorený.
flags – je bitové pole, ktoré sa používa pre určenie alokácie a využitie informácií pri vytváraní pamäť objektu. Príznaky môžu nadobúdať rôzne stavy, ak je však hodnota rovná 0, používa sa štandardne `CL_MEM_READ_WRITE` pre všetky typy objektu image okrem `CL_MEM_OBJECT_IMAGE1D_BUFFER`.
image_format – je ukazovateľ na štruktúru, ktorá popisuje vlastnosti formátu image objektu, ktorý má byť alokovaný. Formát je založený na koncepte kanálov, kedy sú definované počty prvkov, ktoré tvoria prvok image objektu (môžu byť jeden až štyri, založené na RGBA) a veľkosť týchto prvkov (1 až 4 byty) vo formáte ako integer alebo float.
image_desc – je ukazovateľ na štruktúru, ktorá popisuje typ a rozmery image objektu, ktorý má byť alokovaný.
host_ptr – je ukazovateľ na dáta image objektu, ktoré môžu byť alokované aplikáciou. Pre rôzne veľkosti a rozmery (dimenzie) obrazov sú definované príslušné podmienky pre požadovanú veľkosť bufferu, na ktorý ukazuje ukazovateľ hostiteľa.
errcode_ret – vracia chybový kód označujúci chybu pri volaní funkcie. Ak je nastavený na hodnotu „NULL“ nevracia žiadnu chybu kódu.
`clEnqueueCopyImageToBuffer` je funkcia pre kopírovanie z image objektu do bufferu a `clEnqueueCopyBufferToImage` naopak z bufferu do objektu image [24].

- Pipe

Je typ pamäťového objektu, tvorený usporiadaným sledom dátových položiek, ktoré sú uchovávané ako FIFO (First In First Out). Pipe (rúra) má dva koncové body. V koncovom bode zápisu sa dáta vkladajú a v koncovom bode pre čítanie sú dáta odstránené. V každom okamihu môže zapisovať do tohto objektu vždy len jeden kernel a rovnako aj čítať z tohto objektu môže vždy len jeden kernel. Pristupovať k rúre sa dá vždy len z funkcií, ktoré do potrubia zapisujú alebo z neho čítajú. Tento pamäťový objekt nie je prístupný hostiteľovi, nesie informácie ako veľkosť paketu v bytoch, maximálna kapacita paketov, informácie o počte paketov v danom čase v rúre, dátové pakety. Pamäťový objekt rúra je vytvorený pomocou funkcie:

```
cl_mem clCreatePipe(              cl_context context,
                                  cl_mem_flags flags,
                                cl_uint pipe_packet_size,
                               cl_uint pipe_max_packets,
                          const cl_pipe_properties *properties,
                                  cl_int *errcode_ret)
```

context – určuje konkrétny kontext správy, v ktorom je rúra vytvorená.
flags – je bitové pole, ktoré sa používa pre určenie alokácie, definovaní dát a pre určenie ďalších možností pri vytváraní rúry. Príznaky môžu nadobúdať rôzne stavy, ak je však hodnota rovná 0, používa sa štandardne `CL_MEM_READ_WRITE`.
pipe_packet_size – veľkosť paketu rúry udávaná v bytoch.
pipe_max_packets – určuje kapacitu rúry zadaním maximálneho počtu paketov, ktoré rúra môže držať.
properties – určuje zoznam vlastností pre rúru a ich zodpovedajúce hodnoty. Každému názvu vlastnosti prislúcha zodpovedajúca požadovaná hodnota. Zoznam je zakončený nulou. Od OpenCL 2.0, musí mať parameter properties hodnotu „NULL“.
errcode_ret – vracia chybový kód označujúci chybu pri volaní funkcie. Ak je nastavený na hodnotu „NULL“ nevracia žiadnu chybu kódu [24].



Obr. 2.2: Organizácia pamäte v OpenCL [24]

OpenCL od verzie 2.0 podporuje SVM (Shared Virtual Memory) teda zdieľanú virtuálnu pamäť, ktorá strane hostiteľa a výpočtovým kernelom umožňuje priamo zdieľať ukazovatele na dátové štruktúry či potrebu triedenia dát medzi hostiteľom a zariadením. OpenCL je možné využívať v operačných systémoch Windows i Linux, dokáže pracovať na rôznych zariadeniach, okrem GPU a CPU napríklad aj DSP či FPGA, no je náročný na počet krokov počas programovania. V roku 2013 vydala skupina Khronos OpenCL verziu 2.0 , s množstvom vylepšení, napríklad kód GPU môže podľa potreby spúšťať ďalší kód na GPU (dynamická paralelizácia), spomínaná zdieľaná virtuálna pamäť medzi hostiteľom a kernelmi zariadenia ai. [24].

2.3 DirectCompute

Je API pre univerzálne výpočty na GPU, pre Windows Vista či Windows 7/8. Predstavuje vlastne výpočtový shader, teda program ktorý je určený ku riadeniu jednotlivých častí GPU. Výpočtový shader poskytuje zdieľanie pamäte a synchronizáciu vlákien pre efektívnejšie paralelné programovacie metódy. DirectCompute využíva

jazyk HLSL (High Level Shader Language) a je súčasťou kolekcie DirectX10 a DirectX11. Rovnako ako OpenCL či CUDA sa skladá z vlákien, ktoré bežia súčasne a sú rozdelené do určitých skupín. Nepodporuje dynamickú paralelizáciu, má limitovanú zdieľanú pamäť, nepodporuje počítanie s dvojitou presnoťou, výhodou je, že ovládače DirectCompute sú robustnejšie a obsahujú menej chýb [25].

Veľkosť skupiny vlákien je v čase kompilácie konštantná, zatiaľ čo v OpenCL i CUDE možno tento parameter meniť až do odoslania a možno ho meniť i pre každé ďalšie vyvolanie inej skupiny. Za nevýhodu možno pokladať aj nie príliš dostupnú dokumentáciu o jednotlivých inštrukciách, funkciách a všeobecnom popise daného API. S DirectCompute sa môžeme stretnúť najmä v spojení s Direct3D.

2.4 Stream

ATI Stream technológia je sada pokročilých hardvérových a softvérových technológií, ktoré umožňujú AMD GPU pracovať súčasne s CPU, pre správnu optimalizáciu a zrýchlenie vyžadovaných úloh na týchto platformách. V súčasnosti firma AMD vývíja tzv. AMD APP (Accelerated Parallel Processing) technológiu pre rovnaký účel. AMD APP SDK (Software Development Kit) je kompletná vývojová platforma, ktorá obsahuje vzory, dokumentáciu a iný materiál pre rýchle výpočty pomocou OpenCL, Bolt, C++ AMP alebo Aparapi pre Java applikácie. Pokrok možno zaznamenať väčším výkonom počas runtime, teda počas doby kedy sa program vykonáva a pridaním matematických knižníc pre OpenCL. Najnovšia verzia AMD APP SDK v2.9 obsahuje vzory pre OpenCL, zrýchlené knižnice ako Open Source C++ šablónu knižnice nazývanú „Bolt“ a OpenCV (Open Computer Vision) knižnicu pre OpenCL. V súčasnosti sa však firma AMD zaoberá najmä OpenCL štandardom [26].

3 GPGPU A ALTERNATÍVY PRE PRIEMYSELNÉ SYSTÉMY

Ako bolo už viac krát spomínané, myšlienkou heterogénnych systémov je spustiť každú požadovanú úlohu na výpočtovej jednotke zariadenia s cieľom dosiahnuť optimálny výkon. Takéto výpočtové jednotky môžu byť napríklad DSP (Digital Signal Processor), GPU či FPGA (Field Programmable Gate Array). Dôležitým faktorom je zvolenie správnej jednotky, vhodnej pre dané účely a zabezpečenie, že využitie daného heterogénneho systému sa vyplatí aj z pohľadu ceny a obtiažnosti.

3.1 CPU

Táto skratka je už vo všeobecnosti známa, no pre krátku špecifikáciu – je to hardvérová jednotka, ktorej úlohou je spracovávať a vykonávať inštrukcie a výpočty. V súčasnosti je najčastejšie realizovaná formou mikroprocesorov. Od počiatku vývoja je funkcia procesoru takmer rovnaká, inovuje sa najmä jeho dizajn a rôzna forma paralelizmu (vektorové jednotky, multivláknové technológie) za zámerom robustnosti, lepšieho výkonu, rýchlejšieho spracovania kódu a pod. Medzi dôležité vlastnosti a charakterizácie CPU možno zaradiť rôzny stupeň produktivity a nastavenia vplyvom pokročilých programovacích jazykov, spôsobilosť pre beh operačných systémov, viacúrovňové vyrovnávacie pamäte pre menšie oneskorenie ai. Vďaka vysokému clock-rate „taktovacej frekvencii“ (vyjadruje frekvenciu, na ktorej CPU pracuje) je najlepšou voľbou pre riešenie skalárnych problémov, zároveň je však nutné počítať s vyššou spotrebou. Pri operácii s rádovou desatinou čiarkou sa používa výraz FLOPS (Floating-Point Operations Per Second) kedy procesor dosahuje teoretických hodnôt 200 GFLOPS s vektorovými jednotkami, energetická účinnosť potom činí 2 GFLOPS/watt a 50 GFLOPS bez vektorových jednotiek, energetická účinnosť potom činí 0,5 GFLOPS/watt bez týchto jednotiek (vo vektorovom procesore sa inštrukcia vykonáva súčasne na viacerých dátových zložkách, pre skalárny je to naopak) [27]. Pri použití moderných architektúr však veľkosť GFLOPS významne vzrástla vďaka kombinácii s grafickými jadrami (ako napr. u AMD A10-5700 dosahuje výkon 693 GFLOPS) [28]. V tab. 3.1 sú uvedené hodnoty frekvencie, na ktorých porovnávané CPU pracujú; TDP (Thermal Design Power) – vyjadruje tepelné straty procesora; počet vlákien na jadro a cena uvedená pri ich predstavení.

CPU	Taktovacia frekvencia	TDP	Jadro/Vlákno	cena
Intel Core i5 4670K	3,4 GHz	84 W	4/4	242 $
AMD A10-5700	3,4 GHz	65 W	4/4	122 $

Tab. 3.1: Porovnanie medzi architektúrou Trinity firmy AMD a architektúrou Haswell firmy Intel [29], [30]

3.2 GPU

Procesory súčasných GPU sú programovateľné paralelné procesory. Okrem riešenia problémov v oblasti grafiky, vedea či obrazov sa využívajú i pre úlohy spojené so spracovaním väčšieho množstva dát v paralélnej podobe. Bližšia definícia a vlastnoti GPU sú uvedené v prvej kapitole 1. Pri porovnaní s CPU sú efektívnejšie, medzi dôležité vlastnosti a charakterizácie GPU patrí najmä možnosť dostupných a kvalitných API, štruktúra tvorená veľkým množstvom jadier, ktoré uskutočňujú tisícky inštrukcií za cyklus, dostatočná šírka pásma pamätí postačujúca pre každé jadro, nenáročná veľkosť prevedenia – šetrí sa priestor i spotreba. Pri operácii s desatinou čiarkou dosahuje teoretických hodnôt 3500 GFLOPS s vektorovými jednotkami, energetická účinnosť činí teoreticky 16 GFLOPS / watt [27]. V tab. 3.2 sú uvedené frekvencie, na ktorých porovnávané GPU pracujú, TDP, maximálny teoretický výkon a cena uvedená prí ich predstavení. Je vhodné uviesť, že NVIDIA GeForce GTX 770 bola vydaná v máji 2013 a AMD Radeon HD 7970 v júni 2012.

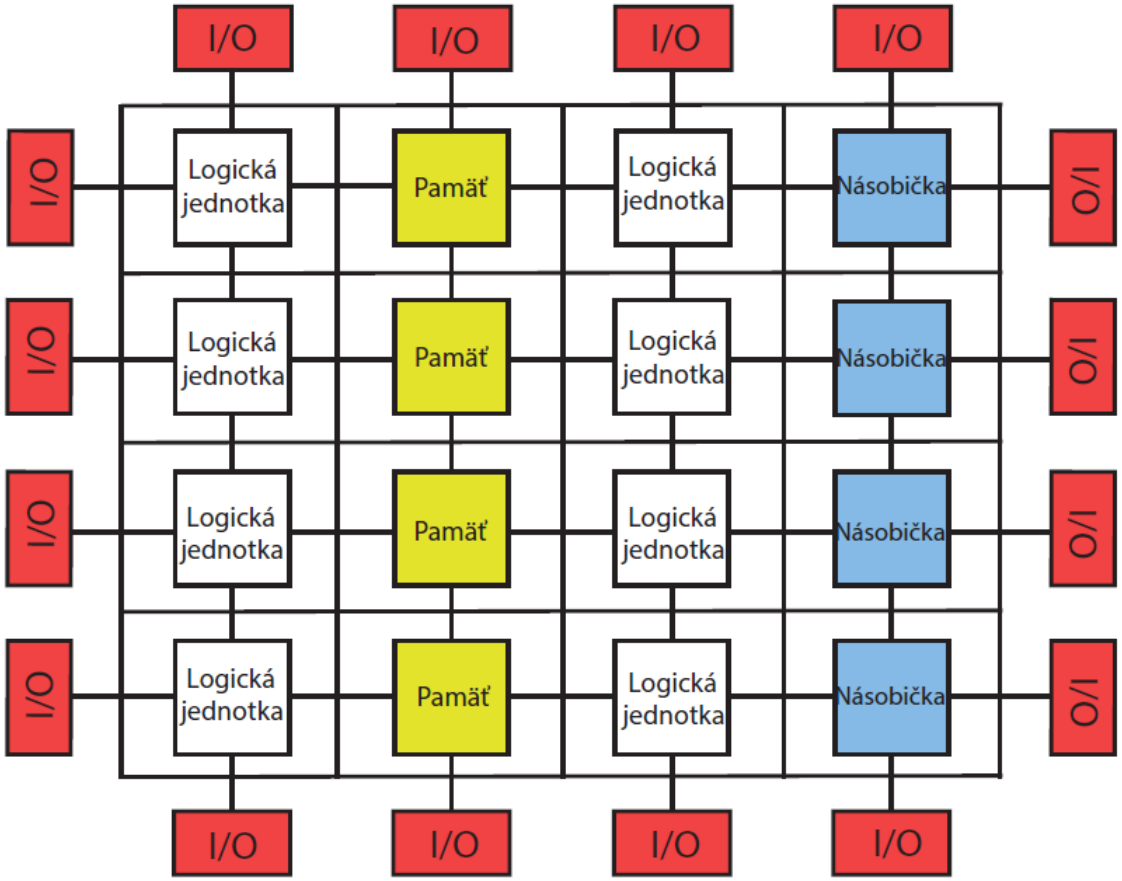
CPU	Taktovacia frekvencia	TDP	Max.výkon	cena
GeForce GTX 770	1,046 GHz	230 W	3213 GFLOPS	399 $
Radeon HD 7970	1 GHz	300 W	4096 GFLOPS	499 $

Tab. 3.2: Porovnanie tzv. diskrétnych GPU od firmy NVIDIA a AMD [31], [32]

3.3 FPGA

FPGA (Field-Programmable Gate Array) – „programovateľné hradlové polia“ boli vynájdené už v roku 1894 zakľadateľom firmy Xilinx. Prvé FPGA obsahovali len zopár tisícok brán a mali množstvo nedostatkov, boli pomalšie, náročnejšie na spotrebu a limitované z pohľadu funkcií. Dnešné FPGA sú využiteľné pre množstvo rôznych aplikácií, vývojom vzniklo množstvo sérií a rodín obsahujúc rôznu logiku pre rýchlejšie spracovanie, disponujú veľkou kapacitou pre paralelizáciu a reťazenie procesov. Často sa využívajú ako periférne zariadenia ku CPU pre vykonávanie procesov, ktoré sú pre samotné CPU obtiažnejšie realizovať.

Architektúra FPGA predstavuje polovodičové zariadenie zložené z množstva logických blokov, ktorých prepojenie sa dá nakonfigurovať. Medzi každým logickým blokom sa nachádzajú smerovacie kanáliky, ktoré sú programovateľné a umožňujú komunikáciu medzi blokmi. Do FPGA boli implementované okruhy pre rôzne účely, napr. násobička, DSP okruhy a bloky RAM [33].



Obr. 3.1: Štruktúra FPGA, programovateľné bloky pospájané cestičkami, ktoré umožňujú komunikáciu. I/O (input/output) bloky pre spojenie čipu s vonkajším prostredím [34]

Používateľ má možnosť definovať správanie sa FPGA pomocou programovacieho jazyka HDL (Hardware Description Language) pre popis funkcie číslicových prístrojov, pričom v súčasnosti sa využíva jeho vylepšená verzia VHDL (VHSIC Hardware Description Language). Nevýhodou je menší stupeň produktivity, jazyk je pomalší a komplexnejší, pretože programátor je zopodvedný okrem softvérovej časti aj za hardvérový návrh. Existuje celý rad programovacích technológií, ktoré majú rôzny vplyv na programovateľnú logickú architektúru. Medzi prístupy, ktoré boli využívané v minulosti patrí napr. EPROM, EEPROM, flash, statická pamäť a anti-fuses,

z ktorých sa v moderných FPGA (založených najmä na statickej pamäti) využívajú posledné tri menované.

Statické pamäťové bunky sú základom pre SRAM technológiu, ktorá sa stala dominantnou technológiou prístupu u FPGA vďaka opakovateľnej možnosti preprogramovania a použitia CMOS technológie. Moderné FPGA na báze SRAM majú najvyššiu hustotu ale spotrebujú veľké množstvo energie a vyžadujú externú pamäť pre ukladanie konfigurácie bitového toku. Externá pamäť nie je potrebná v prípade, že FPGA obsahuje vnútorný flash modul. SRAM sa využíva vo FPGA najväčších súčasných výrobcov ako Xilinx, Altera, Lattice. FPGA na báze flash a antifuse spotrebujú oveľa menej energie ako SRAM. Antifuse však možno naprogramovať iba raz.

FPGA dosahujú na úkor výkonu menšiu spotrebu energie, tá záleží od konkrétneho návrhu ale väčšinou činí 20–40 Watt, pri ktorej efektivita dosahuje 25 GFLOPS/ Watt [27]. FPGA nepredstavuje teda pevný výpočtový/programovací model ako GPU ale ponúka možnosť logiky navrhnutej pre potreby aplikácie, zároveň môže byť využitý pre širokú škálu zariadení vďaka prispôsobiteľnému vstupu/výstupu. V súčasnosti majú najväčšie povedomie na trhu najmä firmy Xilinx a Altera. Pri porovnaní s GPU, kedy najvýkonnejšie typy dosahujú vrcholový výkon okolo 4 TFLOPS, je hodnota takéhoto výkonu u FPGA okolo 1 TFLOPS. Ako príklad možno uviesť populárnu Spartan-6 FPGA, ktorá pracuje na maximálnej frekvencii 500 MHz, pričom konkrétna frekvencia závisí na vykonávanom kóde, jej cena je 399 $ [35].

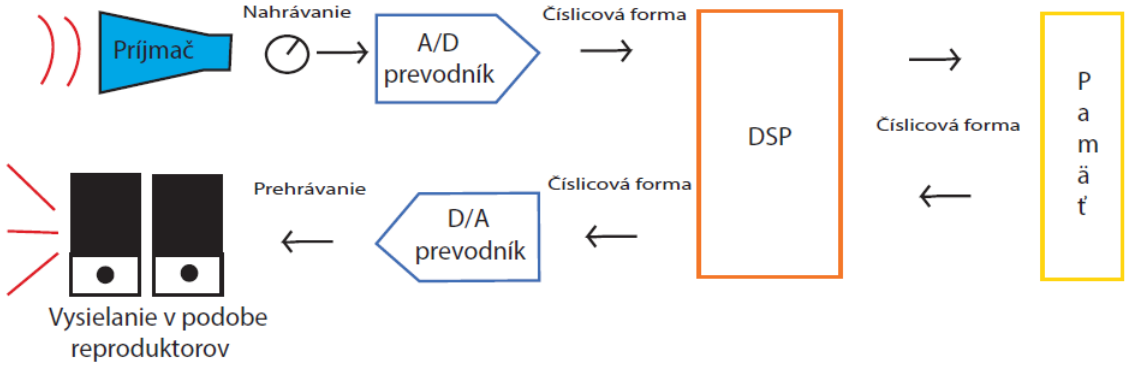
3.4 DSP

DSP (Digital Signal Processor) patrí do skupiny programovateľných súčiastok. Predstavuje špeciálnu formu mikroprocesora, ktorý bol navrhnutý pre potreby časovo závislého a rýchleho spracovania dát. Prvé typy DSP sa používali už na prelome sedemdesiatych–osemdesiatych rokov minulého storočia ako súčasť minipočítačov. V prípade že nebolo možné využívať minipočítač, alternatívou sa stávali analógové počítače, ktoré boli však rovnako nahradené práve DSP procesormi vďaka pokroku vo vývoji polovodičových čipov. Z pohľadu spracovania číslicových signálov si našli svoje uplatnenie napr. u aplikovaní teórie o vzorkovaní signálov, u rôznej transformácie signálov (diskrétna/rýchla Fourierova transformácia), pri spracovávaní digitálneho navzorkovaného signálu pomocou číslicových filtrov s konečnou impulznou odozvou (FIR) a s nekonečnou impulznou odozvou (IIR).

Ako príklad pri spracovaní signálu v reálnom čase možno uviesť využitie DSP pri spracovaní zvuku (napríklad pri implementácii ekvalizéru). V ňom sa digitalizovaný a navzorkovaný zvuk spracováva s určitým oneskorením a vďaka tomuto procesoru je zaručené že sa doba oneskorenia nezmení. Digitálny signálový procesor je aj dô-

ležitou súčasťou modemu. Zabezpečuje že signál, nepretržite príjmaný modemom z komunikačnej linky, bude spracovávaný bez prerušenia a straty vzorkov. Ak by to program uložený v DSP nedokázal, celý dátový blok by musel byť opakovane odoslaný a pri pokračovaní takéhoto problému by bola komunikácia nemožná [36] [37].

Obr. 3.2: Zjednodušená verzia použitia DSP pri spracovaní analógového signálu [36]



Obr. 3.3: Príklad konkrétneho využitia DSP (MP3 audio prehrávač) [37]

Pri spracovaní analógového signálu digitálnym signálovým procesorom je tento signál privádzaný na analógovo-digitálny prevodník (A/D), v ktorom sa vzorkuje a výstupom je sled číslicových vzorkov. Výsledné vzorky sú privádzané a spracovávané programom uloženom v DSP. Výsledkom po spracovaní sú väčšinou opäť číslicové vzorky, ktoré sú privádzané buď na digitálno-analógový (D/A) prevodník kde je signál späť prevedený na analógový alebo sú priamo poslané do ďalšieho digitálneho zariadenia (napr. prípad modemu). A/D a D/A prevodníky môžu byť priamo súčasťou DSP alebo sú realizované samostatnými obvodmi. Niektoré typy DSP tieto prevodníky neobsahujú pretože vstupný alebo výstupný signál môže byť spracovávaný iba v číslicovej podobe.

Filtre FIR a IIR možno implementovať pomocou základných aritmetických operácií sčítania a násobenia. Pri implementácii týchto filtrov je možné vykonávať operácie ako čítanie či zapisovanie hodnôt do polí pevnej dĺžky. Rovnako ako pre tieto filtre, sú niektoré DSP uspôsobené aj pre rôzne druhy fourierovej či kosínovej

transformácii aby sa dali naprogramovať čo najjednoduchším spôsobom. Prispôsobené bývajú časti DSP ako pracovné registre, inštrukčné sady a v neposlednej rade aritmeticko-logické jednotky (usporiadanie sčítačky a násobičky). Veľkou výhodou DSP je, že pri porovnaní s analógovým obvodom, ktorý je síce lacnejší z finančného pohľadu na súčiastky napr. pri funkcii filtra ale obtiažnosť takéhoto obvodu rastie exponenciálne pri zväčšujúcich sa požiadavkách filtru. DSP je jednoduchšie navrhnúť, upraviť a najmä je táto funkcia založená na softvére. To znamená, že úpravou softvéru možno dosiahnuť flexibilne nastaviteľné filtre pre väčšie požiadavky v prípade neskoršieho využitia bez ďalšieho nakupovania súčiastok ako by to bolo v prípade podobného analógového filtru [38].

MAC (Multiply And Accumulate) je inštrukcia, kedy je možné vykonávať niektoré operácie súbežne – vždy jednu operáciu násobenia a sčítania zlúčiť do jedinej inštrukcie, ktorá zároveň svoj výsledok ukladá do akumulátoru. Akumulátor je teda pracovný register, do ktorého sa akumulujú výsledky. Z pohľadu pamäte je potrebné aby DSP mal k dispozícii oddelenú a samostatne adresovateľnú pamäťovú oblasť, do ktorej sa vkladajú koeficienty filtru. Okrem tejto požiadavky by mal mať DSP dostatočne veľký počet pracovných registrov alebo podporovať súčasný prístup do viacerých oblastí pamäte [36].

Pre vykonávanie inštrukcií sa používajú dve metódy: **SIMD** (Single Instruction Multiple Data) je metóda, kedy každá inštrukcia vyjadruje rovnakú operáciu, ktorá sa uskutoční na viacerých dátovych vzorkoch. U metódy **VLIW** (Very Long Instruction Word) je inštrukčné slovo veľmi dlhé a obsahuje viac ako jednu operáciu, ktoré sa potom vykonávajú paralelne – výhodou je vykonanie odlišných operácií súčasne). V porovnaní s CPU pracujú na menšej frekvencii, majú menšiu veľkosť pamäte [39].

Ako je uvedené v [40], najlacnejšie DSP, ktoré majú zároveň najmenšiu spotrebu sú série TMS320C553x, podporujúce matematické algoritmy, pracujú pri frekvencii 50/100 MHz, (0,15 mV/Hz pri 1,05V) pri rôznom vybavení sa ich cena pohybuje od 1,95 $ po 4,70 $ pričom posledná cenová varianta obsahuje väčší počet periférií, má väčšiu pamäť a iné výhody.

4 PARALELIZÁCIA ALGORITMOV

Algoritmus je postup alebo predpis, podľa ktorého sa požadovaná úloha spĺňa. Charakter algoritmu možno rozdeliť podľa kritérií, napr. aké operácie máme k dispozícii či postup poradia v akom môžu byť algoritmy vykonávané:

- práve jeden (sekvenčné)
- niekoľko súčasne (paralelné)

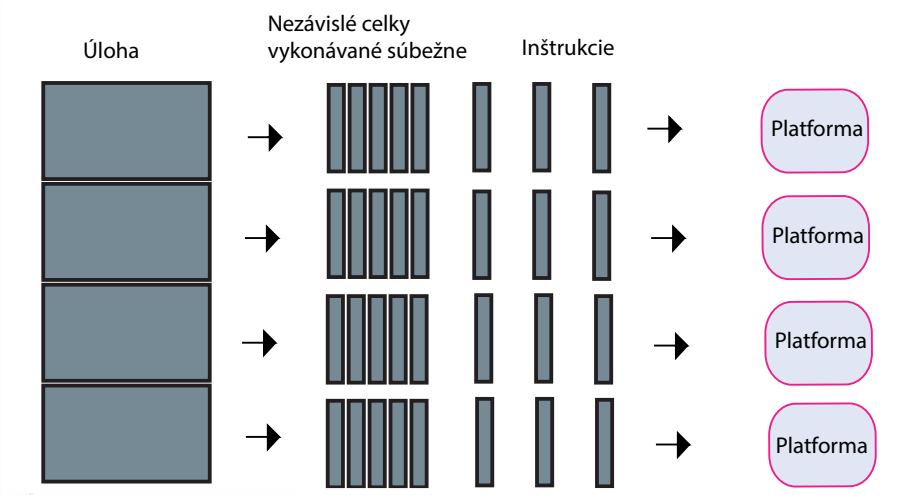
Výber algoritmu pre danú úlohu v súčasnosti záleží najmä na type procesora a organizácii pamäte. Logicky sa môže čas potrebný pre vykonanie požadovanej úlohy zmenšiť, vďaka súčasnému vykonávaniu väčšieho počtu operácií. Pri paralelnom počítaní je úloha rozložená do viacerých častí, ktoré môžu byť vykonávané súčasne. Čas potrebný pre vykonanie paralelného programu závisí od veľkosti vstupu, parametrov komunikácie počítačového systému a počtu procesorov, ďalej je potrebná analýza algoritmu podľa platformy, na ktorej bude spustený.

Ak máme napríklad paralelný systém, operácie v paralelnom algoritme sa môžu vykonávať súčasne odlišnými procesormi. Rovnako sa však môže využiť paralelný algoritmus aj v systéme obsahujúcom práve jeden procesor s viacerými jadrami. Preto je potrebné rozlíšiť paralelizáciu v algoritme a schopnosť paralelného spracovania operácií konkrétneho počítača. Ak máme dva procesory, nemusí to znamenať že program bude bežať dvakrát rýchlejšie. Pri viacerých zdrojoch réžie vznikajú nadbytočné výpočty, interakcia medzi procesormi – teda potreba komunikácie, idling – nečinnosť či obsadenie zdrojov spôsobujúce degradáciu výkonu [41].

Nečinnosť môže vzniknút v dôsledku nevyrovnanej záťaže, synchronizácie či výskytu sekvenčnej časti v probléme. Nevyrovnanosť záťaže vzniká ak záťaž nie je rovnomerne rozdelená medzi procesory, v dôsledku čoho nečinné procesory musia čakať na iné, ktoré ešte neukončili svoj výpočet. Nadbytočné výpočty môžu byť výpočty, ktoré nie je potrebné realizovať v sekvenčnej časti alebo v prípade minimalizácie komunikácie sa vykonávajú niektoré výpočty na viacerých procesoroch.

Ďalším dôležitým atribútom pri paralelizácii je cena. Cena vyjadruje súčin času behu paralélneho programu a počtu procesorov, odzrkadľuje súčet časov, ktoré strávil každý procesor pri riešení zadaného problému. Paralelný systém je potom optimálny ak je cena riešenia problému v paralelnej podobe asymptoticky rovná cene sériového riešenia problému. Asymptotická zložitosť sa používa ako nástroj pre porovnanie efektivity a rýchlosti vykonávania algoritmov. Zisťuje akým spôsobom sa bude meniť povaha algoritmov pri zmene vstupných dát [42].

Nie vždy je možné program efektívne paralelizovať, pred jeho implementáciou je potrebné ho dekomponovať – zabezpečiť aby sa predišlo vyššie spomenutým problémom ako je nevyvážená záťaž, náklady na pamäť ai.



Obr. 4.1: Príklad paralelného počítania, dekompozícia na celky až inštrukcie s využitím viacerých zdrojov

Z pohľadu algoritmov rozoznávame [43]:

- inherentný paralelizmus – problém možno riešit sekvenčným i paralelným algoritmom
- ohraničený paralélny algoritmus – vyjadrený paralelným časom výpočtu vo vzťahu na zdroje počítačovej architektúry
- neohraničený paralélny algoritmus – vyjadrený paralelným časom výpočtu bez ohľadu na zdroje počítačovej architektúry

Masívny paralelizmus

Masívny paralelizmus vyjadruje rozsiahlu množinu funkčne rovnakých problémov, ktoré paralelne spracovávajú veľké množstvo úloh v rovnakom čase. Využíva sa pre vysokovýkonné výpočty, je často označovaný ako údajový paralelizmus. Výpočet býva vykonávaný nezávislým počtom úloh ktoré spracovávajú nezávislé množstvo údajov. Býva využívaný v architektúrach SIMD a MIMD. Okrem masívneho paralelizmu poznáme jednoduchý, expanzívny a prúdový paralelizmus [43].

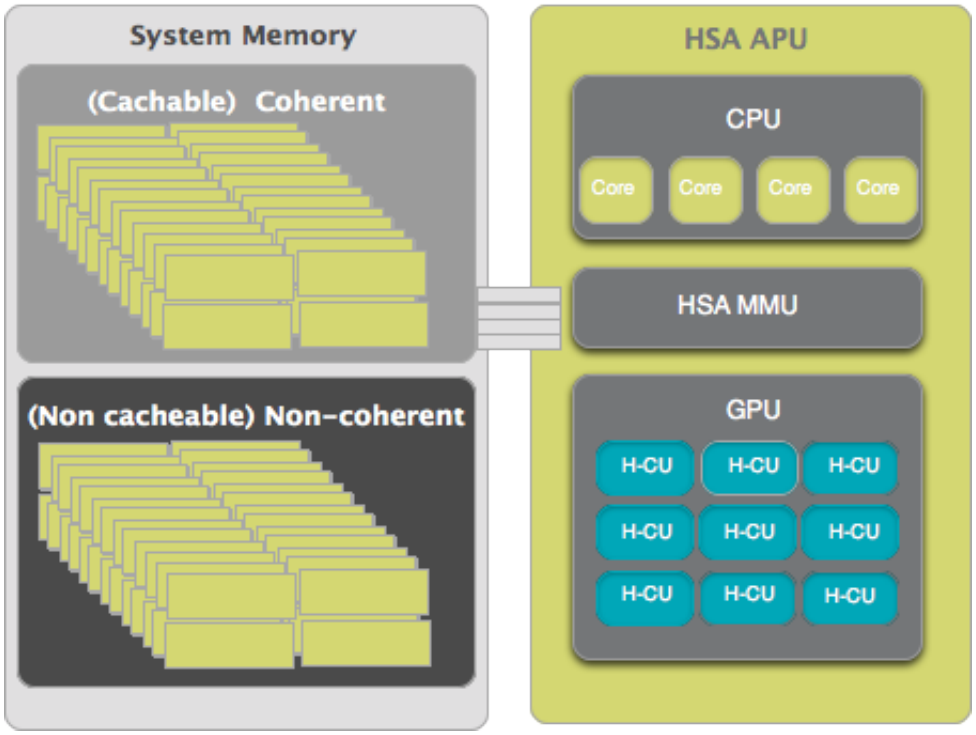
5 HETEROGÉNNA ARCHITEKTÚRA A KOMUNIKÁCIA S HARDWAROM

5.1 Heterogeneous Systems Architecture

HSA (Heterogeneous Systems Architecture) je hardvérová architektúra, ktorá spája jednotlivé heterogénne jednotky spracovania do koherentného prostredia. Koherentným prostredím sa myslí prostredie, v ktorom je zabezpečené, že viacero procesorov má rovnaký pohľad na pamäť, aj napriek tomu že hodnoty môžu byť menené nezávisle ktorýmkoľvek zo skupiny procesorov. Nepochybne možno hovoriť o výhodách najmä z pohľadu kratšieho času vykonania inštrukcií, menšej energetickej náročnosti a výrazne ľahšieho programovania. Programy vykonávané pomocou CPU tak môžu byť odovzdané na spracovanie pomocou GPU či DSP rovnako jednoducho ako iný program na rovnakom CPU. Princíp spočíva v tom, že v pamäti, zdieľanej niekoľkými procesormi, sa vytvorí ukazovateľ na dáta a aktualizuje sa niekoľko front. Dáta, ktoré majú byť spracované systémom bez HSA, musia príslušné CPU programy zabaliť a následne vykonať požiadavku prostredníctvom I/O pre odoslanie dát pomocou ovládačov zariadenia, ktoré spolupracujú s GPU alebo DSP, čo je značne komplikovanejšia záležitosť. Do vývoja virtuálnych pamätí boli pred štyridsiatimi rokmi investované veľké peniaze, no v súčasnosti sa táto investícia ukazuje ako výhodná a virtuálne pamäte sa stávajú čoraz dostupnejšími.

Výdrž batérie je v elektronike všeobecne dôležité kritérium a všetky aspekty dizajnu soketov v tejto architektúre boli uspôsobené pre čo najlepšie riešenie aj u najnáročnejších aplikácií. Pre vykonanie zadanej úlohy, umožňuje HSA jednotlivým softvérovým balíkom použiť taký procesor zo soketu, ktorý je najvýhodnejší z pohľadu úspory energie. Napríklad ak sa jedná o obtiažnu paralelnú úlohu, softvér využije GPU, ktorý spotrebuje menej energie na výpočet ako univerzálne navrhnuté jadrá CPU. Rovnako prichádzajúce toky dát môžu byť presmerované na DSP za účelom ich analýzy, pre ktorú bol tento procesor optimalizovaný. A v konečnom dôsledku práve zdieľanie dátových štruktúr v pamäti systému všetkými jednotkami spracovania eliminuje potrebu kopírovania z jednotlivých sekcií pamäte riadenými CPU do inej sekcie pamäte riadenými GPU. Toto riešenie predstavuje veľkú úsporu energie i času [44].

V tradičných GPU zariadeniach, i v prípade že GPU a CPU využívajú rovnakú oblasť pamäte, GPU využíva samostatný adresový priestor CPU a grafický ovládač musí v požadovaných intervaloch vyprázdniť cache pamäť v prípade potreby zdieľania výsledkov pre GPU a CPU. V porovnaní s týmto systémom, HSA obsahuje plne koherentný model zdieľanej pamäte so zjednoteným adresovaním, obr. 5.1.



Obr. 5.1: HSA APU platforma zložená z viacjadrového CPU, GPU s viacnásobnými výpočtovými jednotkami H–CUs a pamäťovou riadiacou jednotkou HMMU. Tieto komponenty komunikujú s koherentnou i nekoherentnou pamäťou systému [45]

Okrem základnej myšlienky spojenia CPU a GPU sa HSA aplikuje aj u FPGA či DSP. V HSA systéme je GPU považovaný ako ďalší procesor, kódy špecifické pre GPU sú vyvolávané prostredníctvom sady štandardných knižníc rozhrania, a sú odosielané cez fronty úloh rovnako ako v prípade CPU. HSA obsahuje i sadu virtuálnych inštrukcií architektúry, známu pod názvom HSAIL (HSA Intermediate Language), ktorý zjednoduchšuje softvérovú distribúciu. Zároveň bolo prehlásené, že kódy vytvorené na prvotných HSAIL štandardoch bude možné spustiť i na novších štandardoch, vzniknutých postupným vývojom.

U HSA architektúry sa pracuje s dvomi základnými výpočtovými jednotkami:

- LCU (Latency Compute Unit) – zovšeobecnením CPU, podporuje sadu základných inštrukcií CPU, sadu inštrukcií jazyka HSAIL.
- TCU (Throughput Compute Unit) – zovšeobecnením GPU, podporuje len sadu inštrukcií jazyka HSAIL a je veľmi efektívna pre paralelizmus [46].

HSA zariadenia vzájomne komunikujú používaním front. CPU a GPU si môžu jednotlivé úlohy pridávať do front samostatne pre seba alebo vzájomne. V závislosti od zložitosti systému HSA, môžu byť fronty riadené kombináciou softvéru a hardvéru.

Organizácia pamäte

Všetky prístupné oblasti pamäte v HSA sú zmapované do jednotného virtuálneho adresového priestoru – SVM (Shared Virtual Memory). Táto virtuálna pamäť má následovné členenie [46]:

- Global – pamäť dostupná pre všetky pracovné prvky a pracovné skupiny vo všetkých LCU a TCU jednotkách. Zahŕňa najväčšiu výhodu zjednoteného pamäťového modelu HSA – zdieľanie dát medzi LCU a TCU jednotkami.
- Group – pamäť dostupná pre všetky pracovné prvky v pracovnej skupine.
- Private – pamäť dostupná pre jednotlivý pracovný prvok.
- Kernang – pamäť určená len pre čítanie, využíva sa k odovzdaniu argumentov do výpočtového kernelu.
- Readonly – globálna pamäť určená len pre čítanie.
- Spill – pamäť pre načítanie a alokáciu registrov.
- Arg – pamäť určená na zadanie argumentov do a z funkcií.

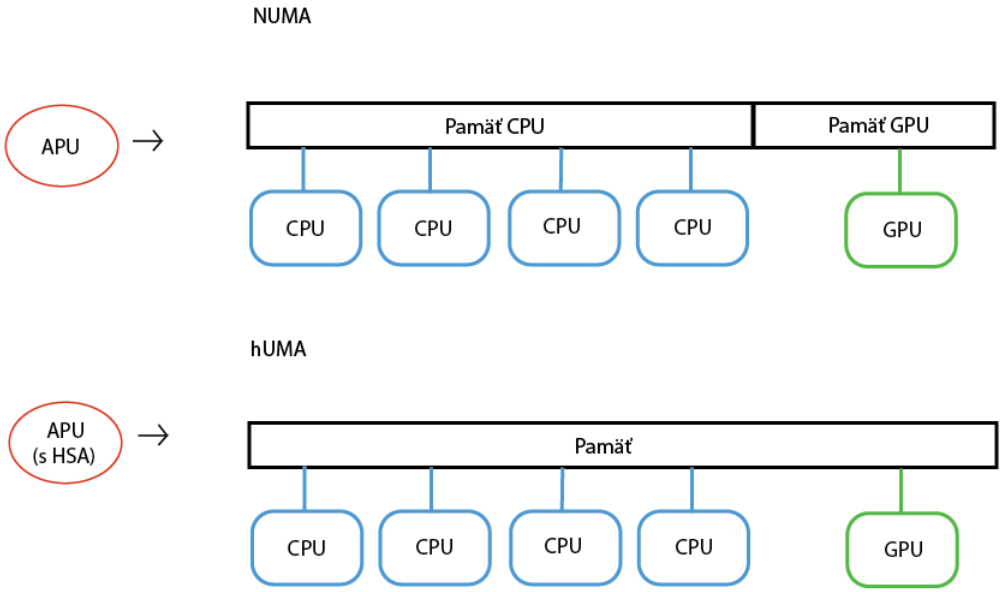
5.2 Pamäťové modely NUMA a hUMA

NUMA (Non-Uniform Memory Access) – predstavuje pamäťový model, kedy sú zoskupené procesory do multiprocesorového systému, v ktorom má každý procesor vlastnú pamäť a ktorú môže lokálne zdielať. Z názvu vyplýva, že sa jedná o nejednotný prístup do pamäte – procesor má najrýchlejší prístup do svojej pamäte a najdlhší čas je pre prístup do pamäte iného procesora, prípadne zdieľanej pamäte.

NUMA sa využíva u SMP (Symmetric Multiprocessing System), takéto zoskupenie pracuje pod rovnakým operačným systémom a je pevne spojené spoločnou zbernicou alebo vzájomne prepojenými cestami. Limitáciou SMP je, že čím viac procesorov sa pripojí, tým preťaženejšie sú zdieľané zbernice alebo cesty a výkon sa zmenšuje. NUMA v tomto prípade ešte pridáva strednú úroveň pamäte zdieľanej medzi niekoľkými mikroprocesormi aby všetky prístupy k dátam nemuseli viesť na hlavnú zbernicu.

Pri vyhľadávaní dát procesorom v pamäti, najskôr vyhľadáva v L1 cache na vlastnom mikroprocesore, pokračuje na o niečo väčšej L2 cache, ktorá je umiestnená zvlášť na čipe, je väčšia ale pomalšia v porovnaní s L1 cache. Tretiu úroveň tvorí L3 cache, ktorá býva implementovaná rovnako zvlášť na čipe od CPU [47].

Na obr. 5.2 je zobrazený rozdiel medzi pamäťovým modelom NUMA a hUMA. CPU a GPU sú síce viazané k systémovej pamäti, avšak v oboch prípadoch majú svoj vlastný pamäťový priestor.

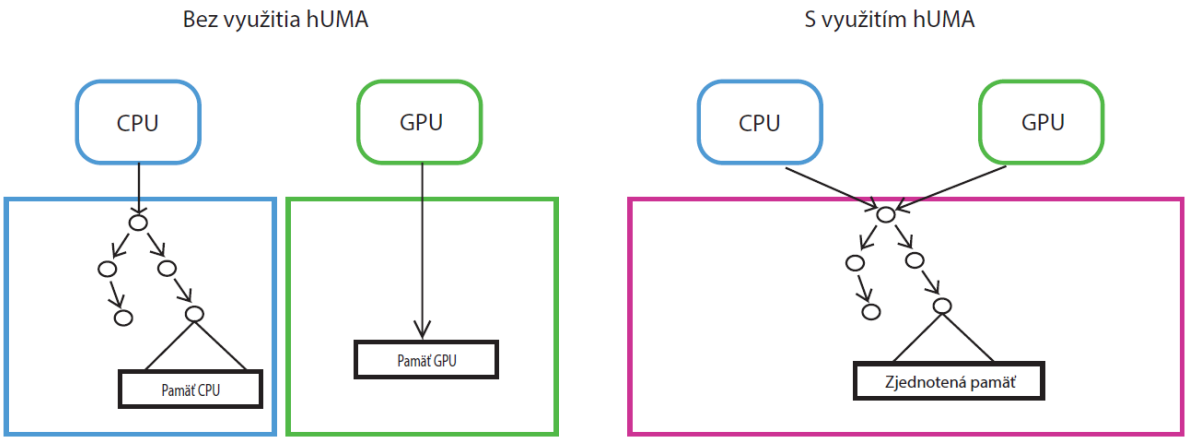


Obr. 5.2: Porovnanie jednotlivých pamäťových modelov [48]

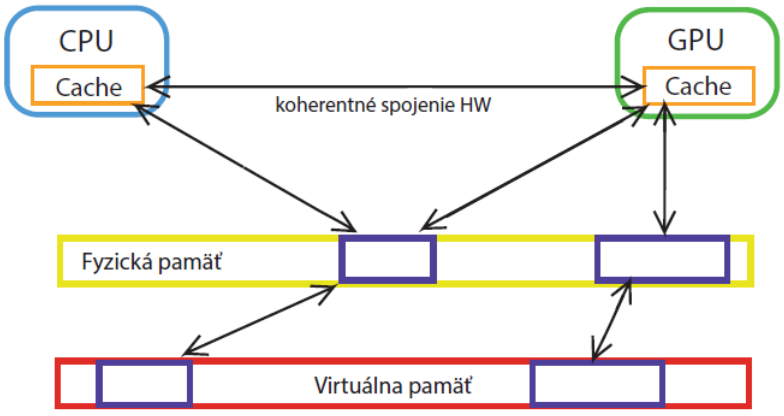
hUMA (heterogeneous Uniform Memory Access) je model pamäte, ktorý zjednocuje pamäť pre GPU a CPU, čím je zabezpečené, že akékoľvek zmeny, ktoré boli vykonané jedným zariadením budú dostupné a viditeľné aj pre ostatné zariadenia, teda CPU alebo GPU. Obe zariadenia majú prístup k celej pamäti a podľa potreby môžu pamäť dynamicky alokovať, zobrazené na obr. 5.3.

V architektúre bez využitia hUMA musia jednotlivé jadrá procesoru vykonať niekoľko skokov ak sa chcú dostať do pamäte používanej grafickým hardvérom a naopak. Pri zdielaní dát sú kopírované tam a späť medzi CPU a GPU. Všetky úkony potrebné k tejto realizácií sú zodpovedné za určitú stratu výkonu. hUMA odstraňuje potrebu skokov, jadrá procesoru spoločne s integrovanou grafikou majú zdielaný adresný priestor a obe zariadenia zdielajú virtuálnu i fyzickú pamäť. Tento model zároveň zabezpečuje bezstratovosť cyklov, potrebných ku synchronizácii GPU a CPU (vyskytujúce sa v NUMA) vďaka obojsmernej koherentnej pamäti [49].

Porovnanie pri výmene zdieľaných dát u architektúry využivajúcej hUMA model je zobrazené na obr. 5.4.



Obr. 5.3: hUMA model odstraňuje potrebu kopírovania dát a výsledkov z pamäte do pamäte, GPU má možnosť sledovať vložené odkazy [48]



Obr. 5.4: Výhody dosiahnuté hUMA modelom, napr. stránkovateľná pamäť – GPU má bezproblémový prístup k adresám virtuálnej pamäte, ktoré (zatiaľ) nie sú obsiahnuté vo fyzickej pamäti [48]

5.3 Komunikácia s hardwarom a pamäťový prístup

Komunikácia medzi dvomi zariadeniami je vlastne inak povedané komunikácia medzi pamäťovými prvkami na strane jedného zariadenia a pamäťovými prvkami na strane druhého zariadenia. Cieľom periférnej operácie je preniesť informáciu z počítača (resp. jeho operačnej pamäte) do periférneho zariadenia. Na tejto operácii sa podieľajú počítač, radič periférneho zariadenia, periférne zariadenie (PZ). Princíp komunikácie medzi procesorom a radičom je založený na komunikácii s registrami radiča a býva realizovaný pred zahájením periférnej operácie a po jej skončení. Pred samotnou realizáciou periférnej operácie musí počítač zistiť v akom stave sa dané periférne zariadenie nachádza pomocou procesov, ktoré tento stav hlásia. Ak je hlásený výskyt chyby (akejkoľvek, napríklad aj porucha hardware PZ), potom bude operácia považovaná za neplatnú a musí sa zopakovať.

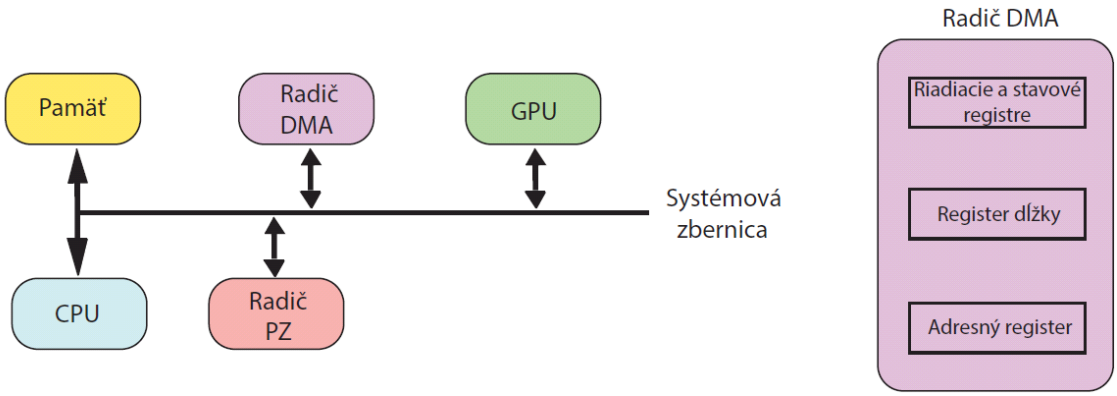
Obr. 5.5: Periférna operácia

Radič obsahuje pamäťové prvky register a pamäť. V registri sa nachádzajú informácie o parametroch periférnej operácie. Do vyrovnávacej pamäte sa nahrajú dáta rýchlosťou systémovej zbernice, až budú všetky dáta nahraté, radič na to upozorní periférne zariadenie alebo procesor (podľa smeru komunikácie) a môže pokračovať častou vkladania/zapisovania dát na periférne zariadenie. Z názvu vyrovnávacej pamäte vyplýva, že vyrovnáva rozdiel v rýchlosti zariadení komunikujúcich medzi sebou [50].

Pre uľahčenie práce procesora a zariadení požadujúcich veľmi rýchlu obsluhu sa využíva špeciálny hardvérový mechanizmus nazývaný Direct Memory Access (DMA), v preklade „priamy prístup do pamäte“. Využitím DMA odpadá práca a réžia procesoru, kedy by sa informácia musela prenášať inštrukciou (typu IN) z radiča PZ do registru procesoru a ďalšou inštrukciou z registru procesora do pamäte, čo je časovo náročné. DMA prenos je riadený DMA radičom.

Pre vykonanie prenosu, medzi I/O (vstupno/výstupným) zariadením a systémovou DRAM pamäťou, vyšle PZ žiadosť o DMA prenos DMA radiču a DMA radič vyšle následne žiadosť procesoru o súhlas s riadením systémovej zbernice DMA radičom. Pri súhlase procesor vyšle potvrdenie a odpojí sa od systémovej zbernice

(uvedie všetky výstupy do stavu vysokej impedancie) aby sa zamedzilo riadeniu systémovej zbernice dvomi zariadeniami. Pri prenose bude na adresovej časti zbernice vždy prítomná počiatočná adresa operačnej pamäte, tú procesor pred odpojením nastaví do radiča DMA spolu s definovaním režimu činnosti (napr. smer toku dát) a dĺžkou bloku. O vkladanie adresy do systémovej zbernice sa potom stará DMA radič spolu s vysielaním riadiacich signálov. Zatiaľ čo je kopírovanie dát vykonávané pomocou DMA, CPU môže vykonávať programy, ktoré nie sú závislé od I/O dát. Kopírovanie dát je v porovnaní s mnohými procesormi rýchlejšie, aj vďaka samotnému hardwaru, ktorý je na kopírovanie dát prispôsobený. Po ukončení prenosu procesor prevezme späť riadenie zbernice a pokračuje opäť vo vykonávaní programu [51].



Obr. 5.6: Využitie radiča DMA

6 PRAKTICKÉ MERANIE

Pre porovnanie výkonnosti, paralelizmu a vhodnosti pre výpočet na CPU a GPU, bolo vykonané násobenie dvoch matíc o rôznych veľkostiach. Pre otestovanie bolo zvolené API OpenCL najmä vďaka jeho univerzálnosti (konkrétnejší opis je v časti 2.2) a zariadenia, ktoré sú zvlášť uvedené v tab. 6.1 a v tab. 6.2.

Označenie	**Model**	**Architektúra**
CPU 1	Intel® Core™ i3-380M @ 2,53 GHz	Arrandale
CPU 2	Intel® Core™ i5-2410M @ 2,30 GHz	Sandy Bridge
CPU 3	Intel® Core™ i5-3210M @ 2,50 GHz	Ivy Bridge
CPU 4	Intel® Core™ i5-2400S @ 2,50 GHz 3,20 GHz	Sandy Bridge
CPU 5	Intel® Core™ i7-3770 @ 3,40GHz	Ivy Bridge

Tab. 6.1: Popis použitých CPU na testovanie

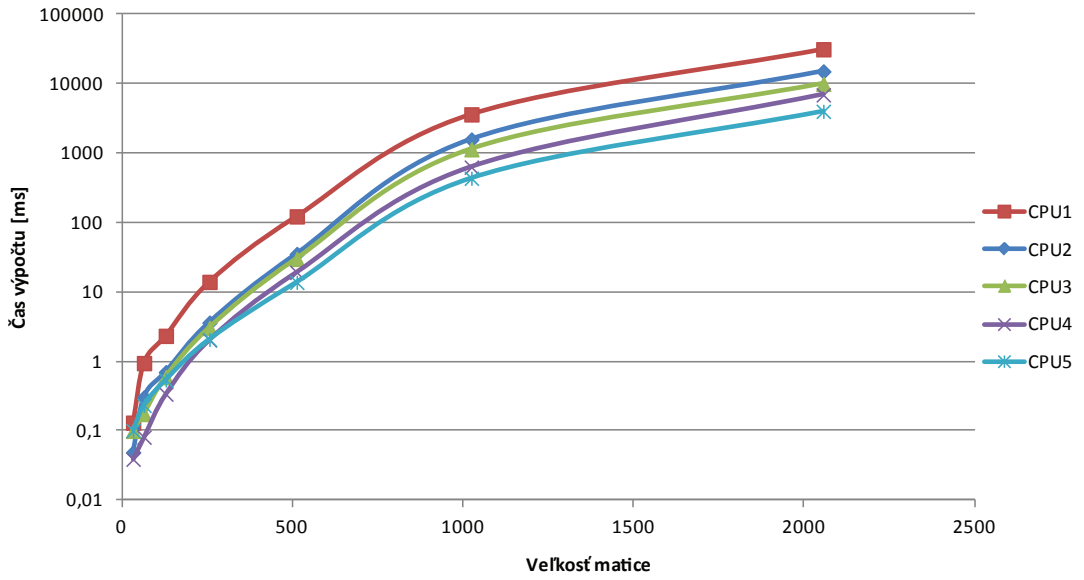
Označenie	**Model**	**Architektúra**
CPU 1	NVIDIA GeForce 310M	GT2xx
CPU 2	Intel® HD Graphics 4000	Ivy Bridge
CPU 3	NVIDIA GeForce GT 540M	Fermi
CPU 4	NVIDIA GeForce GT 635M	Fermi

Tab. 6.2: Popis použitých GPU na testovanie

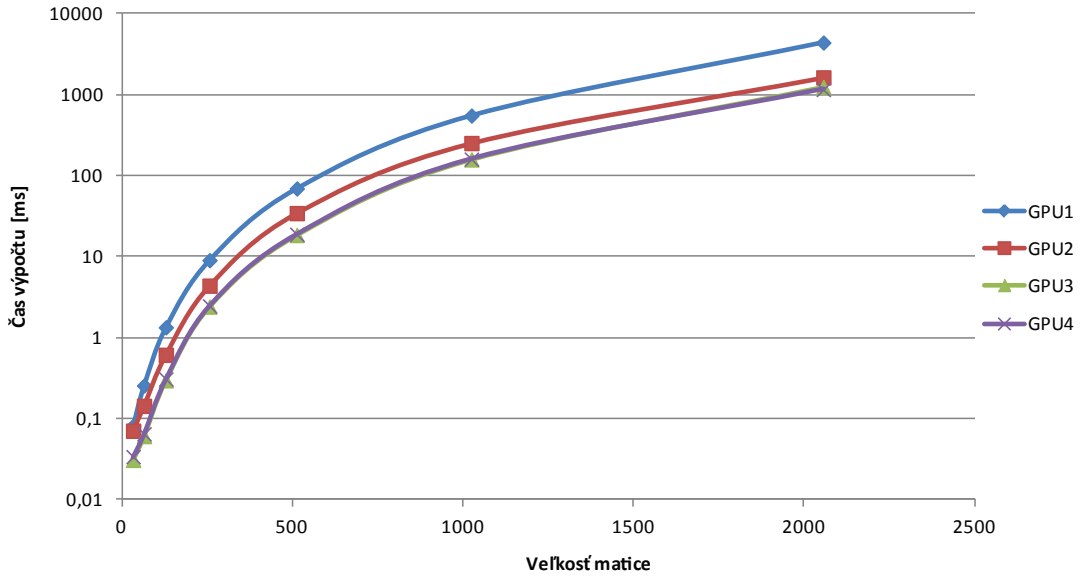
Pre podporu aplikácií OpenCL bolo potrebné nainštalovať SDK, zvlášť pre zariadenia NVIDIA a zvlášť pre zariadenia Intel. Po úspešnej inštalácii je ešte potrebné nastaviť vývojové prostredie, v tomto prípade Visual Studio Professional 2010.

Pred samotnou implementáciou výpočtu boli overené platformy a zároveň zariadenia prislúchajúce pod jednotlivé platformy. V prípade, že bola rozpoznaná viac ako jedna platforma, v našom prípade u zostavy, kde sa nachádzal CPU Intel a GPU NVIDIA, bola platforma definovaná pomocou `cl_platform_id platform_id[2]` a následne sa pre výber zariadenia zvolilo číslo platformy a typ zariadenia CPU alebo GPU, napr. `clGetDeviceIDs(platform_id[1], CL_DEVICE_TYPE_CPU, 1, ...)`.

Pre realizáciu výpočtu boli implementované rovnaké kódy na CPU i GPU. Pre meranie času prenosu dát do zariadenia, výpočtu kernelu a prenosu dát zo zariadenia sa používali eventy, pomocou `clGetEventProfilingInfo`. Pre meranie času natívnej funkcie na CPU sa používala funkcia `QueryPerformanceCounter` a `QueryPerformanceFrequency`.



Obr. 6.1: Porovnanie časovej náročnosti násobenia dvoch matíc o rôznej veľkosti na uvedených typoch CPU

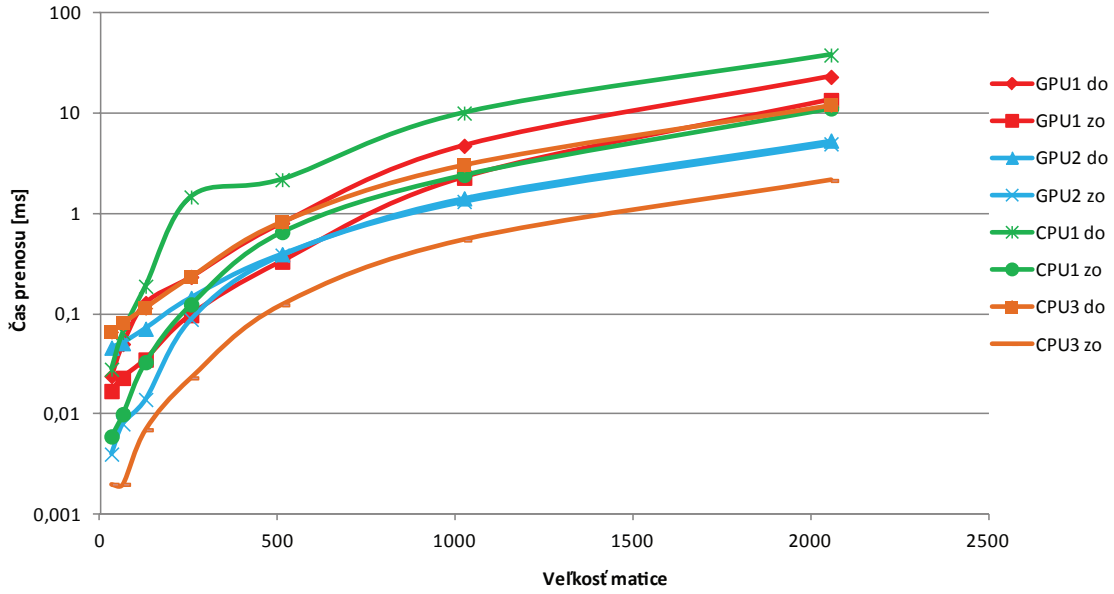


Obr. 6.2: Porovnanie časovej náročnosti násobenia dvoch matíc o rôznej veľkosti na uvedených typoch GPU

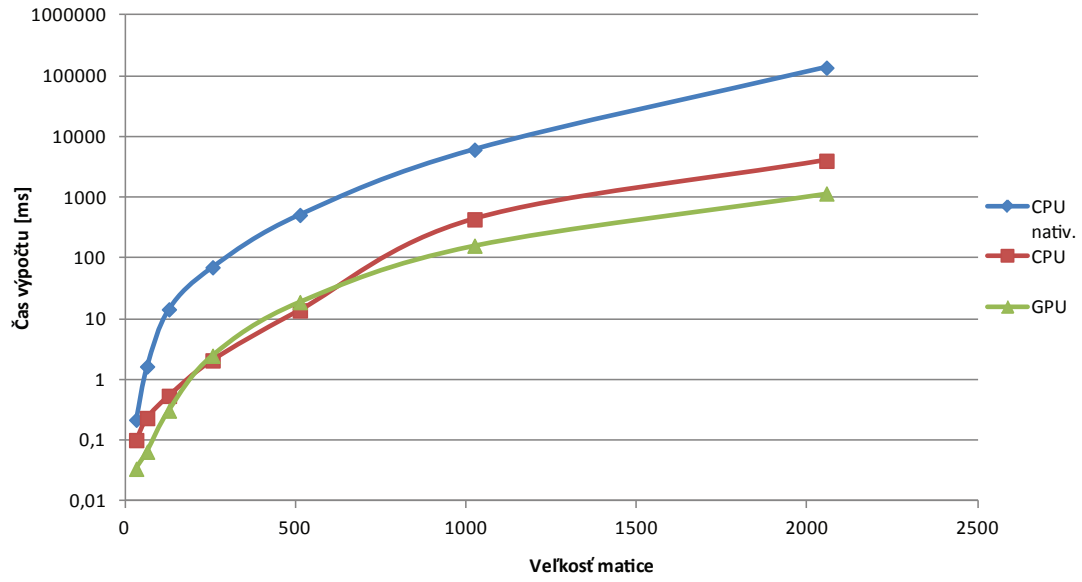
Zobrazené priebehy boli spustené pomocou OpenCL a predstavujú celkový čas vrátane kopírovania vstupných dát do zariadenia, výpočtu kernelu a následného kopírovania výsledkov späť. U kopírovania jednotlivých dát do/zo zariadenia bola použitá blokovacia operácia, čo znamená že sa táto funkcia vráti až potom, čo bude prenos dokončený. Rovnako bola použitá funkcia `clWaitForEvents` pre zaistenie synchronizácie a dokončenia všetkých príkazov vo fronte pred vykonaním ďalšieho.

Na obr. 6.1 sú zobrazené priebehy na CPU, kedy najhoršie výsledky dosahuje CPU1 (Intel Core i3-380m) a najlepšie výsledky dosahuje podľa očakávaní CPU5 (Intel Core i7-3770). Pre porovnanie parametrov má CPU1 2 jadrá, pracovnú frekvenciu 2,53 GHz a 4 paralelné výpočtové jednotky, zatiaľ čo CPU5 má 4 jadrá, pracovnú frekvenciu 3,40 GHz a 8 paralelných výpočtových jednotiek. Okrem samotného času, potrebného pre výpočet, je na tom CPU1 horšie aj v oblasti kopírovania dát do/zo zariadenia. V systéme s CPU1 sa používa systémová pamäť RAM o veľkosti 4 GB, CPU1 disponuje pamäťou typu DDR3, maximálnou veľkosťou pamäte 8 GB, šírkou pásma pamäte 17,1 GB/s a zbernicou PCI-Expres verzie 2.0. Systémová pamäť RAM používaná v zostave s CPU5 je o veľkosti 16 GB, okrem toho má DDR3 novšieho typu a teda aj lepších vlastností, maximálnu veľkosť pamäte 32 GB, šírku pásma pamäte 25,6 GB/s a zbernicu PCI-Expres verzie 3.0. Najpomalší presun dát do/zo zariadenia predstavuje CPU1 (37,879/11,136 ms) a najrýchlejší presun CPU3 (11,951/2,131 ms), namerané pri najvyššej použitej matici 2048x2048, vid obr. 6.3.

Na obr. 6.2 sú zobrazené priebehy na GPU, kedy najhoršie výsledky dosahuje GPU1 (NVIDIA GeForce 310M), slušné výsledky dosahuje integrovaná GPU – GPU2 (Intel HD Graphics 4000), ktorá má 16 paralelných výpočtových jednotiek. Veľmi porovnateľné výkony dosahujú GPU3 (NVIDIA GeForce GT 540M) a GPU4 (NVIDIA GeForce GT 635M), kedy je pri daných výpočtoch o niekoľko milisekúnd lepšia GPU3 do veľkosti matíc 1024x1024, zlom nastáva u veľkosti matice 2048x2048, kedy dosahuje lepší čas pre výpočet GPU4. Obe GPU sú architektúry Fermi, obsahujú zhodne 4 paralelné výpočtové jednotky a dosahujú trojnásobne kratší čas potrebný pre výpočet oproti najrýchlejšiemu CPU5. GPU3 má 96 jadier, frekvenciu jadra 672 MHz, veľkosť pamäte 1536 MB, frekvenciu pamäte 900 MHz a využíva systémovú pamäť RAM o veľkosti 4 GB. GPU4 má však 144 jadier, frekvenciu jadra 660 MHz, veľkosť pamäte 2048 MB, frekvenciu pamäte 1800 MHz a využíva systémovú pamäť RAM o veľkosti 8 GB, čo ju robí vhodnejšou pri paralelnom spracovaní veľkého objemu dát. Oproti GPU1 majú dva krát väčšiu šírku pamäťovej zbernice (64 bit v porovnaní so 128 bit). Najpomalší presun dát do/zo zariadenia predstavuje GPU1 (22,924/13,470 ms) a najrýchlejší pomer dosiahla GPU2 (5,322/4,934 ms), uvedené hodnoty boli namerané pri najväčšej použitej matici 2048x2048, vid obr. 6.3.



Obr. 6.3: Porovnanie časovej náročnosti prenosu dát do/zo zariadenia pri maticiach o rôznej veľkosti na najrýchlejšom a najpomalšom CPU a GPU



Obr. 6.4: Porovnanie časovej náročnosti násobenia dvoch matíc o rôznej veľkosti na najrýchlejšom CPU a GPU a natívnej funkcie na CPU

7 ZÁVER

Práca sa zaoberala problematikou grafických procesorov, ich popisu a vlastností. Na niekoľkých obrázkoch je poukázané na architektúru či princíp fungovania. Zhrnutím časti, v ktorej sú opisované už existujúce a do budúcna plánované architektúry možno tvrdiť, že vývojári prichádzajú stále s novším dizajnom a spôsobom infraštruktúry v oblasti GPU, čím sa zlepšuje ich výkon i využitie, no najmä je tu snaha o uľahčenie práce procesoru ako takému a čo najlepšieho využitia potenciálu GPU. V práci sú na niekoľkých príkladoch zrovnané cenové variácie prevedenia nielen GPU a CPU ale aj iných alternatív a ich vzájomné technické porovnanie a vhodnosť využitia. Aby sa overila teória a princíp fungovania GPU či iných alternatív v oblasti univerzálnych výpočtov, boli navrhnuté API, z ktorých sa v súčasnoti využíva najmä OpenCL a CUDA. OpenCL je otvorený štandard, na vývoji sa podiela nezisková organizácia, v ktorej majú zastúpenie popredné firmy pôsobiace v tejto oblasti a je spustiteľná na rôznych platformách. V porovnaní s tým NVIDIA CUDA funguje výlučne len na platformách firmy NVIDIA. Aj preto bolo OpenCL vybrané pre testovanie na zariadeniach od rôznych výrobcov v experimentálnej časti práce.

V prílohe sú popísané hlavné časti zdrojového kódu. V dôsledku technických problémov nebolo možné otestovať a zmerať čas, ktorý by potrebovalo AMD APU pri použití rovnakého OpenCL kódu ako u ostatných meraných zariadeniach, čo by bolo určite zaujímavé. Škála použitých CPU a GPU však bola pre overenie postačujúca a jednotlivé architektúry dosahovali výsledky podľa predpokladov. Meraním bola potvrdená teória, že GPU sú vhodnejšie pre paralélne spracovanie väčšieho množstva dát. Narastajúcou obtiažnosťou, v tomto prípade veľkosťou matíc a ich následného násobenia, narastal aj rozdiel medzi časom výpočtu u CPU a GPU. OpenCL potvrdil teóriu o programovaní paralelných aplikácii na heterogénnych platformách či využitie GPU i na iné výpočty ako grafické operácie. Pri meraní sa prejavilo niekoľkonásobne rýchlejšie spracovanie výpočtu na CPU oproti natívnej funkcii v C. Všetky merania boli vykonané 5krát a aritmeticky spriemerované.

Oblasť GPGPU sama o sebe ponúka veľké množstvo zaujimavostí a ďalšieho uplatnenia. Otázne je, dokedy budú inovátori a konštruktéri dizajnu GPU schopní posúvať pomyselnú hranicu do ešte väčších extrémov so zámerom ešte väčšej efektivity.

LITERATÚRA

[1] PEDDIE, J. *An Analysis of the GPU Market* [online]. 2011, [cit. 20. 11. 2013]. Dostupné z URL: <http://jonpeddie.com/publications/whitepapers/an-analysis-of-the-gpu-market>.

[2] PALACIOS, J.; TRISKA, J. *A Comparison of Modern GPU and CPU Architectures: And the Common Convergence of Both* [online]. 15. 3. 2011 [cit. 20. 11. 2013]. Dostupné z URL: <http://cours.do.am/ParadigmeAgent/final.pdf>.

[3] NVIDIA: *NVIDIA's Next Generation CUDA Compute Architecture: Fermi* [online]. Verzia 1.1 [cit. 23. 11. 2013]. Dostupné z URL: <http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf>.

[4] NVIDIA: *KEPLER – THE WORLD'S FASTEST, MOST EFFICIENT HPC ARCHITECTURE* [online]. [cit. 23. 11. 2013]. Dostupné z URL: <http://www.nvidia.com/object/nvidia-kepler.html>.

[5] SMITH, R. *NVIDIA Updates GPU Roadmap; Unveils Pascal Architecture For 2016* [online]. 26. 3. 2014 [cit. 28. 5. 2014]. Dostupné z URL: <http://www.anandtech.com/show/7900/nvidia-updates-gpu-roadmap-unveils-pascal-architecture-for-2016>.

[6] NVIDIA: *CUDA C Programming Guide* [online]. posledná aktualizácia 19. 7. 2013 [cit. 24. 11. 2013]. Dostupné z URL: <http://docs.nvidia.com/cuda/cuda-c-programming-guide/>.

[7] AMD: *APU 101: All about AMD Fusion Accelerated Processing Units* [online]. Advanced Micro Devices 2011 [cit. 1. 12. 2013]. Dostupné z URL: <http://developer.amd.com/wordpress/media/2012/10/apu101.pdf>.

[8] ANGELINI, CH. *Intel's Mobile Core i5 And Core i3: Arrandale Is For The Rest Of Us* [online]. posledná aktualizácia 3. 1. 2010 [cit. 1. 12. 2013]. Dostupné z URL: <http://www.tomshardware.com/reviews/mobile-core-i5-arrandale,2522-3.html>.

[9] GAVRICHENKOV, I. *Intel HD Graphics 3000 and Intel HD Graphics 2000 Review* [online]. posledná aktualizácia 22. 2. 2011 [cit. 2. 12. 2013]. Dostupné z URL: <http://www.xbitlabs.com/articles/graphics/display/intel-hd-graphics-2000-3000_2.html>.

[10] GAVRICHENKOV, I. *Intel HD Graphics 4000 and Intel HD Graphics 2500 Review* [online]. posledná aktualizácia 26. 6. 2012 [cit. 2. 12. 2013]. Dostupné z URL: <http://www.xbitlabs.com/articles/graphics/display/intel-hd-graphics-4000-2500.html>.

[11] Intel: *DirectX* Developer's Guide for Intel® Processor Graphics* [online]. Intel 2013 [cit. 3. 12. 2013]. Dostupné z URL: <http://download-software.intel.com/sites/default/files/4th_Generation_Core_Graphics_Developers_Guide_Final.pdf>.

[12] GAVRICHENKOV, I. *AMD Brazos Platform and AMD E-350 (Zacate) CPU Review* [online]. posledná aktualizácia 03. 07. 2011 [cit. 3. 12. 2013]. Dostupné z URL: <http://www.xbitlabs.com/articles/cpu/display/amd-e-350_3.html>.

[13] SHILOV, A. *AMD Formally Launches A-Series „Trinity" Accelerated Processing Units.* [online]. posledná aktualizácia 10. 02. 2012 [cit. 5. 12. 2013]. Dostupné z URL: <http://goo.gl/Ogaisj>.

[14] SHILOV, A. *AMD Introduces G-Series „Kabini" SoC for Embedded Applications.* [online]. posledná aktualizácia 23. 04. 2013 [cit. 5. 12. 2013]. Dostupné z URL: <http://www.xbitlabs.com/news/cpu/display/20130423194313_AMD_Introduces_G_Series_Kabini_SoC_for_Embedded_Applications.html>.

[15] HRUSKA, J. *AMD's next-gen Carrizo APU features leaked, shows greater focus on power efficiency* [online]. posledná aktualizácia 19. 3. 2014 [cit. 28 .5. 2014]. Dostupné z URL: <http://goo.gl/Ydmv0z>.

[16] AMD: *What is Heterogeneous System Architecture (HSA)?* [online]. [cit. 10. 12. 2013]. Dostupné z URL: <http://developer.amd.com/resources/heterogeneous-computing/what-is-heterogeneous-system-architecture-hsa/>.

[17] ELECTRONISTA: *AMD confirms Radeon HD 6000 coming next week* [online]. posledná aktualizácia 15. 10. 2010 [cit. 5. 12. 2013]. Dostupné z URL: <http://www.electronista.com/articles/10/10/15/amd.says.northern.islands.gpus.coming.next.week/>.

[18] SMITH, R. *AMD Reiterates 2013 GPU Plans: Sea Islands & Beyond* [online]. posledná aktualizácia 15. 02. 2013 [cit. 6. 12. 2013]. Dostupné z URL: <http://www.anandtech.com/show/6751/amd-reiterates-2013-gpu-plans-sea-islands-beyond>.

[19] MICK, J. *AMD Soft Launches "Volcanic Islands"GPUs With Programmable Audio in Hawaii* [online]. posledná aktualizácia 25. 09. 2013 [cit. 6. 12. 2013]. Dostupné z URL: <http://www.dailytech.com/AMD+Soft+Launches+Volcanic+Islands+GPUs+With+Programmable+Audio+in+Hawaii/article33449.htm>.

[20] MUJTABA, H. *AMD Launches Next Generation Volcanic Islands (VI) GPUs in 2014 – Successor to Sea Islands* [online]. [cit. 6. 12. 2013]. Dostupné z URL: <http://goo.gl/EeaGF5>.

[21] AMD: *AMD Desktop Graphics* [online]. [cit. 15. 12. 2013]. Dostupné z URL: <http://www.amd.com/US/PRODUCTS/DESKTOP/GRAPHICS/Pages/desktop-graphics.aspx>.

[22] WILLES, D. *AMD Introduces R7 Series Mainstream GPUs* [online]. posledná aktualizácia 10. 10. 2013 [cit. 15. 12. 2013]. Dostupné z URL: <http://www.cpu-world.com/news_2013/2013101002_AMD_Launches_R7_Series_Mainstream_GPUs.html>.

[23] AMD: *AMD Radeon R9 290 Graphics Card Delivers Stunning UltraHD Performance for Just $399* [online]. posledná aktualizácia 05. 11. 2013 [cit. 15. 12. 2013]. Dostupné z URL: <http://ir.amd.com/phoenix.zhtml?c=74093&p=irol-newsArticle&ID=1872344&highlight=>.

[24] Khronos OpenCL Working Group: *The OpenCL Specification v2.0* [online]. 14. 11. 2013 [cit. 12. 2013]. Dostupné z URL: <http://www.khronos.org/registry/cl/specs/opencl-2.0.pdf>.

[25] NVIDIA: *DirectCompute* [online]. [cit. 7. 12. 2013]. Dostupné z URL: <https://developer.nvidia.com/directcompute>.

[26] AMD: *Accelerated Parallel Processing (APP) SDK* [online]. [cit. 7. 12. 2013]. Dostupné z URL: <http://developer.amd.com/tools-and-sdks/opencl-zone/opencl-tools-sdks/amd-accelerated-parallel-processing-app-sdk/>.

[27] CARABANO, J.; FRANCISCO, D. et al. *An Exploration of Heterogeneous Systems* [online]. University of Turku, Finland 7. 2013 [cit. 7. 12. 2013]. Dostupné z URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6581542>.

[28] YIM, L. *APUs for workstations launching* [online]. posledná aktualizácia 7. 8. 2012 [cit. 7. 12. 2013]. Dostupné z URL: <http://semiaccurate.com/2012/08/07/apus-for-workstations-launching/>.

[29] CPU World: *Intel Core i5-4670K specifications* [online]. [cit. 11. 12. 2013]. Dostupné z URL: <http://www.cpu-world.com/CPUs/Core_i5/Intel-Core%20i5-4670K.html>.

[30] AMD: *New AMD A-Series Processors Bring Faster Speeds, High Core Count and AMD Radeon HD 7000 Series Graphics to Do-It-Yourself PC Enthusiasts and Gamers* [online]. posledná aktualizácia 10. 2. 2012 [cit. 11. 12. 2013]. Dostupné z URL: <http://ir.amd.com/phoenix.zhtml?c=74093&p=irol-newsArticle&ID=1740412>.

[31] TECHPOWERUP: *NVIDIA GeForce GTX 770* [online]. [cit. 11. 12. 2013]. Dostupné z URL: <http://www.techpowerup.com/gpudb/1856/geforce-gtx-770.html>.

[32] TECHPOWERUP: *AMD Radeon HD 7970 GHz Edition* [online]. [cit. 15. 12. 2013]. Dostupné z URL: <http://www.techpowerup.com/gpudb/365/radeon-hd-7970-ghz-edition.html>.

[33] CULLINAN, CH.; WYANT, CH.; FRATTESI, T. *Computing Performance Benchmarks among CPU, GPU, and FPGA* [online]. [cit. 10. 12. 2013]. Dostupné z URL: <http://www.wpi.edu/Pubs/E-project/Available/E-project-030212-123508/unrestricted/Benchmarking_Final.pdf>.

[34] KUON, I.; TESSIER, R.; ROSE, J. *FPGA Architecture: Survey and Challenges* [online]. 2008, [cit. 15. 4. 2014]. Dostupné z URL: <https://inst.cs.berkeley.edu/~cs294-59/fa10/resources/kuon-book.pdf>.

[35] DIGILENT: *Atlys™ Spartan-6 FPGA Development Board* [online]. [cit. 10. 12. 2013]. Dostupné z URL: <http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,836&Prod=ATLYS>.

[36] TIŠNOVSKÝ, P. *Od mikrořadičů k digitálním signálovým procesorům (DSP)* [online]. posledná aktualizácia 5. 4. 2011 [cit. 11. 4. 2014]. Dostupné z URL: <http://www.root.cz/clanky/od-mikroradicu-k-digitalnim-signalovym-procesorum-dsp/>.

[37] ANALOG DEVICES: *A BEGINNER'S GUIDE TO DIGITAL SIGNAL PROCESSING* [online]. [cit. 12. 4. 2014]. Dostupné z URL: <http://www.analog.com/en/content/beginners_guide_to_dsp/fca.html>.

[38] SKOLNICK, D.; LEVINE, N. *Why Use DSP?* [online]. [cit. 13. 4. 2014]. Dostupné z URL: <http://www.analog.com/library/analogDialogue/archives/31-1/DSP.html>.

[39] FELLA, J. *ON PERFORMANCE OF GPU AND DSP ARCHITECTURES FOR COMPUTATIONALLY INTENSIVE APPLICATIONS.* Diplomová práca na fakulte Elektrického, Počítačového a Biomedicínskeho inžinierstava Univerzity Rhode Island 2013, Vedúci práce: Dr. Jien-Chung Lo, Dr. Resit Sendag, Dr. Lutz Hamel. [cit. 10. 12. 2013]. Dostupné z URL: <http://digitalcommons.uri.edu/cgi/viewcontent.cgi?article=1002&context=theses>.

[40] TEXAS INSTRUMENTS ICORPORATED: *C553x Ultra-Low-Power DSPs* [online]. [cit. 12. 12. 2013]. Dostupné z URL: <http://goo.gl/htjgTh>.

[41] CERŇANSKÝ, M. *Paralelné programovanie* [online]. Fakulta informatiky a informačných technológií, STU Bratislava [cit. 27. 12. 2013]. Dostupné z URL: <http://www2.fiit.stuba.sk/~cernans/pp/pp_texts/pp_prednaska_2_uvod.pdf>

[42] CERŇANSKÝ, M. *Paralelné programovanie – Analytické modelovanie* [online]. Fakulta informatiky a informačných technológií, STU Bratislava [cit. 29. 12. 2013]. Dostupné z URL: <http://www2.fiit.stuba.sk/~cernans/pp/pp_texts/pp_prednaska_12_analytickemodelovanie.pdf>.

[43] VIDIŠČAK, M. *Message–passing Implementation for Process Functional Language* [online]. [cit. 30. 12. 2013]. Dostupné z URL: <http://conf.uni-obuda.hu/SAMI2005/Vidiscak.pdf>

[44] BROOKWOOD, N. *Everything You Always Wanted to Know About HSA* [online]. Október 2013, [cit. 31. 3. 2014]. Dostupné z URL: <http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/10/Everything_You_Always_Wanted_to_Know_About_HSA_Final2.pdf>.

[45] HSA FOUNDATION: *Tools and SDK's for HSA* [online]. [cit. 1. 4. 2014]. Dostupné z URL: <http://www.hsafoundation.com/hsa-developer-tools/>.

[46] KYRIAZIS, G. *Heterogeneous System Architecture: A Technical Review* [online]. posledná aktualizácia 30. 8. 2012 [cit. 1. 4. 2014]. Dostupné z URL: <http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/hsa10.pdf>.

[47] ROUSE, M. *NUMA (non-uniform memory access)* [online]. posledná aktualizácia Marec 2011 [cit. 4. 4. 2014]. Dostupné z URL: <http://whatis.techtarget.com/definition/NUMA-non-uniform-memory-access>.

[48] ROGERS, P.; MACRI, J.; MARINKOVIC, S. *AMD heterogenous Uniform Memory Access* [prezentácia PowerPoint]. Advanced Micro Devices 2013, posledná aktualizácia 30. 3. 2013 [cit. 4. 4. 2014]. Dostupné z URL: <http://www.slideshare.net/AMD/amd-heterogeneous-uniform-memory-access#>.

[49] KOWALISKI, C. *AMD sheds light on Kaveri's uniform memory architecture* [online]. posledná aktualizácia 29. 4. 2013 [cit. 5. 4. 2014]. Dostupné z URL: <http://techreport.com/news/24737/amd-sheds-light-on-kaveri-uniform-memory-architecture>.

[50] KOTÁSEK, Z. *Principy řízení periferních operací* [prednáška]. Brno: VUT, 2012, [cit. 16. 5. 2014]. Dostupné z URL: <http://www.fit.vutbr.cz/study/courses/IPZ/public/texty/principy/principy_po_2013.pdf>.

[51] KIRK, David B.; Wen-mei W. HWU. *Programming Massively Parallel Processors: A Hands-on Approach*. 2nd ed. San Francisco, California: Morgan Kaufmann, 2012. ISBN 01-241-5992-3.

ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

GPGPU	General-Purpose Computing On Graphics Processing Units
GPU	Graphics Processing Unit
dGPU	dedicated-Graphics Processing Unit
APU	Accelerated Processing Unit
CPU	Central Processing Unit
VGA	Video Graphics Array
EU	Execution Unit
ALU	Aritmetic-Logic Unit
API	Application Programming Interface
OpenCL	Open Computing Language
CUDA	Compute Unified Device Architecture
SDK	Software Development Kit
FLOPs	Float Point Operations Per Second
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Array
RAM	Random Access Memory
TDP	Thermal Design Power
FIR	Infinite Impulse Response
IIR	Finite Impulse Response
H-CUs	HSA-Compute Units
HMMU	HSA Memory Management Unit
SM	Streaming Multriprocessor
HSA	Heterogeneous System Architecture
SVM	Shared Virtual Memory

FIFO	First In First Out
HLSL	High Level Shader Language
APP	Accelerated Parallel Processing
NUMA	Non-Uniform Memory Access
hUMA	heterogenous-Uniform Memory Access
SMP	Symmetric Multiprocessing System
TCU	Throughput Compute Unit
LCU	Latency Compute Unit
MAC	Multiply-And-Accumulate
OpenCV	Open Computer Vision
HSAIL	HSA Intermediate Language
VLIW	Very Long Instruction Word
SIMD	Single Instruction Multiple Data
DMA	Direct Memory Access
I/O	Input/Output
A/D	Analog/Dialog
D/A	Dialog/Analog
PZ	Periférne zariadenie
W	Watt

ZOZNAM PRÍLOH

A Zdrojové kódy **56**

B Obsah priloženého DVD **59**

A ZDROJOVÉ KÓDY

Implementácia násobenia matíc v OpenCL

```
//Deklarácia premenných
cl_device_id zariadenie;
cl_context context;
cl_command_queue command_queue;
cl_mem memobjA = NULL, memobjB = NULL, memobjC = NULL,
 rowA = NULL, colC = NULL;
cl_program program;
cl_kernel kernel;
cl_platform_id platforma[2];
cl_uint ret_num_devices=0, ret_num_platforms=0;
cl_int ret;
cl_event kernelevent,
  zapis1, zapis2, zapis3, zapis4,
  citanie;
cl_ulong time_start, time_end,
    time_szapisu1, time_kzapisu1, time_szapisu2, time_kzapisu2,
    time_szapisu3, time_kzapisu3, time_szapisu4, time_kzapisu4,
    koniec_citania, start_citania;
float total_cistokernel,total_zapis,
    total_zapis1, total_zapis2, total_zapis3, total_zapis4,
    celkove_citanie,
    elapsedTime;
LARGE_INTEGER frequency;
LARGE_INTEGER t1, t2;

   //Získanie dostupných platform
ret = clGetPlatformIDs(2, platforma, &ret_num_platforms);

   //Výber zariadenia zo zvolenej platformy
ret = clGetDeviceIDs(platforma[0], CL_DEVICE_TYPE_GPU, 1,
 &zariadenie, &ret_num_devices);

   //Vytvorenie kontextu
context = clCreateContext(NULL, 1, &zariadenie, NULL, NULL, &ret);
```

```
   //Vytvorenie fronty
command_queue = clCreateCommandQueue(context, zariadenie,
 CL_QUEUE_PROFILING_ENABLE, &ret);

   //Vytvorenie pamäťového bufferu
memobjA = clCreateBuffer(context, CL_MEM_READ_WRITE,
 widthA * heightA * sizeof(float), NULL, &ret);

   //Kopírovanie od hostitela na zariadenie
ret = clEnqueueWriteBuffer(command_queue, memobjA, CL_TRUE,
 0, widthA * heightA * sizeof(int), A, 0, NULL, &zapis1);
clWaitForEvents(1, &zapis1);

   //Vytvorenie a vybudovanie kernel programu, vytvorenie kernelu
program = clCreateProgramWithSource(context, 1,
 (const char **)&source_str,(const size_t *)&source_size, &ret);
ret = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);
kernel = clCreateKernel(program, "matrixMultiplication", &ret);

   //Realizácia kernelu v paralélnej forme
ret = clEnqueueNDRangeKernel(command_queue, kernel, 2, NULL,
 globalThreads, localThreads, NULL, 0, &kernelevent);
clWaitForEvents(1, &kernelevent);

   //Prenos výsledkov z pamäte zariadenia
ret = clEnqueueReadBuffer(command_queue, memobjC, CL_TRUE, 0,
 widthA * heightC * sizeof(float),Res, 0, NULL, &citanie);

   //Blokovanie po dobu vykonania všetkých príkazov fronty
clFinish(command_queue);

   //Odmeraný čas pre zvolený event
clGetEventProfilingInfo(kernelevent, CL_PROFILING_COMMAND_START,
sizeof(time_start), &time_start, NULL);
clGetEventProfilingInfo(kernelevent, CL_PROFILING_COMMAND_END,
sizeof(time_end), &time_end, NULL);
total_cistokernel = time_end - time_start;
```

//Uvoľnenie pamätí

```
ret = clFlush(command_queue);
ret = clFinish(command_queue);
ret = clReleaseKernel(kernel);
ret = clReleaseProgram(program);
ret = clReleaseMemObject(memobjA);
ret = clReleaseMemObject(memobjB);
ret = clReleaseMemObject(memobjC);
ret = clReleaseCommandQueue(command_queue);
ret = clReleaseContext(context);
```

Výpočtový kernel

//Kódová časť pre násobenie matíc

```
__kernel void matrixMultiplication(__global float* A,
 __global float* B, __global float* C, int widthA, int widthB)


{
   int i = get_global_id(0);
   int j = get_global_id(1);
   float value=0;
   for ( int k = 0; k < widthA; k++)
  {
   value = value + A[k + j * widthA] * B[k*widthB + i];
  }
   C[i + widthA * j] = value;
}
```

B OBSAH PRILOŽENÉHO DVD

1. OpenCL – zložka so zdrojovými kódmi používaného projektu v Microsoft Visual Studio 2010
2. Vysledky – zložka s nameranými hodnotami
3. Text – zložka s elektronickou verziou bakalárskej práce