VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

SEMESTRÁLNÍ PRÁCE

Brno, 2016 Bc. Zdeněk Skulínek

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

PARALELIZACE GOERTZELOVA ALGORITMU

SEMESTRÁLNÍ PRÁCE
SEMESTRAL THESIS

AUTOR PRÁCE — **Bc. Zdeněk Skulínek**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Petr Sysel, Ph.D.**
SUPERVISOR

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Semestrální práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Zdeněk Skulínek ***ID:*** 16865
Ročník: 2 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Paralelizace Goertzelova algoritmu

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s možnostmi paralelního zpracování na počítačích typu PC a grafických procesorech. Prostudujte principy a techniky paralelizace používané např. v prostředí Matlab nebo v rámci některé z knihoven pro paralelní zpracování. Vytvořte ukázkové úlohy paralelního zpracování. Porovnejte výpočetní náročnost sekvenčního a paralelního zpracování. Dále se seznamte s Goertzelovým algoritmem a navrhněte, jakým způsobem by bylo možné jej urychlit pomocí paralelního zpracování.

DOPORUČENÁ LITERATURA:

[1] OpenCV 2.4.11 Documentaion. 2014. Dostupné na URL http://docs.opencv.org

[2] SYSEL, P.; RAJMIC, P. Goertzel Algorithm Generalized to Non-integer Multiples of Fundamental Frequency. EURASIP Journal on Advances in Signal Processing. 2012. 2012(56). p. 1 - 20. ISSN 1687-6172.

Termín zadání: 19.9.2016 ***Termín odevzdání:*** 14.12.2016

Vedoucí práce: Ing. Petr Sysel, Ph.D.
Konzultant semestrální práce:

doc. Ing. Jiří Mišurec, CSc., *předseda oborové rady*

ABSTRAKT

Technické problémy znemožňují neustále zvyšovat hodinové frekvence procesorů. Jejich výkon tak v současné době roste díky zvyšování počtu jader. To s sebou přináší nutnost nových přístupů pro programování takovýchto paralelních systémů. Tato práce ukazuje, jak využít paralelismus k číslicovému zpracování signálu. Jako příklad zde bude uvedena implementace Georzelova algoritmu s využitím výpočetního výkonu grafického čipu.

KLÍČOVÁ SLOVA

Goertzelův algoritmus, zpracování signálu, openCL, paralelní výpočet, GPU

ABSTRACT

Technical problems make impossible steadily increase processor's clock frequency. Their power are currently growing due to increasing number of cores. It brings need for new approaches in programming such parallel systems. This thesis shows how to use paralelism in digital signal processing. As an example, it will be presented here implementation of the Georzel's algorithm using the processing power of the graphics chip.

KEYWORDS

Goertzel's algorithm, signal processing, openCL, parallel computing, GPU

SKULÍNEK, Zdeněk *Paralelizace Goertzelova algoritmu*: semestrální projekt. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, Rok. 52 s. Vedoucí práce byl Ing. Petr Sysel, PhDr.

PROHLÁŠENÍ

Prohlašuji, že svůj semestrální projekt na téma „Paralelizace Goertzelova algoritmu“ jsem vypracoval(a) samostatně pod vedením vedoucího semestrálního projektu a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedeného semestrálního projektu dále prohlašuji, že v souvislosti s vytvořením tohoto semestrálního projektu jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Petru Syslovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora(-ky)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Moderní architektury procesorů využívají paralelismus jako důležitou cestu ke zvýšení výpočetního výkonu. Řeší se tím zejména technické problémy při zvyšování hodinových kmitočtů, kde se naráží na fyzikální limity. CPU zvyšují výkon přidáváním jader. GPU se vyvinula z pevně dané renderovací funkcionality do podoby paralelních programovatelných procesorů. Protože dnešní počítače často obsahují vícejádrové CPU a GPU a další procesory, je velmi důležité vzít v úvahu specifika programování pro takovéto paralelní systémy.

Vytváření aplikací pro vícejádrové procesory je oproti tradičním jednojádrovým aplikacím výzva, neboť je potřeba použít zcela jiného přístupu. Více procesorové systémy jsou navíc silně platformně a hardwarově závislé, což činí jejich programování obtížnější.

Mým úkolem je ukázat možnosti dnešních počítačů při zpracování signálu. Jako modelový příklad mám implementovat Goertzelův algoritmus, což je číslicový filtr, na jehož vstupu je číslicový signál a parametr k, udávající konkrétní harmonickou složku spektra signálu navzorkovaného s kmitočtem f_{vz}. Jeho výstupem je pak jediná komplexní hodnota vyjadřující amplitudu na určitém kmitočtu.

1 GOERTZELŮV ALGORITMUS

1.1 Diskrétní Fourierova řada

Máme řadu N hodnot libovolné posloupnosti. Fourier ukázal, že je možno převést ji na N hodnot nějaké frekvenční charakteristiky. *Diskrétní Fourierova řada* přiřazuje časové periodické posloupnosti periody N, posloupnost spektra, rovněž periodickou a rovněž periody N. Následující vzorec pochází z knihy *Systémy a signály* od prof. Smékala[3].

$$S_p[k] = \sum_{n=0}^{N-1} s_p[n] \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N}n}, \tag{1.1}$$

kde k k. komplexní hodnota amplitudy ve spektru, nabývá hodnot $0..N-1, N$ je délka sekvence časové posloupnosti i délka spektra DFŘ. Řada má tedy určitý, omezený počet členů.

1.2 Diskrétní Fourierova transformace - DFT

Na rozdíl od DFŘ, není obraz ani jeho spektrum periodické. DFT přiřazuje časové posloupnosti délky N, posloupnost spektra, také délky N. S pomocí DFŘ by se vytvořil asi takto:

- Zperiodizování průběhu s, kde $s[n + lN] = s_p[n]$
- Výpočet DFŘ.
- Oříznutí spektra na jednorázovou posloupnost.

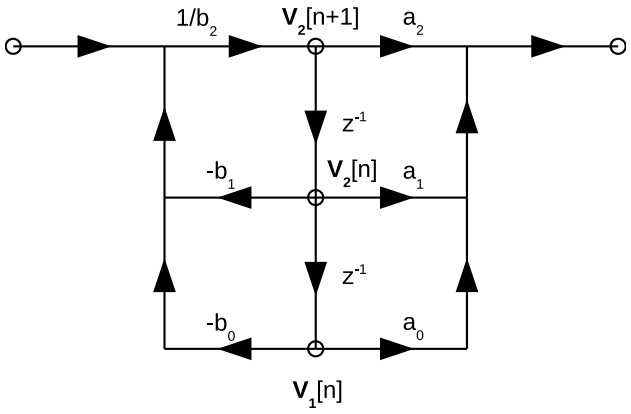
DFT zapisujeme jako

$$S[k] = \sum_{n=0}^{N-1} s[n] \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N}n}. \tag{1.2}$$

I tento vzorec pochází z knihy prof. Smékala ([3]).

1.3 Goertzelův algoritmus

Často je třeba řešit požadavek na výpočet spektrální hustoty energie v určitém malém pásmu hodnot, nejlépe nulové šířky pásma. Šlo by samozřejmě spočítat Fourierovou transformací celé spektrum a vybrat jen ten kmitočet, který je pro nás zajímavý. Další možností je spočítání jedné hodnoty z definice Fourierovy transformace, tak jako ve vzorci pro DFT (1.2). Myšlenkou *Goertzelova algoritmu* je k



Obr. 1.1: Graf signálových toků druhé kanonické struktury.

tomuto účelu použít číslicové filtrace. Goertzel použil druhou kanonickou strukturu (1.3), filtr IIR (na obrázku 1.1).

$$\begin{aligned} v_1[n+1] &= v_2[n] \\ v_2[n+1] &= \frac{1}{b_2}(x[n] - b_1 v_2[n] - b_0 v_1[n]) \\ y[n] &= a_2 v_2[n+1] + a_1 v_2[n] + a_0 v_1[n] \end{aligned} \tag{1.3}$$

1.4 Odvození Goertzelova algoritmu

Následující odvození pochází rovněž z knihy prof. Smékala Systémy a signály [3]. Označme:

$$W_N = \mathrm{e}^{-\mathrm{j}\frac{2\pi}{N}}, \tag{1.4}$$

pak

$$(W_N)^{-kN} = (\mathrm{e}^{-\mathrm{j}\frac{2\pi}{N}})^{-kN} = 1, \tag{1.5}$$

Protože člen (1.5) je jedna, můžeme s ním vynásobit definiční vztah Fourierovy transformace (1.2), aniž by došlo ke změně.

$$S[k] = \sum_{m=0}^{N-1} s[m] W_N^{-kN} = (W_N)^{-kN} \sum_{m=0}^{N-1} s[m] W_N^{-kN} = \\ = \sum_{m=0}^{N-1} s[m] W_N^{-k(N-m)} = \sum_{m=0}^{N-1} s[m] \mathrm{e}^{-\mathrm{j}\frac{2\pi}{N}k(N-m)} \tag{1.6}$$

Konečný tvar vztahu (1.6) zjevně vypadá jako konvoluce vstupní posloupnosti $x[n] = s[n]$ s impulsní charakteristikou číslicového filtru typu IIR

$$h[n] = \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N}} = W_N^{-kN} = 1, n = 0, 1, 2..\infty \tag{1.7}$$

Impulsní charakteristika má komplexní koeficienty, což je zřejmé i z toho, že spektrum DFT je také komplexní. Výstupní odezva potom je

$$y[n] = x[n] * h[n] = \sum_{m=0}^{\infty} s[m] h[n-m] = \sum_{m=0}^{\infty} s[m] W_N^{-k(n-m)} = \\ \sum_{m=0}^{\infty} s[m] \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N}(n-m)} \tag{1.8}$$

Tento vzorec (1.8) je ovšem nekonečná řada. Naštěstí není nutné počítat do nekonečna, jelikož v hledané spektrální složce je jediný vzorek v čase N roven

$$y[N] = \sum_{m=0}^{\infty} s[n] \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N}(N-m)} = \sum_{m=0}^{\infty} s[n] \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N}N} \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N}m} = \sum_{m=0}^{N-1} s[n] \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N}m} = \\ S[k] \tag{1.9}$$

Přenosová charakteristika filtru typu IIR s impulsní charakteristikou (1.7) je prvního řádu

$$H(z) = \sum_{n=0}^{\infty} h[n] z^{-n} = \sum_{n=0}^{\infty} \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N}n} z^{-n} = \sum_{n=0}^{\infty} (\mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N}} z^{-1})^n = \\ \frac{1}{1 - \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N}} z^{-1}} = \frac{z}{z - W_N^{-k}} \tag{1.10}$$

Nevýhodou zcela jistě je skutečnost, že se v přenosové funkci vyskytují komplexní koeficienty. Následující úprava vztahu (1.10) tohle řeší. Daň za možnost práce s jen reálnými čísly je přenosová funkce druhého řádu.

$$H(z) = \frac{1}{1 - W_N^{-k} z^{-1}} = \frac{1}{1 - W_N^{-k} z^{-1}} . \frac{1 - W_N^{k} z^{-1}}{1 - W_N^{k} z^{-1}} = \frac{1}{1 - \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N} z^{-1}}} . \frac{1 - \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N} z^{-1}}}{1 - \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N} z^{-1}}} = \frac{1 - \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N} z^{-1}}}{1 - (\mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N}} + \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N}}) z^{-1} + z^{-2}} = \frac{1 - \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N} z^{-1}}}{1 - 2\cos k\frac{2\pi}{N} z^{-1} + z^{-2}} = \frac{z^2 - \mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N} z}}{z^2 - 2\cos k\frac{2\pi}{N} z + 1} = \frac{a_2 z^2 + a_1 z + a_0}{b_2 z^2 + b_1 z + b_0} \tag{1.11}$$

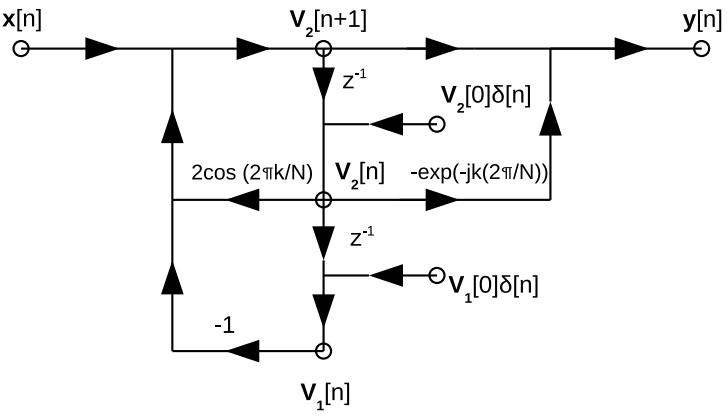
Výsledek porovnáme se stavovými rovnicemi (1.3) A zjistíme tak následující koeficienty.

$$\begin{aligned} b_2 &= 1, \\ b_1 &= -2\cos k\frac{2\pi}{N}, \\ b_0 &= 1, \\ a_2 &= 1, \\ a_1 &= -\mathrm{e}^{-\mathrm{j}k\frac{2\pi}{N}}, \\ a_0 &= 0 \end{aligned} \tag{1.12}$$

Jejich dosazením do (1.3) dostaneme...

$$\begin{aligned} v_1[n+1] &= v_2[n], \\ v_2[n+1] &= x[n] + 2\cos k\frac{2\pi}{N} v_2[n] - v_1[n], \\ y[n] &= v_2[n+1] - (\cos k\frac{2\pi}{N} - \mathrm{j}\sin k\frac{2\pi}{N}) v_2[n], \end{aligned} \tag{1.13}$$

Výhodou této struktury je, že nám stačí ve smyčce počítat jen proměnné v_1 a v_2. Výstup y se počítá jen jednou, při posledním průchodu cyklem, kdy $n = N$. Počáteční hodnoty v_1 a v_2 jsou nulové. Komplexní operace tak bude provedena pouze jednou a to až na závěr.



Obr. 1.2: Graf signálových toků Goertzelova algoritmu ve 2. kanonické struktuře.

2 KNIHOVNA OPENCV

2.1 Úvod do openCV

OpenCV (Open Source Computer Vision Library) dostupná na http://opencv.org [2] je open-source BSD-licencovaná knihovna, která zahrnuje několik set algoritmů z počítačového vidění.
OpenCV má modulární strukturu, jsou v ní zejména moduly uvedené níže:

- *core* - Základní datové struktury a funkce, zejména pak třída `Mat`.
- *imgproc* - Filtrace, geometrické transformace a barevné konverze.
- *video* - Výpočet rozdílu snímků a detekce pohybu.
- *calib3d* - Základní víceobrazové geometrické transformace.
- *features2D* - Detektory charakteristických rysů a jejich srovnávače .
- *objdetect* - Detekce objektů a jejich zařazení do tříd (lidé, oči, obličeje, auta...).
- *highgui* - Jednoduché uživatelské rozhraní a a interface pro zachytávání obrazu, obrazové a video kodeky.
- *gpu* - GPU akcelerované algoritmy ostatních zde uvedených modulů.
- *a další...*

2.2 Základní API

Knihovna open cv je umístěna ve jmenném prostoru `cv`. Je to velmi důležité, protože řada jmen funkcí může být stejná jako u jiných knihoven. Asi nejdůležitější třídou je třída `Mat` reprezentující obecnou matici, se kterou se pracuje téměř stejně jako v programech `matlab`, `octave`. Je tu ale jeden zásadní rozdíl. Třída nemá svoje data, přesněji, několik instancí třídy `Mat` může sdílet stejná data. Je to výhodné, protože ve většině případů se tak zamezí kopírování mnoha bytů. Pokud tedy potřebujeme skutečně Matici zkopírovat, použijeme její členskou metodu `clone`. Je třeba upřesnit, že `Mat` může být i jen určitá oblast jiné `Mat` a přesto sdílejí společná data. `Mat` definuje možnost mít jako data většinu základních typů, samozřejmě jako `unsigned char`, `signed char`, `usigned short`, `float`, `double`, a další. Umí samozřejmě několik kanálů. Uvedu krátký příklad:

```
// vytvoří  8MB velkou matici
Mat A(1000, 1000, CV_64F);

// vytvoří jinou hlavičku pro stejná data
Mat B = A;
```

```
// vytvoří jinou hlavičku, jako třetí řádek matice A.
// žádná data se nekopírují
Mat C = B.row(3);

// teď vytvoří samostatnou kopii matice B
Mat D = B.clone();

// zkopíruje 5tý řádek matice B do matice C,
// matice C nyní bude mít stejná data jako matice A.
B.row(5).copyTo(C);

// teď A a D sdílí stejná data. Matice B a C ale stále ukazují na data
// staré modifikované matice A
A = D;

// nyní smažeme referenci na data u matice B. Matice C
// ale stále ukazuje na data staré modifikované matice
// A. A to přesto, že ukazuje jen na jeden osamocený řádek těchto dat.
B.release();;

// nakonec, vytvořením úplné kopie matice C zaručíme,
// že původní data budou dealokované, protože
// žádné matice na ně již nebude držet odkaz.
C = C.clone();
```

Matice se dá snadno vypsat, stačí jí poslat do výstupního streamu.

```
cout<<R;

//nebo

cout << "R (python)  = " << endl << format(R,"python") << endl << endl;
```

3 KNIHOVNA OPENCL

Tato kapitola je výtahem z dokumentace k openCL ([1]). Úplnou dokumentaci lze najít `kronos.org`.

3.1 Architektura openCL

OpenCL je průmyslový standard pro programování heterogenních skupin CPU, GPU a dalších zařízení organizovaných do jedné platformy. Je to více než jen jazyk. Je to celý framework který obsahuje programovací jazyk, API, knihovny a runtime systém pro podporu vývoje. OpenCL poskytuje nízkoúrovňovou abstrakci hardware plus framework.
V dalších kapitolách blíže popíši následující modely:

- Model platformy.
- Model paměťový.
- Model prováděcí.
- Model programovací.

3.2 Diagram tříd

Obrázek (viz část 3.1) popisuje specifikaci openCL jako UML diagram tříd. Jsou na něm znázorněny jen základní třídy, nikoli jejich atributy.

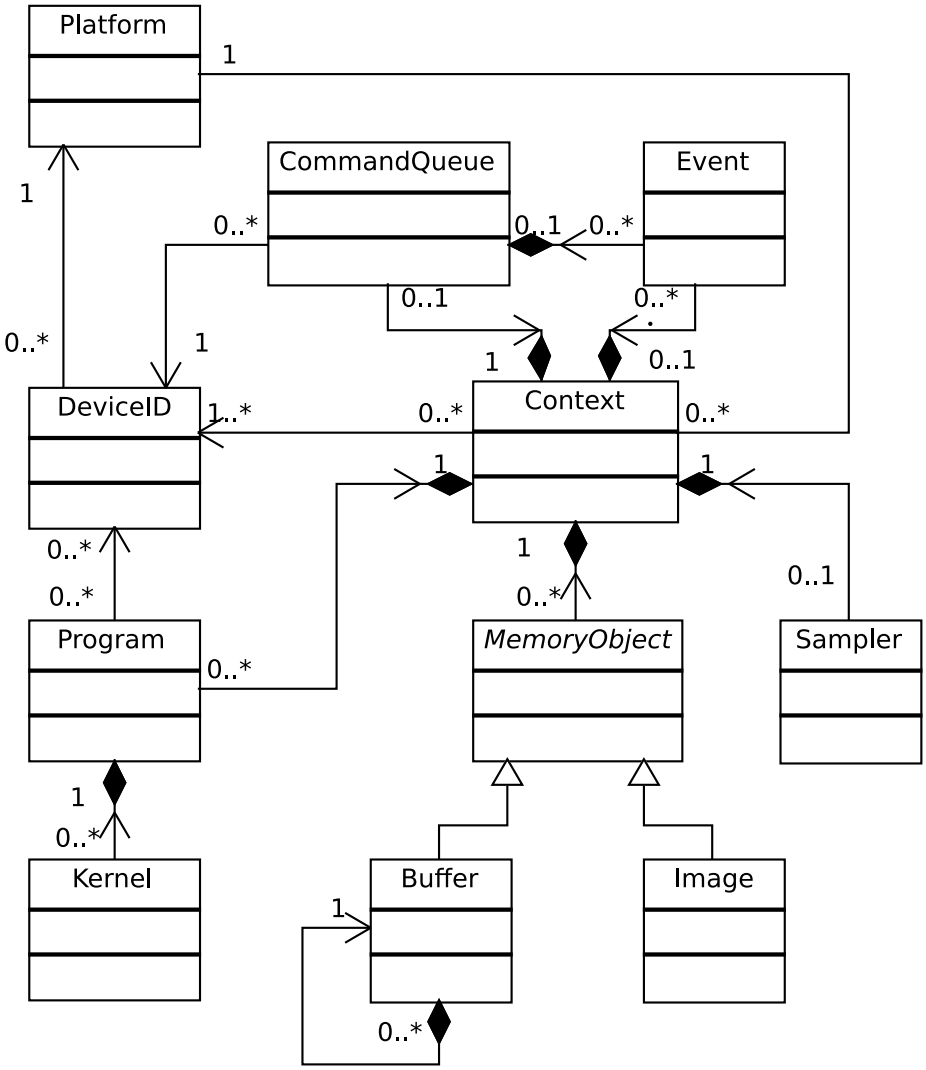
3.3 Model platformy

Model se sestává z *hostitele* připojeného k jednomu nebo více openCL *zařízením* . Toto *zařízení* je dále děleno na jednu nebo více *výpočetních jednotek* a tyto *výpočetní jednotky* jsou dále děleny na jednu nebo více *zpracujících jednotek* .

OpenCL *aplikace* běží na *hostiteli* OpenCL *aplikace* posílá *příkazy* z *hostitele* ke zpracování výpočtů do *zpracujících jednotek* v rámci *zařízení. Zpracující jednotky* v rámci *výpočetní jednotky* provádějí jednu sadu instrukcí jako *SIMD* jednotky, nebo jako *SPMD* jednotky, kde každá *zpracující jednotka* má svůj čítač instrukcí.

3.3.1 Podpora různých verzí

OpenCL je navrženo s podporou více *zařízení* s různými možnostmi společně provozovanými pod jedním *hostitelem*. To ovšem znamená, že každé zařízení může splňovat jinou verzi knihovny openCL. Máme tedy verzi runtime prostředí a zvlášť



Obr. 3.1: UML Diagram tříd

verzi u každého *zařízení*. Dále každé *zařízení* může plnit svou vlastní verzi jazyka openCL C.

3.4 Prováděcí model

Když je *jádro* pověřeno *hostitelem* k provedení, je třeba definovat indexový prostor. Instance *jádra* se nazývá *pracovní položka* a je definována bodem v indexovém prostoru, který definuje *globální ID* pro tuto *pracovní položku*. *Pracovní položky* jsou organizovány v *pracovních skupinách* . *Pracovní skupiny* mají unikání ID *pracovní skupiny*. *Pracovní položky* můžeme v indexovém prostoru identifikovat stejným způsobem, Také mají své *globální ID*. Mimo to je ale můžeme identifikovat dvojicí *globální ID pracovní skupiny* a *lokálním ID* (v rámci této *pracovná skupiny*).
Indexový prostor, který je definován knihovnou openCL, je N-rozměrná pole. Nazývají se *NDRange*. N může být jedna, dvě nebo tři. To znamená, že i indexy jsou N-prvkové. Položky indexu začínají vždy od nuly.
Index tedy definuje určitou *pracovní skupinu* v rámci *výpočetní jednotky*. Stejným způsobem je také definován index *pracovní položky* v rámci *pracovní skupiny*.

3.4.1 Kontext a fronty příkazů

Kontext vytváří a spravuje *hostitel* prostřednicvím funkčních volání openCL API. *Hostitel* v *kontextu* vytvoří zejména *frontu příkazů* , aby mohl zařízení ovládat. Do těchto *front příkazů* zapisuje *příkazy* k provedení. Mezi tyto *příkazy* patří:

- Příkazy k provádění jádra.
- Příkazy pro práci s pamětí.
- Synchronizační příkazy.

V jednom *kontextu* může být více *front příkazů* které jsou prováděny nezávisle na sobě.

3.5 Paměťový model

Pracovní položky rozlišují čtyři druhy pamětí:

- Globální paměť.
- Paměť konstant.
- Lokální paměť.
- Soukromá paměť.

Aplikace běžící na *hostiteli* vytváří *paměťové objekty* v globální paměti.

Paměť *hostitele* a *zařízení* jsou na sobě nezávislé. Nicméně je potřeba aby *zařízení* a *hostitel* spolu nějakým způsobem komunikovali. Toho je docíleno kopírováním

	Global	**Constant**	**Local**	**Private**
Host	Dynamická alokace Přístup pro čtení / zápis	Dynamická alokace Přístup pro čtení / zápis	Dynamická alokace Není přístup	Bez alokace Není přístup
Jádro	Není alokace Přístup pro čtení / zápis	Statická alokace Přístup pro jen čtení	Statická alokace Přístup pro čtení / zápis	Statická alokace Přístup pro čtení / zápis

Tab. 3.1: Typy pamětí v openCL

paměti nebo jejím sdílením. Kopírování může býti jak blokující, tak neblokující. Blokující nebo neblokující může být rovněž mapování paměti. *Aplikace* většinou mapuje paměť na nezbytně nutnou dobu a když jsou všechny operace čtení a zápisu dokončeny, opět paměť odmapuje.

3.6 Programovací model

OpenCL nabízí dva programovací modely, respektive dva základní přístupy. **Datově paralelní model** a **úlohově paralelní model**.

3.6.1 Datově paralelní model

V datově paralelním programovacím modelu provádějí všechny *pracovní položky* v rámci *pracovní skupiny* současně jednu sadu instrukcí. Model je rozdělen na dvě kategorie. U **explicitní** definuje aplikace, jak bude *pracovní skupina* rozdělena na *pracovní položky*. U **implicitní** aplikace pouze definuje, kolik pracovních položek chce použít a rozdělení *pracovní skupiny* na *pracovní položky* nechá na knihovně openCL.

3.6.2 Úlohově paralelní model

V úlohově paralelním modelu je každé *jádro* vykonáváno zcela nezávisle na jiných. Znamená to, že je prováděno v *pracovní skupině*, která má jednu jedinou *pracovní položku*. V tomto modelu je zdůrazněna paralelnost:

- používáním vektorových datových typů.
- řízením více úloh.

3.6.3 Synchronizace

Pracovní položky položky mohou být synchronizovány mezi sebou využitím *bariér*. Pro synchronizaci *pracovních skupin* mezi sebou, žádný takový mechanismus neexistuje.

Příkazy mají také možnost používat *bariéry*. Tyto *bariéry* zajistí, že všechny příkazy zařazené před touto *bariérou* ve *frontě příkazů*, budou provedeny dříve, než se začne s prováděním dalších *příkazů*.

Všechny *příkazy hostitele*, které vkládají *příkazy* do *fronty příkazů*, vracejí objekt **události**. Další *příkaz* ve *frontě příkazů* může na tuto událost čekat.

3.7 Paměťové objekty

Jsou dvě kategorie *paměťových objektů*: *obrázky* a *buffery* . *Buffer* je řada objektů nějakého typu a můžeme k nim přistupovat pomocí rukojeti. Naproti tomu je struktura *obrázku* skryta a k *obrázku* můžeme přistupovat pouze pomocí speciálních funkcí jádra. *Obrázek* nemusí mít stejný formát u *hostitele* a uvnitř *jádra*.

3.8 OpenCL framework

OpenCL *framework* obsahuje tři hlavní části:

- OpenCL platformní vrstva – dovoluje *hostiteli* zjišťovat možnosti *zařízení*, a vytváří *kontext*.
- OpenCL runtime – všechny operace manipulující s kontextem.
- Překladač openCL – vytváří spustitelné objekty obsahující *jádra*. Vychází z ISO C99.

4 MOŽNOSTI PARALELIZACE

4.1 Rozdělení na N-částí a jejich průměrování

Jako první nápad je dát do každého jádra procesoru jeho poměrnou část výpočtu. Dostali bychom N výsledků (kde je N-počet jader procesoru) a z těch by se udělal průměr. Jenže z N-krát méně vzorků pro výpočet by znamenalo také N-krát širší pásmo, které zkoumám. Filtr IIR je totiž pásmová propust. Abych měřil co nejpřesněji, musí být co nejužší. To znamená N-krát menší přesnost. Průměrování to zcela jistě nespraví. Navíc se tím neušetří žádný výpočet. Proto dále nebudu v této úvaze pokračovat.

4.2 Variace stavové proměnné

V našem případě by bylo výhodnější, kdybych mohl počítat s neznámými stavovými proměnnými. Ty totiž počítá jiné jádro. Řekněme, že začínám n-tým vzorkem, počítám pro 4 hodnoty $x[n]$, ve stavových proměnných v_1 a v_2 mám neznámé hodnoty

R a S.

$$
\begin{aligned}
v_1[n] &= R, \\
v_2[n] &= S, \\
2\cos k\frac{2\pi}{N} &= C \\
v_1[n+1] &= v_2[n] = S, \\
v_2[n+1] &= x[n] + 2\cos k\frac{2\pi}{N}v_2[n] - v_1[n] = x[n] + CS - R, \\
v_1[n+2] &= v_2[n+1] = x[n] + CS - R, \\
v_2[n+2] &= x[n+1] + 2\cos k\frac{2\pi}{N}v_2[n+1] - v_1[n+1] = \\
&\quad x[n+1] + C(x[n] + CS - R) - S = \\
&\quad x[n+1] + Cx[n] + (C^2-1)S - CR, \\
v_1[n+3] &= v_2[n+2] = \\
&\quad x[n+1] + Cx[n] + (C^2-1)S - CR, \\
v_2[n+3] &= x[n+2] + 2\cos k\frac{2\pi}{N}v_2[n+2] - v_1[n+2] = \\
&\quad x[n+2] + C(x[n+1] + Cx[n] + (C^2-1)S - CR) \\
&\quad - x[n] - CS + R = \\
&\quad x[n+2] + Cx[n+1] + (C^2-1)x[n] + CS(C^2-2) - \\
&\quad (C^2-1)R \\
v_1[n+4] &= v_2[n+3] = \\
&\quad x[n+2] + Cx[n+1] + (C^2-1)x[n] + CS(C^2-2) - \\
&\quad (C^2-1)R \\
v_2[n+4] &= x[n+3] + 2\cos k\frac{2\pi}{N}v_2[n+3] - v_1[n+3] = \\
&\quad x[n+3] + C(x[n+2] + Cx[n+1] + (C^2-1)x[n] + \\
&\quad CS(C^2-2) - (C^2-1)R) - x[n+1] - Cx[n] - \\
&\quad (C^2-1)S + CR = \\
&\quad x[n+3] + Cx[n+2] + (c^2-1)x[n+1] + C(C^2-2)x[n] + \\
&\quad (C^4 - 2C^2 - C^2 + 1)S - (C^3 - 2C)R
\end{aligned}
\tag{4.1}
$$

Zcela zjevně se dají naše dvě neznámé stavové proměnné nahradit třemi novými, zato známými. Každá rovnice se totiž dá nahradit výrazy:

$$\begin{aligned} v_1[n+1] &= v_2[n] = w_0[n] + w_1[n]S - w_2[n]R, \\ v_2[n+1] &= x[n] + Cv_2[n] - v_1[n] = \\ &\quad w_0[n+1] + w_1[n+1]S + w_2[n+1]R, \end{aligned} \tag{4.2}$$

Vyskytuje se tu řada: $1, C, C^2-1, C^3-2C, C^4-2C^2-C^2+1, C^5-3C^3+C-C^3+2C$

Při bližším prozkoumání této řady vidím, že má určitou zákonitost. Založím si tedy novou stavovou proměnnou, nazvu ji w_3. Začnu...

$$\begin{aligned} w_3[-1] &= 0 \\ w_3[0] &= 1 \\ w_3[1] &= C \\ &... \\ w_3[n+1] &= Cw_3[n] - w_3[n-1] \\ &... \\ w_3[2] &= Cw_3[1] - w_3[0] = C^2 - 1 \\ w_3[3] &= Cw_3[2] - w_3[1] = C(C^2-1) - C = C^3 - 2C \\ w_3[4] &= Cw_3[3] - w_3[2] = C(C^3-2C) - (C^2-1) = \\ &\quad C^4 - 3C^2 + 1 \\ w_3[5] &= Cw_3[4] - w_3[3] = C(C^4-3C^2+1) - (C^3-2C) = \\ &\quad C^5 - 3C^3 + C - C^3 + 2C = C^5 - 4C^3 + 3C \end{aligned} \tag{4.3}$$

Pokud zkombinuji rovnice (4.1) a (4.2) Můžu si vyjádřit nové stavové proměnné

w_1, w_2 a w_3. Začnu s w_0:

$$\begin{aligned} w_0[n] &= 0 \\ w_0[n+1] &= x[n] \\ w_0[n+2] &= x[n+1] + Cx[n] \\ w_0[n+3] &= x[n+2] + Cx[n+1] + (C^2-1)x[n] \\ w_0[n+4] &= x[n+3] + Cx[n+2] + (C^2-1)x[n+1] + C(C^2-2)x[n] \end{aligned} \tag{4.4}$$

Což mohu vyjádřit stavovou proměnnou w_3

$$\begin{aligned} w_0[n+1] &= w_3[0]x[n] \\ w_0[n+2] &= w_3[0]x[n+1] + w_3[1]x[n] \\ w_0[n+3] &= w_3[0]x[n+2] + w_3[1]x[n+1] + w_3[2]x[n] \\ w_0[n+4] &= w_3[0]x[n+3] + w_3[1]x[n+2] + w_3[2]x[n+1] + w_3[3]x[n] \end{aligned} \tag{4.5}$$

Zkusím to opět rozložit podle vzorce (4.3),což vede na původní tvar. Rovnice lze ale přeskládat tak, abych se dostal k rekurentní formuli.

$$
\begin{aligned}
w_0[n+1] &= w_3[0]x[n] = x[n] \\
w_0[n+2] &= w_3[0]x[n+1] + (Cw_3[0] - w_3[-1])x[n] = \\
&\quad Cw_3[0]x[n] + w_3[0]x[n+1] - w_3[-1]x[n] = \\
&\quad Cw_0[n+1] + x[n+1] \\
w_0[n+3] &= w_3[0]x[n+2] + w_3[1]x[n+1] + (Cw_3[1] - w_3[0])x[n] = \\
&\quad w_3[0]x[n+2] + (Cw_3[0] - w_3[-1])x[n+1] + (Cw_3[0]- \\
&\quad w_3[-1])Cx[n] - w_3[0]x[n] = \\
&\quad x[n+2] + C(x[n+1] + Cx[n]) - x[n] = \\
&\quad x[n+2] + Cw_0[n+2] - w_0[n+1] \\
w_0[n+4] &= w_3[0]x[n+3] + w_3[1]x[n+2] + w_3[2]x[n+1]+ \\
&\quad w_3[3]x[n] = x[n+3] + (Cw_3[0] - w_3[-1])x[n+2] + (Cw_3[1] \\
&\quad - w_3[0])x[n+1] + (Cw_3[2] - w_3[1])x[n] = \\
&\quad x[n+3] + Cx[n+2] + (C^2 - 1)x[n+1] + (C^3 - 2C)x[n] = \\
&\quad x[n+3] + C(x[n+2] + Cx[n+1] + C^2x[n] - x[n])- \\
&\quad Cx[n] - x[n+1] = x[n+4] + Cw_0[n+3] - w_0[n+2]
\end{aligned}
\tag{4.6}
$$

Pokračuji s w_2:

$$
\begin{aligned}
w_1[n] &= 1 = w_3[0] \\
w_1[n+1] &= C = w_3[1] \\
w_1[n+2] &= C^2 - 1 = w_3[2] \\
w_1[n+3] &= C^3 - 2C = w_3[3] \\
w_1[n+4] &= C^4 - 3C^2 + 1 = w_3[4]
\end{aligned}
\tag{4.7}
$$

což je již odvozené w_3. Proto tedy mohu napsat

$$
w_1[n+1] = w_3[1] = Cw_1[0] - w_1[-1] \tag{4.8}
$$

A nakonec w_2:

$$\begin{aligned} w_2[n] &= 0 \\ w_2[n+1] &= 1 \\ w_2[n+2] &= C \\ w_2[n+3] &= C^2 - 1 \\ w_2[n+4] &= C^3 - 2C \end{aligned} \tag{4.9}$$

w_2 je posunuté w_3

$$w_2[n+1] = w_3[n] = w_1[n] \tag{4.10}$$

Ještě je tu malý problém. Rekurentní vzorce pro w_0 a w_1 se odkazují nejen na poslední člen historie, ale i předposlední. Proto formálně zavedu nové stavové proměnné. Z pohledu procesoru je totiž stejně jedno, zda nějaké paměťové místo nazývám jako $w_4[n]$ nebo $w_0[n-1]$.

$$\begin{aligned} w_4[n+1] &= w_0[n] \\ w_0[n+1] &= x[n+1] + Cw_0[n] - w_0[n-1] = \\ &\qquad x[n+1] + Cw_0[n] - w_4[n] \\ w_2[n+1] &= w_1[n] \\ w_1[n+1] &= Cw_1[n] - w_2[n] \end{aligned} \tag{4.11}$$

Vztahy pro w_0 a w_4, které jsem odvodil jsou vlastně vztahy Goertzelova algoritmu (1.13). Přibyly jen rovnice pro w_1 a w_2. Zajímavé je že w_1 a w_2 jsou jen konstanty a vůbec nezávisejí na vstupním signálu. To by ovšem znamenalo zjednodušení spojování těch mezivýsledků, co víc, w_1 a w_2 by se vůbec nemusely počítat v tomto algoritmu, protože jsou již předem známé.

Nyní zkusím malou kontrolu. Zkusím, zda výše uvedené vzorce předpovídají další hodnotu posloupnosti v_2, respektive, zda koeficienty u neznámých R a S jsou z řady

w_3.

$$
\begin{aligned}
v_1[n+5] = v_2[n+4] = \\
&x[n+3] + Cx[n+2] + (c^2-1)x[n+1] + C(C^2-2)x[n] + \\
&(C^4 - 2C^2 - C^2 + 1)S - (C^3 - 2C)R \\
v_2[n+5] = x[n+4] + 2\cos k\frac{2\pi}{N}v_2[n+4] - v_1[n+4] = \\
&x[n+4] + C(x[n+3] + Cx[n+2] + (c^2-1)x[n+1] + \\
&C(C^2-2)x[n] + (C^4 - 2C^2 - C^2 + 1)S - (C^3 - 2C)R) - \\
&(x[n+2] + Cx[n+1] + (C^2-1)x[n] + CS(C^2-2) - \\
&(C^2-1)R) = \\
&[n+4] + Cx[n+3] + (C^2-1)x[n+2] + (c^3-2C)x[n+1] + \\
&(C^4 - 2C^2 - C^2 + 1)x[n] + (C^5 - 3C^3 + C - C^3 + 2C)S - \\
&(C^4 - 2C^2 - C^2 + 1)R
\end{aligned}
\tag{4.12}
$$

A na závěr vyzkouším, zda jsem předpověděl dobře novou stavovou proměnnou w_0.

$$
\begin{aligned}
w_0[n+5] = w_3[0]x[n+4] + w_3[1]x[n+3] + w_3[2]x[n+2] + \\
&w_3[3]x[n+1] + w_3[4]x[n] = \\
&x[n+4] + Cx[n+3] + (C^2-1)x[n+2] + \\
&(C^3 - 2C)x[n+1] + (C^4 - 3C^2 + 1)x[n] = \\
&x[n+4] + C(x[n+3] + Cx[n+2] + C^2x[n+1] - x[n+1] + \\
&C^3x[n] - 2Cx[n]) - (x[n+2] + Cx[n+1] + C^2x[n] - \\
&x[n]) = x[n+4] + Cw_1[n+4] - w_1[n+3]
\end{aligned}
\tag{4.13}
$$

Zkusím tedy malou rekapitulaci výše uvedeného.

Mám signál o N diskrétních hodnotách a chci jej počítat na P procesorových jádrech. Pro jednoduchost budu předpokládat že N je násobek P. Vlastně je úplně

jedno, zda bloky, na které N-prvkový signál záhy rozdělím, jsou stejně dlouhé. Každopádně by měly být co možná nejvíce stejně dlouhé, protože celek bude pracovat efektivněji. Jako prerekvizity si spočítám $C = 2\cos 2\pi/N$ a spočítám si w_1 a w_2, které stačí spočítat jednou v případě, že všech P bloků je stejně dlouhých. Na každém kousku signálu spočítám hodnot podle vztahu (4.11) w_0 a w_4, tedy klasickým způsobem Goertzelovým algoritmem, bez výpočtu výstupní proměnné y. Provedu tedy právě N jednoduchých výpočtů. Nyní musím mezivýsledky nějak spojit. U každého mezivýsledku znám w_0, w_1 i w_2. Mohu tedy napsat v souladu se vztahem (4.2).

$$\begin{aligned} R[0] &= 0 \\ S[0] &= 0 \end{aligned} \tag{4.14}$$

$$\begin{aligned} R[1] &= w_0[0] \\ S[1] &= w_4[0] \end{aligned} \tag{4.15}$$

$$\begin{aligned} R[2] &= w_0[1] + w_1[1]R[1] - w_2[1]S[1] \\ S[2] &= w_4[1] + w_1[1]R[1] - w_2[1]S[1] \end{aligned} \tag{4.16}$$

$$\begin{aligned} R[3] &= w_0[2] + w_1[2]R[2] - w_2[2]S[2] \\ S[3] &= w_4[2] + w_1[2]R[2] - w_2[2]S[2] \end{aligned} \tag{4.17}$$

$$\begin{aligned} R[n+1] &= w_0[n] + w_1[n]R[n] - w_2[n]S[n] \\ S[n+1] &= w_4[n] + w_1[n]R[n] - w_2[n]S[n] \end{aligned} \tag{4.18}$$

4.3 Maticové operace

Protože budu používat knihovnu openCL, která obvykle využívá výpočetní výkon grafické karty, je vhodné to nějakým způsobem využít. V grafických výpočtech se používají matice, druhého, třetího a čtvrtého řádu. Grafický čip je tedy pro výpočty s těmito maticemi optimalizován. Proto převedu předchozí úvahu do maticové terminologie. Začnu s přepsáním Goertzelova vztahu (1.13).

Pro $v[1]$:

$$v[1] = \begin{pmatrix} v_1[1] \\ v_2[1] \end{pmatrix} = Av[0] + Bx[0] = \begin{pmatrix} 0 & 1 \\ -1 & C \end{pmatrix} \begin{pmatrix} v_1[0] \\ v_2[0] \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \Big(x[0] \Big) \tag{4.19}$$

a pro $v[2]$

$$\begin{aligned} v[2] = \begin{pmatrix} v_1[2] \\ v_2[2] \end{pmatrix} = A(Av[0] + Bx[0]) + Bx[1] = A^2 \\ \begin{pmatrix} v_1[0] \\ v_2[0] \end{pmatrix} + A \begin{pmatrix} 0 \\ 1 \end{pmatrix} \Big(x[0] \Big) + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \Big(x[1] \Big) \end{aligned} \tag{4.20}$$

a nakonec zobecním. Vztah (4.21) je Goertzelův algoritmus v maticovém zápisu.

$$\begin{aligned} v[n+1] = \begin{pmatrix} v_1[n+1] \\ v_2[n+1] \end{pmatrix} = Av[n] + Bx[n] = \\ \begin{pmatrix} 0 & 1 \\ -1 & C \end{pmatrix} \begin{pmatrix} v_1[n] \\ v_2[n] \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \Big(x[n] \Big) \end{aligned} \tag{4.21}$$

Ten můžu volně přepsat:

$$\begin{aligned} & v[n+k] = \\ & A(A(\dots A(A(A(v[n]) + Bx[n]) + Bx[n+1]) + Bx[n+2]) \dots) + \\ & \qquad\qquad Bx[n+k-1])) + Bx[n+k] \end{aligned} \tag{4.22}$$

Ještě to upravím tak, aby vstup x byl jeden dlouhý sloupcový vektor.

$$v[n+k] = \begin{pmatrix} v_1[n+k] \\ v_2[n+k] \end{pmatrix} = A^k v[n] +$$

$$\begin{pmatrix} A^{k-1}\begin{pmatrix}0\\1\end{pmatrix} & A^{k-2}\begin{pmatrix}0\\1\end{pmatrix} & A^{k-3}\begin{pmatrix}0\\1\end{pmatrix} \dots \begin{pmatrix}0\\1\end{pmatrix} \end{pmatrix} \begin{pmatrix} x[n] \\ x[n+1] \\ x[n+2] \\ \vdots \\ x[n+k] \end{pmatrix} = \tag{4.23}$$

$$A^k v[n] + Dx[n, n+k]^T$$

Tento vzorec, když si představím $k = 3$, dává jasný návod, jak využít matice čtvrtého řádu. Najednou mám čtvrtinu elementárních operací, protože počítám hned se čtyřmi hodnotami. Rovněž popisuje spojovaní mezivýsledků z jednotlivých jader. Strana $Dx[n, n+k]^T$ je Goertzelův vztah, a je k němu připočten předchozí mezivýsledek vynásobený maticí A^{k-1}, tedy konstantou známou před výpočtem. Něco takového jsem již odvodil pro případ počítání po jedné hodnotě x (4.11). Samotná matice D je ale také konstanta, která se dá spočítat předem a při konstrukci programu toho využiji.

4.4 Počítání hodnot na více kmitočtech současně

Předchozí výsledky se dají ještě trochu vylepšit. Prvně nepotřebuji počítat celý horní řádek matice A^k. Dostanu z něj předchozí hodnotu a tu vlastně již znám. Z maticových operací se stanou vektorové.

Protože ale mám možnost počítat matice čtvrtého řádu, proč nevzít rovnou čtyři frekvence současně? Začnu s pravou stranou. Protože jsem v minulé větě uvedl, že stačí použít druhý řádek matice D, tak teď jej 3-krát zkopíruji do ostatních řádků. Stejný ale úplně nebude, jelikož konstanta C v sobě zahrnuje kmitočet, takže čísla v řádcích budou jiná. Na levé straně u matice A^k provedu to stejné. Vektor $v[n]$ rozšířím také 4-krát, ale do šířky. Výsledkem je matice, říkejme jí E, ze které je zajímavý jen vektor na diagonále (kde jsou prováděny stejné operace).

$$v[n+3] = \begin{pmatrix} v_{12}[n+3] \\ v_{22}[n+3] \\ v_{32}[n+3] \\ v_{42}[n+3] \end{pmatrix} =$$

$$diag\left(\begin{pmatrix} a^k_{1,21} & a^k_{1,22} \\ a^k_{2,21} & a^k_{2,22} \\ a^k_{3,21} & a^k_{3,22} \\ a^k_{4,21} & a^k_{4,22} \end{pmatrix} \cdot \begin{pmatrix} v_{1,1} & v_{2,1} & v_{3,1} & v_{4,1} \\ v_{1,2} & v_{2,2} & v_{3,2} & v_{4,2} \end{pmatrix}\right) + \tag{4.24}$$

$$\begin{pmatrix} C_1^3 - 2C_1 & C_1^2 - 1 & C_1 & 1 \\ C_2^3 - 2C_2 & C_2^2 - 1 & C_2 & 1 \\ C_3^3 - 2C_3 & C_3^2 - 1 & C_3 & 1 \\ C_4^3 - 2C_4 & C_4^2 - 1 & C_4 & 1 \end{pmatrix} \cdot \begin{pmatrix} x[n] \\ x[n+1] \\ x[n+2] \\ x[n+3] \end{pmatrix}$$

$diag(A^k v[n])$ jde ještě vyjádřit jako součet dvou vektorů.

$$v[n+3] = \begin{pmatrix} v_{12}[n+3] \\ v_{22}[n+3] \\ v_{32}[n+3] \\ v_{42}[n+3] \end{pmatrix} = \begin{pmatrix} a^k_{1,21} \\ a^k_{2,21} \\ a^k_{3,21} \\ a^k_{4,21} \end{pmatrix} \cdot \begin{pmatrix} v_{1,1} & v_{2,1} & v_{3,1} & v_{4,1} \end{pmatrix} +$$

$$\begin{pmatrix} a^k_{1,22} \\ a^k_{2,22} \\ a^k_{3,22} \\ a^k_{4,22} \end{pmatrix} \cdot \begin{pmatrix} v_{1,2} & v_{2,2} & v_{3,2} & v_{4,2} \end{pmatrix} + \tag{4.25}$$

$$\begin{pmatrix} C_1^3 - 2C_1 & C_1^2 - 1 & C_1 & 1 \\ C_2^3 - 2C_2 & C_2^2 - 1 & C_2 & 1 \\ C_3^3 - 2C_3 & C_3^2 - 1 & C_3 & 1 \\ C_4^3 - 2C_4 & C_4^2 - 1 & C_4 & 1 \end{pmatrix} \cdot \begin{pmatrix} x[n] \\ x[n+1] \\ x[n+2] \\ x[n+3] \end{pmatrix}$$

Kde první index je vždy číslo kmitočtu.

5 ZÁVĚR

V kapitole „Možnosti paralelizace“ (4) jsou uvedeny tři hlavní směry, kterými by se měla diplomová práce ubírat. Prvně bylo ukázáno, že výpočet se dá za cenu mírného zesložitění rozdělit na více procesorových jednotek a následně mezivýsledky spojovat. Dále je v téže kapitole ukázáno, jak využít možnost maticových operací (4.3), které nabízejí dnešní grafické čipy. Jako poslední směr je uvedena možnost počítání několika kmitočtů současně (4.4). V textu je několik otázek, na které lze odpovědět pouze provedením nějakého testu. Například, zda počítání s maticemi velikosti $4 \cdot 4$ skutečně urychlí výpočet oproti popsané skalární variantě (4.3).

LITERATURA

[1] Munshi A.; The OpenCL specification: *Kronos Group online documentation*, 2012, veze 1.2, revize 19, dostupné na `kronos.org`.

[2] OpenCV.org: *OpenCV 2.4.11 Documentaion*, 2014, dostupné na `opencv.org`.

[3] SMÉKAL, Z.: Systémy a signály. *Sdělovací technika* , 2013, Praha, ISBN 978-80-86645-23-0.

[4] SYSEL, P.; RAJMIC, P.: Goertzel Algorithm Generalized to Non-integer Multiples of Fundamental Frequency. *EURASIP Journal on Advances in Signal Processing*, 2012. 2012(56). p. 1 - 20. ISSN 1687-6172.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

DSP	číslicové zpracování signálů – Digital Signal Processing
f_{vz}	vzorkovací kmitočet
CPU	Central Processing Unit – centrální procesorová jednotka
GPU	Graphics Processing Unit – grafická procesorová jednotka

SEZNAM PŘÍLOH

A SLOVNÍČEK POJMŮ KNIHOVNY OPENCL

Tato kapitola je výtahem z dokumentace k openCL ([1]). Úplnou dokumentaci lze najít `kronos.org`.

A.1 Application – Aplikace

Kombinace programů běžících, jak na *hostitelském zařízení (host device)* (viz termín A.24), tak na openCL *zařízení (device)* (viz termín A.16).

A.2 Blocking a Non-Blocking Enqueue API calls – Blokující a neblokující příkazy

Neblokující příkaz (command) (viz termín A.6) ve frontě volání vloží *příkaz* (viz termín A.6) do *příkazové fronty* (viz termín A.7) a okamžitě vrátí řízení *hostiteli* (viz termín A.24). *Blokující* příkaz ve frontě nevrátí řízení hostiteli, dokud není *příkaz* (viz termín A.6) proveden.

A.3 Barrier – Bariéra

Jsou dva druhy *bariér (barriers)* (viz termín A.3) – bariéra *fronty zpráv* (viz termín A.7) a bariéra *pracovní skupiny (work-group)* (viz termín A.57).

- *OpenCL API* poskytuje funkci zařazení *bariéra fronty zpráv* (viz termín A.8). Tento příkaz zajistí, že předchozí příkazy jsou provedeny před tím, než začne příkaz následující.
- *OpenCL C* programovací jazyk poskytuje vestavěnou funkci *bariéra pracovní skupiny* (viz termín A.58). Toto bariéra může býti volána *jádrem (kernel)* (viz termín A.30) pro zajištění synchronizace mezi *pracovními členy (work-items)* (viz termín A.59) v *pracovních skupinách* (viz termín A.57) provádějících *jádra* (viz termín A.30). Všechny *pracovní členy* v *pracovní skupině* musí provést tuto synchronizaci, než bude možno pokračovat v provádění následujících příkazů.

A.4 Buffer object – Buffer

Buffer je paměťový objekt, ve kterém je uložena souvislá řada bytů. Tento buffer je přístupný použitím ukazatele z *jádra* prováděném na *zařízení*. S těmito buffery může

býti manipulováno použitím OpenCL API funkcí. Buffer zapouzdřuje následující informace:

- Velikost v bytech.
- Parametry popisující uložení v paměti a ve které oblasti je alokován.
- Buffer data.

A.5 Built-in kernel – Vestavěné jádro

Vestavěné jádro je je *jádro* (viz termín A.30) prováděné na openCL *zařízení* (viz termín A.16) nebo *speciálním zařízení (custom device)* (viz termín A.14) pevně daným hardwarem nebo firmwarem. Aplikace může používat *vestavěná jádra* nabízená *zařízeními* nebo *speciálními zařízeními*. *Objekty programů (program objects)* (viz termín A.44) mohou obsahovat jen *jádra* napsané v jazyce openCL C nebo *vestavěná jádra*, ale nikdy ne obojí. Více viz *jádra* (viz termín A.30) a *programy* (viz termín A.43).

A.6 Command – Příkaz

Operace v openCL jsou posílány k provedení do *fronty příkazů (command queue)* (viz termín A.7). Například openCL *příkazy* (viz termín A.6) pověřují *jádra (kernels)* (viz termín A.30) k provedení na *výpočetním zařízení* (viz termín A.16), manipulaci s paměťovými objekty atd.

A.7 Command Queue – Fronta příkazů

Fronta příkazů (command queue) (viz termín A.7) je objekt ve kterém jsou uloženy příkazy, které budou vykonány na určitém *zařízení* (viz termín A.16). *Fronta příkazů* (viz termín A.7) je vytvořena na určitém *zařízení* s určitým *kontextem (context)* (viz termín A.13). *Příkazy* (viz termín A.6) jsou zařazeny do *fronty příkazů* (viz termín A.7) v pořadí, ale vykonat se mohou v tomto pořadí nebo v pořadí jiném. Více viz *provádění v pořadí (in-order execution)* (viz termín A.29) a *provádění mimo pořadí (out-order execution)* (viz termín A.38).

A.8 Command Queue Barrier – Bariéra fronty příkazů

Více na *bariéra (barrier)* (viz termín A.3).

A.9 Compute device memory – Paměť výpočetního zařízení

Jedno *zařízení* (viz termín A.16) může mít připojeno jednu nebo více pamětí.

A.10 Compute unit – Výpočetního jednotka

OpenCL *zařízení* (viz termín A.16) může mít jednu nebo více *výpočetních jednotek (compute units)* (viz termín A.10). *Pracovní skupina (work-group)* (viz termín A.57) je provedena na jedné výpočetní jednotce. Výpočetní jednotka je tvořena jedním nebo více *procesorovým prvkem (processing element)* (viz termín A.42) a *lokální pamětí (local memory)* (viz termín A.33). Výpočetní jednotka také může obsahovat filtr textur, který může být přístupný z *procesorových prvků* (viz termín A.42).

A.11 Concurrency – Souběžnost

Vlastnost systému, ve kterém je nějaká skupina úloh současně aktivní a provádí nějakou akci. Pro využití *souběžného provádění* (viz termín A.11) programů, musí programátor identifikovat možné problémy *souběžného* zpracování, zahrnout je do svých zdrojových kódů a využít synchronizačních možností zařízení.

A.12 Constant memory – Paměť konstant

Oblast v paměti přístupná *jádru* (viz termín A.30), která je za běhu *jádra* konstantní. Tuto paměť alokuje i inicializuje *hostitel* (viz termín A.24).

A.13 Context – Kontext

Prostředí, ve kterém se provádí *jádra* (viz termín A.30) a doména, ke které se váží synchronizační a *paměťové objekty* (viz termín A.35). Kontext zahrnuje množinu *zařízení*, příslušnou paměť k těmto *zařízením*, proměnné vázající se k těmto pamětem a jednu nebo více *front příkazů* (viz termín A.7).

A.14 Custom device – Speciální zařízení

Speciální zařízení má podporu openCL runtime, ale není možné pro něj psát programy v openCL C. Tyto zařízení jsou obvykle vysoce efektivní pro určité typy úloh.

Mohou mít vlastní překladač. Pokud ho nemají, je možné spouštět pouze *vestavěná jádra* (viz termín A.5).

A.15 Data parallel programming model – Datový paralelní programovací model

Tradičně je tím myšlen model, kde je jeden program spuštěn souběžně na řadě stejných objektů.

A.16 Device – Zařízení

Zařízení je množina *výpočetních jednotek* (viz termín A.10). Každá jednotka má *fronty příkazů* (viz termín A.7). Příkazy mohou například spouštět *jádra* (viz termín A.30) nebo zapisovat a číst *paměťové objekty* (viz termín A.35). Příkladem takovéhoto *zařízení* je *GPU*, vícejádrové *CPU* nebo například *DSP*.

A.17 Event object – Objekt události

Objekt události zapouzdřuje status operace, jako například nějakého *příkazu* (viz termín A.6). Může být využit k synchronizaci operací v rámci *kontextu* (viz termín A.13).

A.18 Event wait list – Seznam čekajících událostí

Je seznam *objektů událostí* (viz termín A.17) který může být využit k řízení začátku provádění dílčího *příkazu* (viz termín A.6).

A.19 Framework – Framework

Je několik softwarových balíků umožňujících vývoj software. Obvykle zahrnuje knihovny, API, *runtime* prostředí, překladače a podobně.

A.20 Global ID – Globální ID

Global ID jednoznačně specifikuje *pracovní položka* (viz termín A.59) a je založen na počtu globálních *pracovních položek* a je dán před spuštěním *jader* (viz termín

A.30). Global ID je N-rozměrný vektor začínající (0,0,...0). Více na *Lokální ID* (viz termín A.32).

A.21 Global memory – Globální paměť

Tato paměť je přístupná *pracovním položkám* (viz termín A.59) prováděných v rámci *kontextu* (viz termín A.13). Z *hostitele* může být přístupný pomocí *příkazů* jako číst, zapisovat a mapovat.

A.22 GL share group – Sdílená GL skupina

Sdílená GL skupina je objekt pro správu sdílených prostředků s knihovnami openGL a openGL ES. Například tam patří textury, buffery, framebuffery a je asociována s jedním nebo více openGL kontexty. Sdílený GL objekt je obvykle zástupný objekt na není přímo přístupný.

A.23 Handle – Rukojeť

Zástupný typ ukazující na objekty ve správě openCL. Každá operace na nějakém objektu je určena *rukojetí* na tento objekt.

A.24 Host – Hostitel

Hostitel komunikuje s openCL API prostřednicvím *kontextu* (viz termín A.13).

A.25 Host pointer – Ukazatel hostitele

Ukazatel na paměť, která je ve virtuálním adresovém prostoru *hostitele*.

A.26 Illegal – Nedovolený

Chování systému, které není výslovně dovoleno, bude nahlášeno jako chyba systému openCL.

A.27 Image object – Obrázek

Obrázek (viz termín A.27) má dvou nebo tří dimenzionální pole, rozměry atd. Data lze modifikovat pomocí funkcí čtení a zápisu. Operace čtení využívá *vzorkovač (sampler)* (viz termín A.50).

A.28 Implementation defined – Implementačně závislé

Chování, kde je výslovně povolena nějaká odlišnost od openCL rozhraní. V těchto případech je povinná dobrá dokumentace výrobce.

A.29 In-order execution – Provádění v pořadí

Model provádění, kde *příkazy* ve *frontě příkazů* (viz termín A.7) jsou prováděny jeden za druhým, *příkaz* (viz termín A.6) je vždy dokončen před započetím *příkazu* následujícího.

A.30 Kernel – Jádro

Kernel (jádro) je funkce definovaná v programu a označená identifikátorem `kernel` nebo `__kernel`.

A.31 Kernel object – Objekt jádra

Objekt jádra (kernel object) zapouzdřuje *jádro* (viz termín A.30) (funkci definovanou s kvalifikátorem `__kernel`) a vstupní argument.

A.32 Local ID – Lokální ID

Lokální ID jednoznačně specifikuje *pracovní položka* (viz termín A.59) v rámci *pracovní skupiny* (viz termín A.57). Lokální ID je N-rozměrný vektor začínající (0,0,...0). Více na *Globální ID* (viz termín A.20).

A.33 Local memory – Lokální paměť

Tato paměť je přístupná *pracovním položkám* (viz termín A.59) prováděných v rámci *pracovní skupiny* (viz termín A.57).

A.34 Marker – Značka

Je *příkaz* zařazený ve *frontě příkazů* (viz termín A.7), který označí všechny předchozí *příkazy* (viz termín A.6) ve *frontě příkazů* a jeho výsledkem je *událost* (viz termín A.17), kterou může poslouchat aplikace, a tak počkat na všechny *příkazy* zařazené před *značkou* ve *frontě zpráv*.

A.35 Memory objects – Paměťové objekty

Paměťový objekt je *rukojeť* – ukazatel na nějakou oblast v globální paměti. Používá se *počítadlo odkazů (reference counting)* (viz termín A.45). Více na *buffer* (viz termín A.4) nebo *obrázek* (viz termín A.27).

A.36 Memory regions (pools) – Paměťové oblasti

V openCL jsou různé adresové prostory. Paměťové oblasti mohou překrývat ve fyzické paměti, ale openCL je bere jako logicky různé. Paměťové oblasti mohou být označeny jako `private`, `local`, `constant` a `global`.

A.37 Object – Objekt

Objekty jsou abstraktní reprezentace *zdrojů (resources)* (viz termín A.47) a může s nimi býti manipulováno pomoci openCL API. Například lze uvést *objekt programu* (viz termín A.44), *objekt jádra* (viz termín A.31) a *paměťový objekt* (viz termín A.35).

A.38 Out-of-order execution – Provádění mimo pořadí

Model provádění, kde *příkazy* (viz termín A.6) vložené ve *frontě příkazů* (viz termín A.7) jsou prováděny v libovolném pořadí. Je ovšem respektován *seznam čekajících*

událostí (viz termín A.18) a *bariéra fronty zpráv* (viz termín A.8). Viz *provádění v pořadí* (viz termín A.29).

A.39 Parent device – Rodičovské zařízení

OpenCL *zařízení* (viz termín A.16) může být rozděleno na další *subzařízení* (viz termín A.53). Protože i *subzařízení* může být dále děleno na *subzařízení*, nemusí být *rodičovské zařízení* vždy *kořenové zařízení (root device)* (viz termín A.49).

A.40 Platform – Platforma

Je *hostitel* (viz termín A.24) a jedno nebo několik openCL *zařízení* (viz termín A.16) které mohou být ovládány openCL *frameworkem* (viz termín A.19). Tento *framework* umí mezi zařízeními sdílet zdroje a provádět *jádra* (viz termín A.30) na *zařízení* v rámci *platformy*.

A.41 Private memory – Soukromá paměť

Oblast v paměti vyhrazená výhradně pro *pracovní položku* (viz termín A.59). Proměnné zde definované nemohou býti viděny z jiné *pracovní položky*.

A.42 Processing element – Zpracující jednotka

Virtuální skalární procesor. *Pracovní položka* (viz termín A.59) může být prováděna na jedné nebo více *zpracujících jednotkách* (viz termín A.10).

A.43 Program – Program

OpenCL *program* (viz termín A.43) se skládá z množiny *jader* (viz termín A.30). *Programy* mohou také obsahovat pomocné funkce volané z `__kernel` funkcí a konstantní data.

A.44 Program object – Programový objekt

Programový objekt zapouzdřuje následující informace:

- Ukazatel na odpovídající *kontext* (viz termín A.13).
- Zdrojový text nebo binární kód programu.

- Poslední úspěšně přeložený proveditelný program, seznam *zařízení* (viz termín A.16), pro které byl přeložen, nastavení překladače a log.
- Několik připojených *objektů jader* (viz termín A.31).

A.45 Reference count – Počítadlo odkazů

Životní cyklus objektů v openCL je dán *počítadlem odkazů* (viz termín A.45) na tento objekt. Když vytvoříte openCL objekt, *počítadlo odkazů* se nastaví na 1. Příslušný *retain (přivlastni)* (viz termín A.48) jako `clRetainContext`, `clRetainCommandQueue` zvyšují hodnotu tohoto počítadla. Volání *release (uvolni)* (viz termín A.48) jako například `clReleaseContext` nebo `clReleaseCommandQueue` toto počítadlo snižují o 1. Když *počítadlo odkazů* klesne na nulu, objekt je dealokován.

A.46 Relaxed consistency – Rozvolněná soudržnost

Model soudržnosti paměti, ve kterém je viditelnost pro jiné *pracovní položky* (viz termín A.59) nebo *příkazy* (viz termín A.6) různá. Výjimkou jsou synchronizační objekty, jako třeba *bariéry* (viz termín A.3).

A.47 Resource – Zdroj

Je třída *objektů* (viz termín A.37) definovaná v openCL. Instance *zdroje* je *objekt*. Zdroji obyčejně rozumíme *kontexty* (viz termín A.13), *fronty příkazů* (viz termín A.7), *programové objekty* (viz termín A.44), *objekty jádra* (viz termín A.31) a *paměťové objekty* (viz termín A.35). Výpočetní *zdroje* jsou hardwarové součásti využívající čítač instrukcí. Jako příklad lze uvést *hostitele* (viz termín A.24), *zařízení* (viz termín A.16), *výpočetní jednotky (compute units)* (viz termín A.10) nebo *zpracující jednotky (processing elements)* (viz termín A.42).

A.48 Retain, release – Přivlastni, uvolni

Pojmenování akce inkrementace(*retain*) (viz termín A.48), nebo dekrementace(*release*) (viz termín A.48) *počítadla odkazů* (viz termín A.45). Zajišťuje, že objekt nebude smazán, dokud nejsou hotovy všechny procesy, které jej používají. Viz *počítadlo odkazů* (viz termín A.45).

A.49 Root device – Kořenové zařízení

Kořenové zařízení je openCL *zařízení* (viz termín A.16), které není rozdělená část jiného *zařízení*. Více viz *zařízení* a *rodičovské zařízení* (viz termín A.39).

A.50 Sampler – Vzorkovač

Objekt (viz termín A.37), který popisuje jak se má navzorkovat obrázek, když je čten *jádrem* (viz termín A.30). Funkce čtoucí obrázky mají *vzorkovač* jako svůj argument. *Vzorkovačem* se definuje adresní mód, což je chování v případě, že se souřadnice v obrázku nacházejí mimo jeho rozměry. Jako další jsou případy, kdy jsou souřadnice obrázku normalizované a nenormalizované hodnoty a filtrační mód.

A.51 SIMD:Single instruction multiple data – SIMD

Programovací model, kde *jádro* (viz termín A.30) je prováděno současně na více *zpracujících jednotkách* (viz termín A.10), každá z nich má svá vlastní data a společně sdílejí jeden čítač instrukcí. Všechny *zpracující jednotky* provádějí naprosto stejnou množinu instrukcí.

A.52 SPMD: Single program multiple data – SPMD

Programovací model, kde *jádro* (viz termín A.30) je prováděno současně na více *zpracujících jednotkách* (viz termín A.10). Každá z nich má svůj vlastní čítač instrukcí. *Zpracující jednotky* mohou mít množinu instrukcí různou.

A.53 Sub-device – Subzařízení

OpenCL *zařízení* (viz termín A.16) může být rozděleno do více *subzařízení* podle rozdělovacího plánu. Nová *subzařízení* alias specifické skupiny *výpočetních jednotek* (viz termín A.10) v rámci *rodičovského zařízení* (viz termín A.39) mohou býti použita stejně jako jejich *rodičovská zařízení*. Rozdělení *rodičovského zařízení* na *subzařízení* nezničí toto *rodičovské zařízení*, které může být průběžně dále používáno současně se *subzařízeními*. Viz také *zařízení* (viz termín A.16), *rodičovské zařízení* (viz termín A.39) a *kořenové zařízení* (viz termín A.49).

A.54 Task parallel programming model – Paralelně úlohový programovací model

Programovací model, kde jsou výpočty vyjádřeny podmínkami více *souběžných* (viz termín A.11) úloh. Úloha je zde *jádro* (viz termín A.30) v jedné *pracovní skupině* (viz termín A.57) o velikosti jedna. Souběžné úlohy mohou provádět různá *jádra*.

A.55 Thread-safe – Vláknově bezpečné

OpenCL je považováno za *vláknově bezpečné*, pokud interní stav, který je spravován knihovnou openCL, zůstává konzistentní i v případě, že *hostitel* (viz termín A.24) volá z více vláken. OpenCL API, které jsou *vláknově bezpečné*, dovolují aplikaci volat tyto funkce z více současně probíhajících vláken bez použití *mutexu*. Říkáme pak, že jsou i *reentratní (re-entrant-safe)*.

A.56 Undefined – Nedefinováno

Chování volání openCL API, kde *vestavěná funkce* (viz termín A.5) uvnitř *jádra* (viz termín A.30) není výslovně definována. Po implementaci openCL není požadováno definovat, co se stane, když je tato funkce použita.

A.57 Work group – Pracovní skupina

Skupina *pracovních položek* (viz termín A.59), které jsou vykonávány na jedné *výpočetní jednotce* (viz termín A.10). *Pracovní položky* ve skupině provádějí stejné *jádro* (viz termín A.30) a sdílejí *lokální paměť* (viz termín A.33) a *bariéry pracovní skupiny* (viz termín A.58).

A.58 Work group barrier – Bariéra pracovní skupiny

Viz *bariéra* (viz termín A.3).

A.59 Work-Item – Pracovní položka

Jedna z množiny paralelního provádění *jádra* (viz termín A.30) spuštěná na *zařízení* (viz termín A.16) *příkazem* (viz termín A.6). *Pracovní položka* je prováděna

na jedné nebo více *zpracujících jednotkách* (viz termín A.10) jako část provádění *pracovní skupiny* (viz termín A.57) prováděné na *výpočetní jednotce*. *Pracovní položka* je rozeznávána mezi jinými ve skupině svým *lokálním ID* (viz termín A.32) a *globálním ID* (viz termín A.20).

B INSTALACE PROJEKTU

B.1 Instalace GNU C++

Přestože knihovny openCV, openCL jsou multiplatformní, rozhodl jsem se pro operační systém Linux, distribuce Mint 17. Vyhnul jsem se konstrukcím platformně závislých, takže je možná přenositelnost na jiné operační systémy. Jako překladač jsem zvolil GCC, a to ze stejného důvodu jako předchozí, je platformně nezávislý a je v každé distribuci Linuxu jako jeden ze základních balíčků. Mě stačilo k instalaci napsat:

```
sudo apt-get install build-essential
```

a potom...

```
sudo apt-get install g++
```

Pro Windows existuje disrtibuce GNU cygwin, kterou je možno stáhnout a nainstalovat z webu `sourceware.org/cygwin`.

Důležité je nezapomenout zašktnout devel a debug v seznamu balíčků k instalaci.

B.2 Verzovací systém Git

Verzovací systém Git používám nejen v zaměstnání, ale i v domácích projektech a má své místo i v této práci. Může být důležité moci se v případě nutnosti vrátit k předchozím verzím projektu.

Git je standardní balíček snad všech distribucí. Instaluje je jako obvykle napsáním

```
sudo apt-get install git
```

S klíči se zachází standardním způsobem, měly by být uloženy v adresáři `~/.ssh`.

Vytvořil jsem speciální repozitář na svém testovacím serveru. Přihlašovat se jde výhradně pomocí klíče, který najdete na CD–příloze této práce. Projekt stačí už jen naklonovat příkazem,

```
git clone virtualbox@46.167.245.175:dp
```

přesněji, pomocí klíče na CD

```
ssh-agent sh -c 'ssh-add <cesta>/virtualbox_rsa;
git clone virtualbox@46.167.245.175:dp'
```

B.3 NetBeans IDE

NetBeans je dostupný jako klasický deb balíček, bohužel verze nabízená na repozitáři je zastaralá. Proto je lepší stáhnout si ze stránek `netbeans.org/downloads` aktuální. I zde je vhodné povýšit verzi viruálního stroje javy, vyzkoušel jsem open-jdk-8. Instalace je bez problémů.

B.4 Instalace openCV

Instalace OpenCV funguje přesně podle návodu na `opencv.org`.

Nainstalujeme nejdříve knihovny,

```
sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev
libavformat-dev libswscale-dev
```

a také,

```
sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev
 libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev
```

Samotná openCV je kvůli aktuálnosti stažena jako repozitář z git.

```
git clone https://github.com/opencv/opencv.git
```

Knihovnu je třeba přeložit a nainstalovat na správné místo, tak aby při použití mohla být nalezena.

```
cd ~/opencv
mkdir release
cd release
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local
make
sudo make install
```