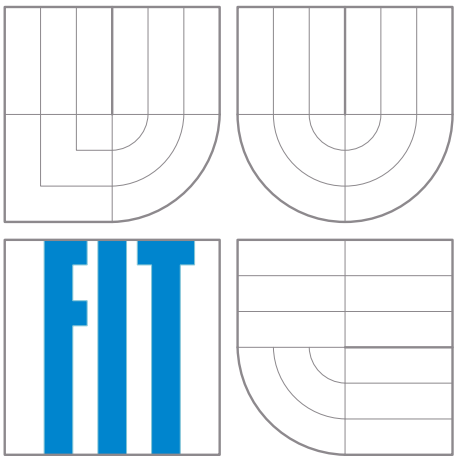


Projekt do předmětu GMU – Grafické a multimediální procesory

Komprese/dekomprese JPEG obrázků pomocí 3D akcelerační karty

13. prosince 2012

Řešitelé: Lucie Matušová (xmatus21@stud.fit.vutbr.cz)
Jan Wozniak (xwozni00@stud.fit.vutbr.cz)
Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

1 Zadání

Úkolem projektu bylo vybrat částí algoritmu při kompresi/dekompresi, které jsou vhodné pro paralelizaci, implementovat a optimalizovat dané části v OpenCL a nakonec porovant rychlost implementace s CPU. Jelikož implementace JPEG kodéru a dekodéru je velmi pracnou záležitostí[1], rozhodli jsme se výkonnosti jednotlivých částí pipeline implementovat a měřit zvlášť.

Dále jsou v textu popsány vybrané algoritmy a porovnání paralelních variant na GPU se seriovými, které vykonává CPU.

2 Použité technologie a zdroje

- OpenCL – pro paralelizaci algoritmů
- STL kontejnery – pro usnadnění práce s některými dynamickými typy, tvořen je vektor JPEG markerů
- Vim – editor
- GNU Make – nástroj pro buildování
- ostatní zdroje jsou uvedeny mezi citacemi

3 Teoretický rozbor

Schéma pipeline pro sekvenční režim JPEG komprese na obrázku 2 obsahuje barevně odlišené bloky, což jsou algoritmy, které jsme se rozhodli paralelizovat. Mezi jednotlivými bloky jsou uvedeny i datové typy, jaké jsou mezi výstupem jednoho bloku a vstupem druhého bloku očekávány.

Převod barevného modelu je prvním krokem zpracování rastrových dat. Provádíme bezeztrátovou transformaci z barvového prostoru RGB do YC_bC_r. Převod jednotlivých složek se provádí podle následujících vzorců:

$$Y = 0,299R + 0,587G + 0,114B \quad (1)$$

$$C_b = -0,1687R - 0,3313G + 0,5B + 128 \quad (2)$$

$$C_r = 0,5R - 0,4187G - 0,0813B + 128 \quad (3)$$

Diskrétní kosinová transformace slouží k převodu obrazu z prostorové domény do frekvenční domény. V normě JPEG je definováno 6 variant, z nichž nejčastěji používaná Baseline DCT, kterou jsme také v rámci projektu implementovali[2]. Po aplikaci kosinové transformace dostaneme dvě složky koeficientů – stejnosměrnou (AC) a střídavou (DC). Tyto složky se dále v algoritmu komprimují zvlášť. Vzorce 4, 5 a 6 jsou matematickým zápisem DCT, kde platí, že *N* je pro JPEG blok rovno 8.

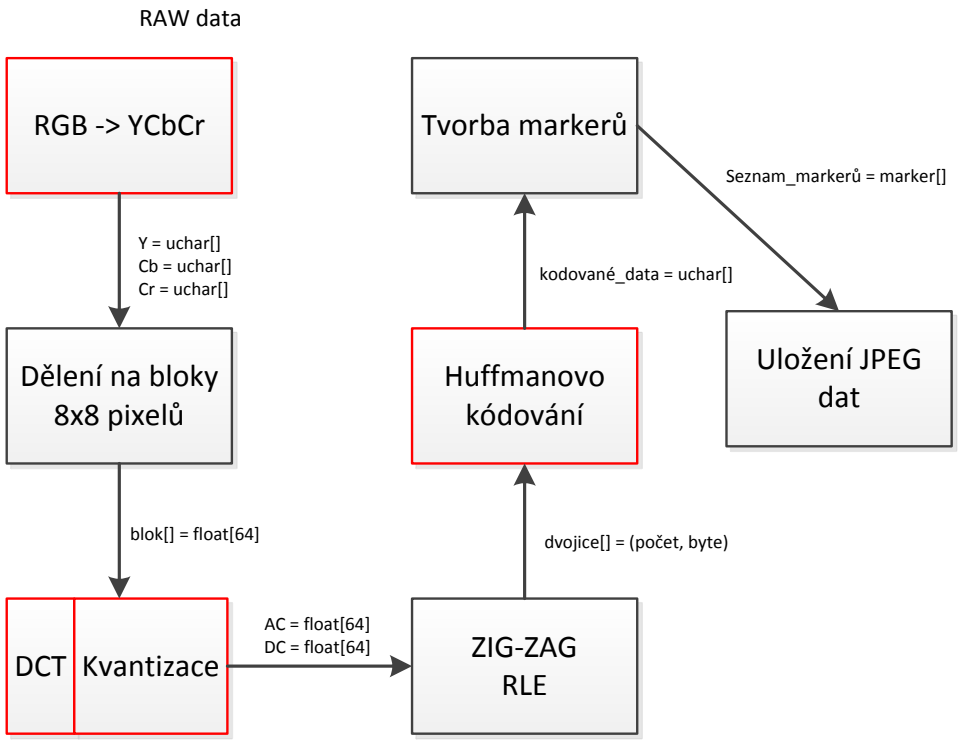
$$\lambda_x = \begin{cases} \frac{1}{\sqrt{2}}, & x = 0 \\ 1, & x \neq 0 \end{cases} \quad (4)$$

$$g_{k,j}[n,m] = \lambda_k \lambda_j \frac{2}{N} cos\left[\frac{k\pi}{N}\left(n + \frac{1}{2}\right)\right] cos\left[\frac{j\pi}{N}\left(m + \frac{1}{2}\right)\right] \quad (5)$$

$$c[k,j] = \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} f[n,m] g_{k,j}[n,m] \quad (6)$$

Kvantizace je implementována pomocí dvou tabulek, zvlášť pro jasovou složku a chrominescenční složky. Hodnota v bloku je nejprve vydělena příslušnou hodnotou v tabulce, zaokrouhlena a poté vynásobena stejnou hodnotou z tabulky. Pro kvantizaci je definováno několik vhodných tabulek[1].

Huffmanovo kódování kvantovaných složek probíhá pro stejnosměrné a střídavé koeficienty zvlášť. Před samotným kódováním se tyto složky upraví, aby došlo k redukci výsledného počtu bitů. Nad blokem DC koeficientů spočítáme diferenci, tj. odečteme hodnotu předchozího bloku. Jelikož DC složka odpovídá průměrnému jasu či barvě pixelů v daném bloku a obrázky většinou obsahují souvislé barevné plochy, vniknou takto nízké hodnoty[4]. U AC složek se provádí RLC kódování. Mnoho z nich má totiž nulovou hodnotu, je proto výhodné zapsat sekvence nul efektivně. Výstupem jsou bajty ve tvaru "RRRRSSSS". RRRR je počet nulových koeficientů předcházejicích nenulový koeficient. Pokud je tento počet větší než 15, zapíše se hodnota 0xF0. Pokud jsme na konci a zbývají již samé nuly, zapisuje se 0x00. SSSS značí kategorii hodnoty koeficientu. Huffmanovo kódování probíhá na základě čtyř tabulek - koeficienty DC a AC, které mohou být pro jasovou složku (Y) nebo barvové složky (C_b, C_r). U DC složky probíhá tak, že se 11-bitová hodnota diference zakóduje podle jedné z tabulek (Y nebo C_b, C_r) a za tento kód se přidá samotná hodnota diference ve dvojkovém doplňku. AC koeficienty projdou stejným procesem, akorát jsou pro ně kvůli většímu rozsahu použity rozsáhlejší tabulky.



Obrázek 1: Schéma JPEG encode pipeline.

4 Implementace

Celý projekt je rozdělen do několika modulů

- `cl_util` – funkce potřebné pro práci s OpenCL, jednotlivé kernely jsou v souboru `jpeg.cl`.
- `color_transform` – funkce pro načtení/uložení obrázku TGA, transformace barevného prostoru a podvzorkování barevných složek
- `dpcm_rle` – výpočet diference pro vstupní blok DC koeficientů a rle kódování AC koeficientů dle normy JPEG
- `huffman` – Huffmanovo kódování a kódovací tabulky
- `jpeg_util` – tvorba a čtení JPEG markerů a další pomocné funkce pro JPEG, které nemají samostaný modul (zig-zag, kvantizace, DCT ...).
- `main` – funkce `main` a časová měření algoritmů.

Diskrétní kosinová transformace je implementována pomocí funkcí

`void dct8x8(float* block, float* dct, int* table)` – sériově
`void dct8x8_gpu(float* block, float* dst, cl_mem* table)` – paralelně

kde prvním argumentem jsou data bloku vstupujícího do DCT, druhým argumentem pole pro výstupní AC a DC koeficienty DCT a třetím argumentem je kvantizační tabulka, neboť v rámci optimalizace jsme sjednotili kvantizaci s DCT.

Převod barevného modelu je implementován pomocí dvou funkcí:

`void rgb_to_ycbcr(pixmap *Y, pixmap *Cb, pixmap *Cr, pixmap *d)` – sériově
`void ycbcr_gpu(pixmap* data, unsigned char* dst)` – paralelně

`pixmap` je struktura obashující rozměry a rastrová data. Do sériového výpočtu vstupují tři pixmapy (Y, C_b, C_r) pro uložení výsledků a pixmapa s načteným obrázkem. V paralelní implementaci vstupuje pixmapa s obrázkem a druhým argumentem je pole, do kterého se ukládají všechny tři složky.

5 Dosažené výsledky

Výsledkem naší práce je srovnání rychlosti implementací na těchto strojích.

- Ubuntu – Core 2 Duo (T7500) 2.20GHz, NVIDIA GeForce 8600M GT
- Debian – Core i5 (2500K) 3.3 GHz, NVIDIA GeForce 8800 GTX
- OS X – Core i5 (I5-3317U) 1.7 GHz, HD Graphics 4000

Hodnoty měření jsou uvedeny v milisekundách, porovnávány jsou paralelní verze a sériové verze algoritmů. Konverze barevných prostorů byla testována na obrázku 4752x3168 pixelů, DCT je měřena pro jediný blok.

Algoritmus	Ubuntu		Debian		OS X	
	serial [ms]	paralel [ms]	serial [ms]	paralel [ms]	serial [ms]	paralel [ms]
`RGB to YCbCr`	4.4789	10.0818	1.14608	0.874043	1.24693	1.94597
`YCbCr to RGB`	1	2	1.63078	0.766993	1.86396	1.86396
`DCT`	0.527859	1.49798	0.320911	0.144005	0.410795	0.264168
`Inv_DCT`	0.527143	1.14584	0.276089	0.113011	0.396967	0.203848

Tabulka 1: Tabulka srovnání doby výpočtu jednotlivých algoritmů.

Obrázek 2: Srovnání před kompresí a po kompresi.

6 Rozdělení práce

- Lucie Matušová – převod barevných modelů, extrakce raw dat z TGA formátu, Huffmanovo kódování
- Jan Wozniak – DCT, kvantizace, JPEG markery a jeho bloková struktura

7 Co bylo nejpracnější

Formát JPEG není triviální. Norma [1] je složitě napsaná a ostatní zdroje poskytují většinou jen kusé informace. Implementace celé pipeline se nakonec ukázala v daném čase jako nereálná. Od začátku jsme se snažili implementovat všechny části podle doporučení z normy tak, aby bylo možné výsledný soubor dekomprimovat běžným prohlížečem obrázků (např. úpravy koeficientů vstupujících do Huffmanova kódování). Důsledkem byl nedostatek času na paralelizaci Huffmanova kódování.

Komplikace také způsoboval fakt, že jeden z členů týmu nemá na svém počítači podporu OpenCL.

8 Zkušenosti získané řešením projektu

Zdokonalili jsme se v programování v OpenCl. Osvojili jsme si práci s jeho paměťovým modelem. Osvěžili jsme si znalosti kódování a důkladně prostudovali JPEG/JFIF. Také jsme si ověřili, že

návrh a časový odhad je na začátku klíčový. Možná by pomohlo na začátku své myšlenky více konzultovat se cvičícími.

9 Autoevaluace

- **Technický návrh: 60%** – původní plán si dával za cíl vytvořit úplný kodér JPEG formátu, což není nikterak triviální úloha. Rozhodně přínosnější, než pochopit vnitřní strukturu JPEG souboru uloženého na disku by bylo vybrat jen určité algoritmy a důkladněji se zabývat optimalizací.
- **Programování: 85%** – absence rozáhlejších komentářů v některých oblastech může zhoršit čitelnost, ale to je jen u triviálních metod/funkcí. Celkově je projekt rozdělen logicky do samostantých modulů.
- **Vzhled vytvořeného řešení: n/a** – mělo se jednat o konzolovou aplikaci.
- **Využití zdrojů: 90%** – přesný popis JPEG formátu poskytuje nejlépe takřka 190 stranová specifikace. Vzhledem k náročnosti tohoto textu bylo ale pro nás nezbytné zpočátku hledat srozumitelnější informace i jinde.
- **Hospodaření s časem: 40%** – viz. technický návrh.
- **Spolupráce v týmu: 100%** – pro verzování kódu jsme použili program Git, práce byla produktivní i přes nedotažení původního cíle dokonce.
- **Celkový dojem 80%** – tím, že je JPEG formát jedním z nejpoužívanějších digitálních obrazových formátů, je znalost jeho struktury určitě užitečná pro každého informatika. Rozhodně jsme se naučili ocenit práci, kterou udělali jiní programátoři a dříve triviální otevření ”džejpegu”respektujeme daleko více.

Literatura

[1] *Recommendation T.81*. [cit. 2012-12-10] <http://www.w3.org/Graphics/JPEG/itu-t81.pdf>

[2] D. Bařina. *Diskrétní kosinová transformace – prezentace ke cvičení*. [cit. 2008-12-08]. <http://www.fit.vutbr.cz/study/course-l.php.cs?id=8766>

[3] *Wikipedia, the free encyclopedia*. [cit. 2012-12-10] <http://en.wikipedia.org/>

[4] Pavel Tišnovský. *Ztrátová komprese obrazových dat pomocí JPEG*. ROOT.CZ, 2006, 14.12.[cit. 2012-12-10] <http://www.root.cz/clanky/ztratova-komprese-obrazovych-dat-pomoci-jpeg/