

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta Informačních Technologií

Pokročilé asemblery

2016/2017

Semestrální projekt IPA

Zadání č. 1 – Akcelerace algoritmu pro zpracování obrazu pomocí vektorových
NEON instrukcí procesoru ARM

1 Úvod

Úkolem projektu bylo implementovat zadaný algoritmus v assembleru.

Zadání spočívalo v převedení obrazu z RGB do HSB, modifikování některé ze složek a následné převedení zpět do RGB implementované v assembleru procesorů ARM.

2 Problematika

Při operacích vyžadujících jiný barevný systém, než v jakém je obraz uložen, je potřeba provést operaci konverze nad každým pixellem. Při použití sekvenčního algoritmu je náročnost operace přímo úměrná počtu pixelů, které je potřeba zpracovat. Jednou z možností akcelerace algoritmu je použití grafického čipu, který podporuje paralelní zpracování více pixelů současně. Grafický akcelérátor ale není součástí definice architektury procesorů **ARM** a na jeho přítomnost nelze vždy spoléhat. Další možností akcelerace je použití **NEON** koprocessoru, kterým disponují všechny **Cortex-A** procesory založené na architektuře **ARM** od verze **ARMv5**.

Formulace problému: Je potřeba vektorově akcelarovat převod RGB komponent obrazu do složek HSB.

2.1 NEON koprocessor

NEON koprocessor se řadí mezi **SIMD** (*Single Instruction, Multiple Data*) procesory pro obecné použití, kdy jedinou instrukcí dokáže zpracovat 64 nebo 128 bitů současně. Tyto vektory jsou naplněny bity, které jsou interpretovány jako elementy po 8, 16, 32, 64 nebo 128 bitech. Odpovídající elementy více vektorů se nazývají linie. **NEON** vždy provádí stejnou operaci nad každou linií vektoru.

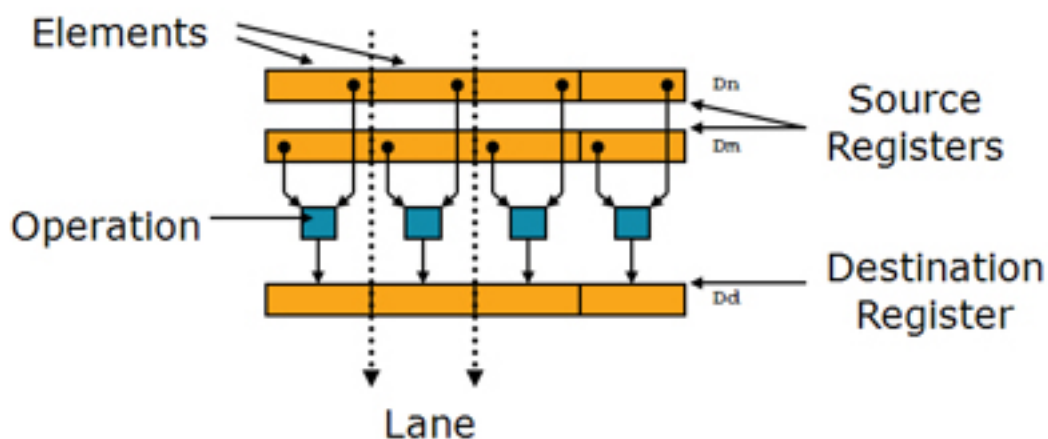


Fig. 1: Schéma NEON architektury

V závislosti na minimálním počtu bitů potřebných k reprezentaci čísla lze tak urychlit výpočet až 16-krát pro 8-bitové operace, které jsou mnohdy dostačující při zpracovávání obrazu s barevnou hloubkou 8 bitů.

2.2 VFP

Vector Floating Point Unit (VFP) je další z koprocetorů, kterým ARM disponuje. Jeho název je však zavádějící, neboť poskytuje pouze pseudo-vektorové výpočty. Instrukce sice pracují nad vektorem dat, výpočet je ale prováděn pro každý skalár vektoru samostatně a neposkytuje tedy významné urychlení.

3 Algoritmy

NEON dokáže při načtení rozřadit jednotlivé pixely z prolínaných dat jedinou instrukcí. Tmto je možno velmi efektivně načíst jednotlivé barevné komponenty do tří samostatných vektorů.

Při akceleraci je třeba provést tři kroky:

1. Konverze obrazu do HSB barevného systému
2. Úprava komponenty v HSB
3. Zpětná konverze do RGB

3.1 Vektorizace algoritmu koprocetorem NEON

Navzdory očekávání nepodporuje instrukční sada architektury **NEON** dělení. Dělení obsažené v převáděcích algoritmech je třeba provádět samostatně v procesoru **ARM** pseudo instrukcemi.

Absence dělení na úrovni hardware značně zpomaluje celý algoritmus. Při každém průchodu je třeba načíst dělenec i dělitel z registru **NEONu** do registru **ARMu**. Zatímco přenos ve směru ARM-NEON je rychlý, přenos NEON-ARM může trvat na architektuře **Cortex-A8** až 20 taktů. U architektury **Cortex-A9** to však není více než 4 takty.

3.2 Konverze do HSB

Pro konverzi byl použit následující algoritmus, popis je udeven v jazyce Modula-2:

```

PROCEDURE RGB2HSB( r, g, b : CARDINAL; VAR H, S, B : INTEGER );
VAR
  Ma, Mi : INTEGER;
  Delta : INTEGER;
BEGIN
  IF r > g THEN Ma := r; ELSE Ma := g; END;
  IF b > Ma THEN Ma := b; END;
  IF r < g THEN Mi := r; ELSE Mi := g; END;
  IF b < Mi THEN Mi := b; END;
  B := Ma;
  Delta := Ma - Mi;
  IF Ma = 0 THEN
    S := 0;
  ELSE
    S := INTEGER(CARDINAL(Delta) * 256 DIV CARDINAL(Ma));
    IF S > 255 THEN S := 255; END;
  END;
  IF S = 0 THEN // gray level
    H := 0;
  ELSE
    IF r = Ma THEN // r dominant
      H := (g - b) * 60 DIV Delta;
    ELSIF g = Ma THEN // g dominant
      H := 120 + (b - r) * 60 DIV Delta;
    ELSE // b dominant
      H := 240 + (r - g) * 60 DIV Delta;
    END;
    IF H < 0 THEN
      H := H + 360;
    END;
  END;
END RGB2HSB;

```

Kód procedury převodu RGB do HSB

Tento kód je vektorizovatelný, a lze jej převést do instrukcí koprocessoru **NEON**. Za použití vektorových instrukcí lze zpracovávat až 8 pixelů naráz nahráním každé z 8-bitových barevných složek do tří odpovídajících 16-bitových elementů 128-bitových vektorů koprocessoru **NEON**.

3.3 Úprava komponenty v HSB

Úprava komponenty v HSB je jednoduchá a snadno vektorizovatelná. Lze ji provést jako součást prováděcí procedury před uložen pro další urychlení kódu.

3.4 Převod zpět do RGB

Pro konverzi byl použit následující algoritmus, popis je udeven v jazyce Modula-2:

```
PROCEDURE HSB2RGB( H, S, B : CARDINAL; VAR r, g, b : CARDINAL );
VAR
  i : CARDINAL;
  f, p, q, t : CARDINAL;
BEGIN
  i := H DIV 60;
  f := H MOD 60;
  p := B * (255 - S) DIV 256;
  q := B * (255 - S * f DIV 60) DIV 256;
  t := B * (255 - S * (60 - f) DIV 60) DIV 256;
  CASE i OF
    | 0: r := B; g := t; b := p;
    | 1: r := q; g := B; b := p;
    | 2: r := p; g := B; b := t;
    | 3: r := p; g := q; b := B;
    | 4: r := t; g := p; b := B;
    | 5: r := B; g := p; b := q;
  ELSE;
  END;
END HSB2RGB;
```

Kód procedury převodu HSB do RGB

Z dříve zmíněných důvodů nemožnosti dělit v korocesoru **NEON** a výraznému větvení nelze tento algoritmus účinně vektorizovat. Algoritmus byl přepsán do jazyka **C** ve kterém je psán celý projekt a bylo ponecháno na překladači provést jakoukoli možnou dodatečnou optimalizaci.

4 Vývojová platforma

Pro vývojové a testovací účely byl použit počítač **Raspberry Pi model 3B**. Jedná se jedno-deskový počítač velikosti kreditní karty vyvíjený ve Velké Británii organizací Raspberry Pi Foundation. Projekt má za cíl rozšiřovat výuku informatiky a informačních technologií ve školách.

Pro přítomnost **SOC** (system on a chip) firmy Broadcom s integrovaným procesorem **ARM Cortex-A53** je ideální platformou na vývoj a testování **ARM** aplikací. Jako plnohodnotný počítač lze přímo na něm spustit textový editor, překladač i debugger. Tím odpadá nutnost použití testovacích a ladících zařízení.

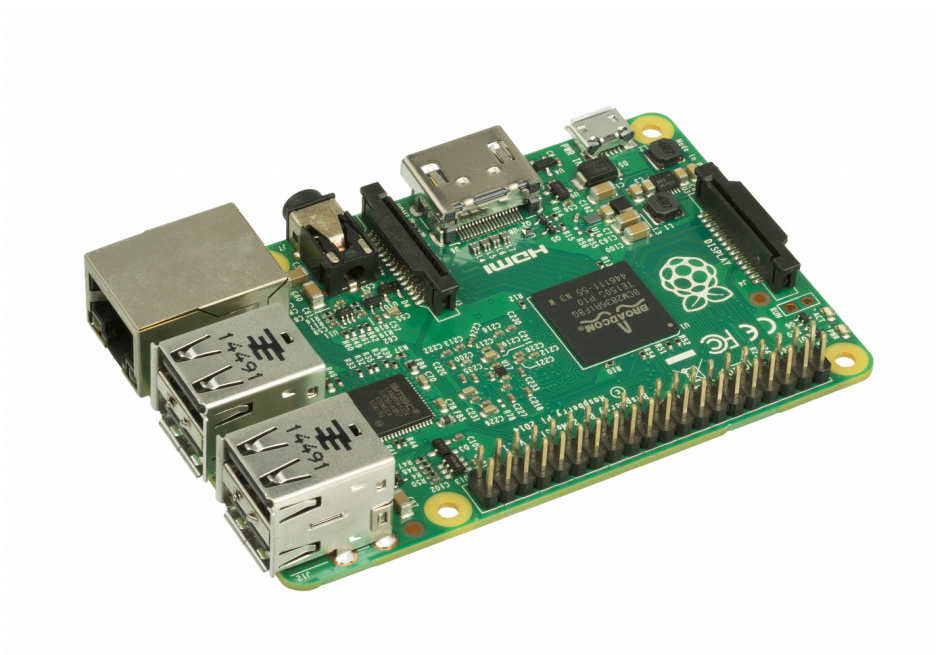


Fig 2: Raspberry Pi 3 model B

4.1 Specifikace

```
Processor Chipset: BCM 2837 64bit ARMv8 Cortex A53 Quad Core
Processor Speed: 1.2Ghz per core
Max Power Draw: 2.5A
model name: ARMv7 Processor rev 4 (v7l)
Features: half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae
evtstrm crc32
CPU architecture: 7
CPU(s): 4
On-line CPU(s) list: 0-3
Thread(s) per core: 1
Core(s) per socket: 4
Socket(s): 1
CPU max MHz: 1200.0000
CPU min MHz: 600.0000
```

Zdroj: <https://www.element14.com/community/community/raspberry-pi/blog/2016/02/29/the-most-comprehensive-raspberry-pi-comparison-benchmark-ever>

5 Použité nástroje

Projekt je celý napsán v jazyce **C** doplněném o in-line assembler překladače **GCC**. Pro úpravy kódu byl použit editor **Geany**. Pro kompilaci poté **GNU C Compiler**. Pro jednoduchou operaci s kódem byl využit GNU program **Make**.

Tvorba probíhala přímo na linuxové distribuci **Raspbian**, derivátu Debianu, od organizace Raspberry Pi Foundation.

6 Závěr

Pro celkovou a časovou náročnost projektu bylo odevzdání projektu opožděno oproti původnímu termínu. Akcelerace naráží na velkou časovou náročnost přesunu mezi registry NEONu a ARMu. Přesto byl projekt velmi zajímavý a věřím, že pro mne i přínosný.

7 Zdroje

<https://www.element14.com/community/community/raspberry-pi/blog/2016/02/29/the-most-comprehensive-raspberry-pi-comparison-benchmark-ever>

https://en.wikipedia.org/wiki/ARM_architecture

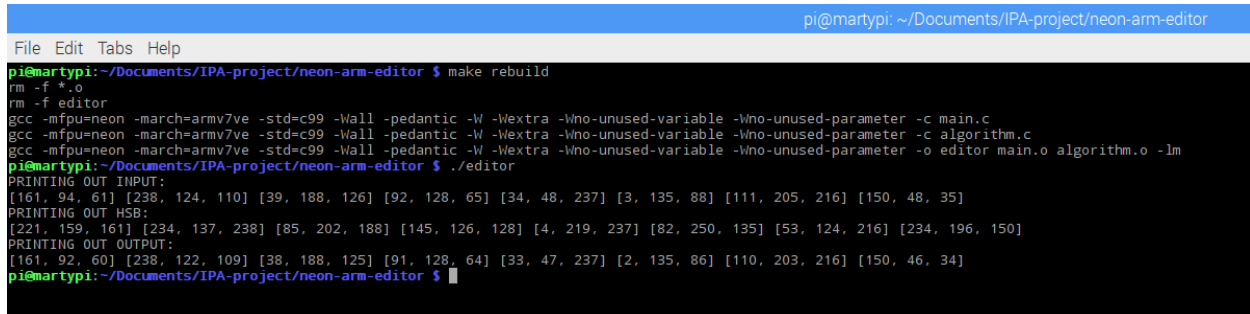
https://en.wikipedia.org/wiki/Raspberry_Pi

<https://www.arm.com/products/processors/technologies/neon.php>

Fig. 1 převzata z <https://www.arm.com/products/processors/technologies/neon.php>
Fig. 2 převzata z Wikimedia Commons

8 Appendix

8.1 Ukázka práce programu nad 8 pixely:



```
pi@martypi: ~/Documents/IPA-project/neon-arm-editor
File Edit Tabs Help
pi@martypi:~/Documents/IPA-project/neon-arm-editor $ make rebuild
rm -f *.o
rm -f editor
gcc -mfpu=neon -march=armv7ve -std=c99 -Wall -pedantic -W -Wextra -Wno-unused-variable -Wno-unused-parameter -c main.c
gcc -mfpu=neon -march=armv7ve -std=c99 -Wall -pedantic -W -Wextra -Wno-unused-variable -Wno-unused-parameter -c algorithm.c
gcc -mfpu=neon -march=armv7ve -std=c99 -Wall -pedantic -W -Wextra -Wno-unused-variable -Wno-unused-parameter -o editor main.o algorithm.o -lm
pi@martypi:~/Documents/IPA-project/neon-arm-editor $ ./editor
PRINTING OUT INPUT:
[161, 94, 61] [238, 124, 110] [39, 188, 126] [92, 128, 65] [34, 48, 237] [3, 135, 88] [111, 205, 216] [150, 48, 35]
PRINTING OUT HSB:
[221, 159, 161] [234, 137, 238] [85, 202, 188] [145, 126, 128] [4, 219, 237] [82, 250, 135] [53, 124, 216] [234, 196, 150]
PRINTING OUT OUTPUT:
[161, 92, 60] [238, 122, 109] [38, 188, 125] [91, 128, 64] [33, 47, 237] [2, 135, 86] [110, 203, 216] [150, 46, 34]
pi@martypi:~/Documents/IPA-project/neon-arm-editor $
```

Fig. 3: Ukázka zpracování 8 pixelů algoritmem