Semestrální projekt č.2: Cache

Semestrální projekt č.2: Cache

Úvod

Představte si situaci, že jste zaměstnancem počítačové firmy a jste členem týmu, který vyvíjí editor obrázků (Photoshop, Gimp apod.). Aby editor našel své zákazníky musí být dostatečně rychlý a přehledný - jinak neuspěje v konkurenčním boji. Vám byl přidělen úkol implementovat operaci konvoluce (http://cs.wikipedia.org/wiki/Konvoluce) do jádra programu. Konvoluce je jednou z nejdůležitějších operací při zpracování digitálního obrazu. Je to ve své podstatě algoritmus, který počítá výsledné pixely jako vážené součty pixelů v jejich okolí. Výpočet se provádí pro každou barevnou složku zvlášť. Vašim dalším úkolem je výpočet histogramu.

Jak funguje konvoluce se můžete podívat i zde: http://setosa.io/ev/image-kernels/ (http://setosa.io/ev/image-kernels/)

Úkol

Vašim domácím úkolem bude implementovat pouze níže uvedenou konvoluční masku pro ostření obrazu, přičemž je žádoucí optimalizovat práci s pamětí a to jakýmkoliv způsobem. Zaměřte se na efektivní práci s cache při použití právě níže uvedené masky.

```
 0  -1 0
-1 5  -1
 0  -1 0
```

Pixely na okraji obrázku pouze překopírujte z původního obrázku (=situace kdy konvoluční jádro přesahuje původní obrázek).

Dále pak spočítejte a zapište do souboru histogram zaostřeného obrázku ve stupních šedi. Pro konverzi z RGB do grayscale použijte model pro výpočet jasu:
Y = round(0.2126*R + 0.7152*G + 0.0722*B)

Histogram spočítejte pro zleva uzavřený, zprava otevřený interval < i*L/N; (i+1)*L/N) vyjma posledního intervalu kdy je i zprava uzavřený; pro i=0…N-1, kde L=255 a N=5 je počet intervalů; Jinými slovy, interval <0; 255> rozdělte na tyto části:

Subinterval:	od 0 do 50	od 51 do 101	od 102 do 152	od 153 do 203	od 204 do 255
Četnost:					

Vstupní data

Vstupní obrázek bude v binárně kódovaném formátu portable pixmap format (PPM) (http://en.wikipedia.org/wiki/Netpbm_format) . Obrázek uložený v tomto formátu má vždy na začátku souboru uvedeno P6, za kterým následují údaje o šířce a výšce obrázku. Dále konstanta 255 (maximální hodnota intenzity dané složky pixelu) a pak samotná data - jednotlivé RGB složky pro každý pixel. Každá složka pixelu (RGB) zabírá právě jeden Byte.

vit-small.zip (../../media/tutorials/08/vit-small.zip) vit-normal.zip (../../media/tutorials/08/vit-normal.zip)

Jméno vstupního souboru bude předáno z příkazové řádky – viz parametry funkce main(int argc, char * *argv).

Výstup programu

Výstupem Vašeho programu bude zaostřený obrázek v souboru **output.ppm** a histogram (četnosti jasu oddělené mezerou) v souboru **output.txt**.

Kritéria hodnocení domácího úkolu

Program, který odevzdáte bude hodnocen z pohledu celkového počtu dotazů do datové L1 cache, do instrukční L1 cache, společné L2 cache (intrukce+data), a především počtu missů na úrovni L1 (datová i instrukční cache) a úrovni L2. Předpokládejte oddělenou instrukční a datovou L1 cache, každá o velikosti 32 KB, 8-cestně asociativní, velikost bloku 64B. Pro L2 cache předpokládejte velikost 1 MB, 16-cetně asociativní, velikost bloku 64B. Algoritmus nahrazování je LRU. L2 cache je inkluzivní (obsahy obou L1 caches jsou i v L2 cache). Váš program by měl být schopen parametry cache detekovat (a přizpůsobovat se jim), nicméně pro účely domácího úkolu je plně postačující optimalizovat Váš program pouze pro výše uváděné hodnoty.

Hodnocení:

- 5 bodů za domácí úkol získá každý kdo odevzdá funkční program (Body_funkcnost)
- další body budou přiděleny dle efektivity používání cache a to pouze pokud je program funkční (Body_efektivita)
- počet získaných bodů za úkol: Body = Body_funkcnost + Body_efektivita

Efektivita používáni cache bude hodnocena na základě výkonnosti Vašeho programu.

Cost = AMAT_i*I_refs + AMAT_d*D_refs,

kde * AMAT_i je průměrný čas přístupu při dotazování se do instrukční cache: AMAT_i = HT_L1i + MR_L1i*(HT_L2 + MR_L2i*HT_RAM) * I_refs je počet referencí do instrukční cache * AMAT_d = HT_L1d + MR_L1d*(HT_L2 + MR_L2d*HT_RAM) * D_refs je počet referencí do datové cache * Předpokládána frekvence CPU: 3.2 GHz * Latence L1: 4 cykly ⇒ HT_L1i = HT_L1d = 1.25 ns * Latence L2: 65 cyklů ⇒ HT_L2 = 20.3 ns * Latence RAM: 140 cyklů + 100 ns. ⇒ HT_RAM = 144 ns

Následně se určí počet bodů za efektivitu dle vztahu:

Body_efektivita = 10*(Cost_Max-Cost)/(Cost_Max - Cost_Min),

kde

Cost_Min = 2,0

Cost_Max = 9,0.

Pokud by měl získat někdo záporný počet bodů, bude mu za efektivitu přiděleno 0 bodů. Pokud naopak někdo přesáhne 10 bodů, bude mu uděleno právě 10 bodů. Platí, že počet bodů se zaokrouhlí směrem nahoru na násobek 0,25. Váš program bude testován nad několika různými obrázky (každý o jiné velikosti). Hodnoty Cost_Min = 2.0 a Cost_Max = 9.0 platí pro obrázky velikosti vit_normal.ppm, 64-bit Ubuntu, gcc 4.8.4.

Celkem tedy student může získat 15 bodů za domácí úkol.

Odevzdání domácího úkolu

Program s algoritmem musí být vytvořený v jazyce C bez použití externích knihoven pro práci s obrázky. Formát obrázku je zvolený tak, aby i jeho načítání bylo možné snadno implementovat na několika řádkách kódu. Program může být optimalizovaný pro překlad na 32 a 64-bitovou architekturu Intel x86. Kompilace a test bude probíhat v podobné prostředí pod OS GNU/Linux, jako je k dispozici v laboratoři na cvičeních.

Odevzdávání domácího úkolu probíhá přes: http://biaps.fit.cvut.cz/second_semestral_project/index.php

Odevzdávejte soubor `Prijmeni_Jmeno_main.c` nebo `Prijmeni_Jmeno_main.cpp` zabalený v zip archivu.

Překlad C:

```
g++ -O1 -Wall -Werror -lm -o main main.c
```

Všechny odevzdané úkoly budou kontrolovány na plagiátorství (všichni zúčastnění budou postiženi stejně, bez rozdílu).

Upozorňení: Soubor output.txt musí obsahovat přesně 5 dekadických čísel. Oddělovačem je mezera. Za posledním číslem nesmí být žádný další znak (ani odřádkování).

Odevzdávací systém bude uzavřen na konci zápočtového týdne (poslední týden před začátkem zkouškého období).

Nápověda

Cena (Cost), ze které vychází hodnocení Vašeho úkolu se velmi dobře odráží v rychlosti běhu Vašeho programu (čím rychlejší program, tím nižší cena). Měřením času tedy můžete velmi snadno zjistit jak je Váš přístup efektivní - jak z pohledu práce s pamětí tak z pohledu algoritmu. Čas lze změrit níže uvedeným způsobem:

```
#include <unistd.h>
#include <time.h>
...
struct timespec start, stop;
```

```
clock_gettime( CLOCK_REALTIME, &start);
...
clock_gettime( CLOCK_REALTIME, &stop);
double accum = ( stop.tv_sec - start.tv_sec )*1000.0 + ( stop.tv_nsec - start.tv_nsec )/ 1000000.0;
printf( "Time: %.6lf ms\n", accum );
```

Poznámka: Při kompilaci nezapomeňte -lrt. Příklad: g + + main.cpp -g -lrt

Vyhodnotit jak program pracuje s cahce lze pomocí nástroje cachegrind. Nezapomeňte nastavit parametry cache (–I1=… –D1=… –LL=…), jinak se použijí defaultní hodnoty (nebo hodnoty Vašeho procesoru).

```
valgrind --tool=cachegrind --I1=32768,8,64 --D1=32768,8,64 --LL=1048576,16,64 ./a.out
```

Tím získáte základní představu o celkovém chování Vašeho programu. Pokud chcete program analyzovat detailněji, můžete využít nástroje cg_annotate. Příklad:

```
cg_annotate <filename> ~/domaci_ukol/main.cpp
```

kde filename je jméno souboru vygenerovaného pomocí cachegrind. Cestu k Vašemu zdrojovému souboru (main.cpp) uvádějte absolutní.

Testovací obrázek 10 x 8 (../../media/tutorials/08/test-10x8.zip)

md5 součet zaostřeného obrázku vit_small.ppm je 32554ccd9b09af5b660a17b05350959b. V hlavičce výstupního obrázku jsou jednotlivé položky odděleny odřádkováním (\n) – hlavička výstupního a vstupního souboru je totožná. První tři položky histogramu jsou: 24432, 16307, 15192. Zbylé dvě nejsou zde uvedeny.