Compresie de imagini folosind Block Truncation Coding pe Cell BroadBand Engine

Vlad Spoiala Adi Muraru

April 27, 2014

1 Introducere

Block Truncation Coding este o metoda de comprimare cu pierderi (lossy) de imagini alb negru. Ideea de baza a metodei este impartirea imaginii de intrare in blocuri disjuncte de dimensiune N x N si transformarea acestor blocuri intr-o masca de biti (bitplane) de dimensiune N x N (fiecare pixel din bloc corespunde unui bit din masca de biti, deci masca va contine N x N biti) si o pereche de valori dependente de media aritmetica si deviatia standard a pixelilor din bloc. Dimensiunea blocului (N) determina si rata de comprimare a metodei.

Odata impartita imaginea in blocuri, vom calcula media aritmetica ($\overline{x}$) si deviatia standard (σ) pentru pixelii din bloc.

Apoi vom construi masca de biti asociata blocului astfel: daca valoarea pixelului este strict mai mare decat media aritmetica atunci valoarea bitului din masca de biti va fi 1, altfel valoarea bitului va fi 0. Altfel spus, daca y este masca noastra de biti si x reprezinta blocul din imaginea din intrare, atunci valoarea bitului corespunzator liniei i si coloanei j din bloc va fi: $y(i,j) = \begin{cases} 1, & \text{daca } x(i,j) > \overline{x} \\ 0, & \text{daca } x(i,j) \leq \overline{x} \end{cases}$

Dupa ce am calculat masca de biti vom determina numarul de pixeli din bloc (m = N * N) si numarul de pixeli din bloc strict mai mari decat media aritmetica (q). Pentru fiecare bloc vom stoca, pe langa masca de biti, 2 valori a si b calculate astfel: $a = \overline{x} - \sigma * \sqrt{\frac{q}{m-q}}$ si $b = \overline{x} + \sigma * \sqrt{\frac{m-q}{q}}$.

Daca presupunem ca un pixel din imaginea originala era stocat folosind un octet atunci dimensiunea unui bloc din imaginea originala va fi de N * N octeti. Dimensiunea unui bloc din imaginea comprimata va fi data 2 + N * N / 8 (2 octeti pentru a si b, restul pentru masca de biti). Pentru N = 4, un bloc din imaginea de intrare va ocupa 16 octeti, in timp ce un bloc din imaginea comprimata va ocupa 4 octeti.

Decomprimarea (reconstructia imaginii originale din imaginea comprimata) se va face, bloc cu bloc, astfel:

$x(i,j) = \begin{cases} a, & \text{daca } y(i,j) = 0 \\ b, & \text{daca } y(i,j) = 1 \end{cases}$

Pentru mai multe informatii puteti consulta [1].

[1] `http://en.wikipedia.org/wiki/Block_Truncation_Coding`

2 Exemplu

Sa presupunem ca avem urmatorul bloc de dimensiune 4 x 4 din imaginea de intrare:

197	201	201	194
197	195	198	195
204	205	201	194
200	206	198	205

Figure 1: Bloc de dimensiune 4 x 4 din imaginea de intrare

Pentru media aritmetica si deviatia standard vom obtine: $\overline{x} = 199.43$ si $\sigma = 4.06$. Masca de biti va fi:

0	1	1	0
0	0	0	0
1	1	1	0
1	1	0	1

Figure 2: Masca de biti pentru blocul de dimensiune 4 x 4

Celelalate valori:

- $m = 16$
- $q = 8$
- $a = int(199.43 - 4.06 * \sqrt{\frac{8}{16-8}}) = 195$
- $b = int(199.43 + 4.06 * \sqrt{\frac{16-8}{8}}) = 204$

Blocul reconstruit:

195	204	204	195
195	195	195	195
204	204	204	195
204	204	195	204

Figure 3: Bloc de dimensiune 4 x 4 din imaginea reconstruita

3 Enunt

Se doreste implementarea comprimarii de tip Block Truncation Coding pe Cell Broad-Band Engine folosind o dimensiune a blocului de **8 x 8 (N = 8)**, precum si reconstructia imaginii originale din imaginea comprimata.

3.1 Input / Output

Imaginile de intrare si imaginile reconstruite vor fi in format PGM P5 (imagini alb-negru, pe 8 biti, versiunea binara de stocare a datelor). Pentru mai multe date legate de formatul PGM puteti consulta [2] si [3].

Imaginea comprimata (format BTC) va avea urmatorul format:

- 1 int reprezentand latimea (width) imaginii
- 1 int reprezentand inaltimea (height) imaginii
- informatia pentru cele $width*height/(N*N)$ blocuri, fiecare bloc fiind stocat astfel:
  - 1 octet reprezentand valoarea lui a
  - 1 octet reprezentand valoarea lui b
  - 8 octeti reprezentand masca de biti

3.2 Implementarea seriala

Ca punct de plecare pentru implementarea Block Truncation Coding pe Cell BroadBand Engine v-a fost pus la dispozitie un cod serial ce comprima o imagine PGM P5 intr-o imagine in format BTC si reconstruieste imaginea PGM dintr-o imagine comprimata.

Nu este obligatoriu sa plecati de la codul serial pentru realizarea implementarii paralele; il puteti folosi doar ca sursa de inspiratie sau va puteti construi codul de la zero.

Codul serial este structurat astfel:

- *pgm.c* - contine cod pentru citirea si scrierea de fisiere in format PGM P5; de interes sunt functiile *read_pgm* si *write_pgm*
- *btc.c* - contine cod pentru citirea si scrierea de fisiere in format BTC; de interes sunt functiile *read_btc* si *write_btc*
- *utils.c* - contine o serie de functii helper utilizate in operatiile de citire si scriere
- *btc.h* - definitii pentru structurile folosite si antete pentru functii
- *main.c* - contine functiile pentru comprimare si decomprimare, precum si main-ul
- *compare.c* - utilitar folosit pentru compararea de imagini in format BTC si format PGM P5

Programul serial primeste ca parametru calea catre un fisier de tip PGM P5 (*in.pgm*) si produce un fisier comprimat obtinut dupa aplicarea Block Truncation Coding (*out.btc*) si un fisier de tip PGM P5 obtinut dupa decomprimarea fisierului comprimat (*out.pgm*). Programul afiseaza timpul necesar comprimarii si decomprimarii, precum si timpul total (cu tot cu operatiile de citire / scriere).

Exemplu de rulare a codului serial:

```
./btc in.pgm out.btc out.pgm
```

[2] `http://netpbm.sourceforge.net/doc/pgm.html`
[3] `http://en.wikipedia.org/wiki/Netpbm_format`

3.3 Implementarea paralela

Se doreste utilizarea arhitecturii Cell BroadBand Engine pentru paralelizarea operatiilor de comprimare si decomprimare.

Partile intensiv computationale ale programului vor fi realizate de SPE-uri, in timp ce PPE-ul se va ocupa de agregarea datelor si coordonarea SPE-urilor.

In urma compilarii va trebui sa rezulte un binar cu numele **tema3**.

Rularea se va realiza astfel:

```
./tema3 mod num_spus in.pgm out.btc out.pgm
```

unde :

- mod poate fi 0 pentru utilizare normala sau 1 pentru varianta cu double buffering
- num_spus poate fi 1, 2, 4 sau 8
- in.pgm este imaginea de intrare
- out.btc este imaginea de iesire comprimata
- out.pgm este imaginea PGM reconstruita din imaginea comprimata

3.4 Testare

Fisierele input se gasesc pe *fep.grid.pub.ro* aici: */export/asc/btc_input*. La corectare se va testa doar cu aceste fisiere. Output-ul rularii seriale (ce poate fi folosit pentru comparatie) se gaseste aici: */export/asc/btc_output*

Pentru a compara 2 fisiere de output puteti utiliza utilitarul compare (prezent in arhiva cu codul serial) astfel:

```
./compare btc file1.btc file2.btc
./compare pgm file1.pgm file2.pgm
```

In faza de dezvoltare a codului recomandam testarea cu imagini de dimensiuni mai mici pentru a nu aglomera cozile. Pentru obtinerea de imagini mai mici puteti utiliza Gimp (sau alte tool-uri similare) pentru taierea (crop) imaginilor de intrare.

3.5 Mod de rulare

Pentru rulare veti folosi coada ibm-cell-qs22.q conform cu tutorialul prezentat pe wiki: `https://cs.curs.pub.ro/wiki/asc/asc:cellcookbook:ncittutorial`

Pe fiecare din cele 3 blade-uri sunt mapate 12 slot-uri (in total sunt 36 de slot-uri). Cum un blade are 2 PPU-uri si 16 SPU-uri in caz de utilizare maxima pe un blade vor fi 96 thread-uri SPU care se vor "bate" pe 16 SPU-uri. Asta inseamna ca in cazul in care pe un blade sunt ocupate mai mult de 2 slot-uri e posibil ca performanta programului vostru sa fie influentata de alte job-uri ce ruleaza pe acel blade la acel moment de timp. Pentru a obtine rezultate relevante pentru analiza performantei din README se poate rezerva un blade in intregime

Moduri de submitere:

- Rezerva un singur slot:

```
qsub -cwd -q ibm-cell-qs22.q -pe openmpi 1 run_all.sh
```

- Rezerva un blade (12 slot-uri):

```
qsub -cwd -q ibm-cell-qs22.q -pe openmpi*12 12 run_all.sh
```

Pentru a evita aparitia unor timpi mari de asteptare recomandam rezervarea unui singur slot pentru rularile pe date mici sau in scopuri de debugging si folosirea celui de-al doilea mod de submitere doar pentru rularile finale pe imaginile mari de input.

4 Punctaj

Punctajul este impartit astfel (100p tema + 20p bonus):

- 60p pentru paralelizarea operatiilor de comprimare si decomprimare
  - transferurile DMA sunt realizate prin utilizarea operatiilor de tip get si put (60p din 60)
  - se foloseste Software Managed Cache in loc de transferuri DMA explicite (30p din 60)
- 20p pentru utilizarea eficienta a instructiunilor vectoriale pe SPE-uri
- 20p pentru README dintre care:
  - 10p pentru explicatii legate de modul de paralelizare (descriere generala a modului de paralelizare, detalii legate de modul de sincronizare PPE - SPE, felul in care sunt gestionate transferurile DMA si utilizarea instructiunilor vectoriale)
  - 10p pentru rularea programului cu 1, 2, 4, 8 thread-uri SPU, prezentarea rezultatelor (timp de comprimare + decomprimare si timp total) si analiza performantei
- 20p (Bonus) pentru utilizarea corecta a mecanismului de double buffering,prezentarea rezultatelor (timp de comprimare + decomprimare si timp total pentru 1, 2, 4 si 8 thread-uri SPU) si analiza performantei

Deadline-ul soft de trimitere a temei este 3 mai 2014. Ultima zi in care se poate trimite tema este 8 mai 2014. Ajustarile de punctaj sunt urmatoarele:

- +10% din punctajul obtinut daca tema a fost trimisa pana pe 3 mai 2014 (de exemplu, daca ati obtinut 60p pe tema veti primi 6p bonus)
- -5p daca tema a fost trimisa pe 4 mai 2014
- -10p daca tema a fost trimisa pe 5 mai 2014
- -15p daca tema a fost trimisa pe 6 mai 2014
- -20p daca tema a fost trimisa pe 7 mai 2014
- -25p daca tema a fost trimisa pe 8 mai 2014

5 Recomandari

Puteti presupune ca ambele dimensiuni ale imaginii sunt multiplu de 16 (pentru a usura realizarea transferurilor DMA).

Puteti stoca pixelii din imaginea originala pe short (in loc de char) la fel cum se face in codul serial pentru a evita eventuale probleme de overflow si pentru a facilita utilizarea operatiilor vectoriale.