ALGORITMI PARALELI SI DISTRIBUITI:
Tema #3 Prelucrări de imagini folosind rețele MPI

Termen de predare: 09 Ianuarie 2018, ora 23:55

Cuprins

1. Cerințele temei

O imagine este reprezentarea unei poze si consta de obicei într-o matrice de numere (m x n). Fiecare element al unei imagini poarta numele de pixel si are o valoare care exprima intensitatea luminii sau culoarea (RGB).

Pentru simplificarea algoritmilor, in cadrul temei, se vor utiliza imagini in tonuri de gri (*gray-scale*) pentru care fiecare pixel va avea o valoare in intervalul *0 - 255*. Imaginile vor fi in formatul PGM (vezi ***Secțiunea 4***)

Exista o multitudine de algoritmi de procesare de imagini si fiecare poate beneficia de pe urma paralelizării. Mai mult, o aceeași operație este aplicata de multe ori unui flux de imagini. Operațiile de procesare de imagini variază de la algoritmi pe pixeli individuali (cum ar fi ajustarea contrastului), la operații locale pe grupuri de pixeli (uniformizare, reducerea zgomotului) pana la operații globale pe toți pixelii (codare, decodare).

Aceasta tema își propune sa implementeze o aplicație distribuita de procesare a unui set de imagini folosind procese modelate într-o topologie de tip arbore. Tema consta din 2 parți: procesarea fiecărei imagini in parte si statistica referitoare la numărul de linii prelucrate la nivel de proces. Detaliile de implementare pentru fiecare din aceste parți le găsiți in Secțiunea 2.

2. Implementare

Se considera ca procesele MPI comunica folosind o topologia predefinita (tip arbore). Topologia este stabila, nu vor apărea modificări ale sale in timpul rulării programului.
La pornire fiecare proces MPI va citi dintr-un fișier lista sa de adiacenta. In acest mod fiecare nod cunoaște doar nodurile cu care este conectat direct si poate comunica doar cu acestea.

Comunicarea (de input/rezultate) in aceasta topologie se va face conform modului de distribuție prezentat la algoritmi unda. Nodul *0* va fi considerat rădăcina arborelui. Programul va primi ca argument de intrare un fișier cu topologia aleasa,

din care fiecare nod va citi doar linia cu lista de adiacenta care ii aparține.

Implementarea poate fi realizata in C sau C++.

2.1 Prelucrarea distribuita de imagini folosind filtre

Programul MPI va trebui sa aplice diferite filtre pe imaginile de intrare, citite ca matrici de dimensiune (m x n). Pentru o imagine se poate alege unul dintre următoarele doua filtre: {"sobel", "mean_removal"}. Modul de implementare a fiecărui filtru este descris in ***Secțiunea 3***.

Modelul de prelucrare

Prelucrare unei imagini va fi implementata folosind paradigma Heartbeat. Idea de rezolvare se bazează pe împărțirea imaginii in P fâșii sau blocuri de pixeli. Fiecărui proces (numit Worker in cadrul paradigmei) i se va asigna un bloc. Pe lângă blocul de procesat un proces va primi si cele doua linii de graniță (sus si jos). Aceste linii de graniță sunt necesare deoarece filtrele vor acționa la nivel de pixel (punct din matrice), iar *pentru a calcula noua valoare a unui pixel e necesara si cunoașterea valorilor pixelilor vecini.*

In cadrul acestei teme doar nodurile frunza din topologia tip arborele vor fi *Workeri*. Orice alt nod din arbore va împarți in mod egal blocul primit de la părinte la fiecare nod copil.

Observație$_1$: Împărțirea la nodurile copii se face astfel: fiecare nod primește parte întreaga din *NR_Linii / NR_copii*; iar ultimul nod va lua restul. *(Exemplu: 203 linii la 3 noduri copii: 67, 67, 69)*

Observație$_2$: Daca s-ar ajunge la cazul *(puțin probabil)* in care blocul nodului curent ar conține mai puține linii decât numărul de noduri copii, doar primele noduri copii vor primi cate o linie din bloc (împreună cu granițele aferente).

Un bloc ajuns la un nod frunza va fi prelucrat folosind filtru aferent. *(Pentru a se specifica ce filtru sa fie folosit la frunze, mesajele de început din rețea pot folosi diferite tag-uri. In funcție de tag, frunza va ști ce filtru sa aplice blocului primit)*

La terminarea prelucrării locale, nodul frunza va trimite blocul rezultat (prin aplicarea filtrului) la părinte.

Un nod intermediar din topologia arbore va concatena bucățile de matrice primite de la toți copii, si trimite mai departe către părinte.

Nodul rădăcina (*rank 0*) va cumula bucățile primite si le va scrie intr-un fișier de ieșire (rezultând imaginea prelucrata).

Acești pași se vor repeta pentru fiecare imagine citita de nodul rădăcini. Rădăcina va trimite blocurile din următoarea imagine, doar după terminarea procesării imaginii anterioare.

După scrierea in fișier a ultimei imagini, rădăcina va trimite mai jos in arbore mesaje cu *tag*-ul de terminare.

2.2 Statistici pentru numărul de linii procesate

In ultima etapa, la recepționarea mesajului cu tag de terminare (*de la părinte*), o frunza va răspunde cu numărul de linii procesate in total. După trimiterea mesajului de tip statistica, procesul își termina execuția.

Observație$_1$: La acest număr se vor aduna liniile de la toate blocurile procesare. Nu contează ca dimensiunea unei linii poate fi diferita de la o imagine la alta.

Aceste statistici (de la toate nodurile) vor ajunge la rădăcina. Acesta le va scrie in fișierul de ieșire dat ca parametru, in ordinea crescătoare a rank-urilor. Pentru rădăcina si nodurile intermediare se va scrie „0” (linii procesare).

3. Filtre

Un filtru de convoluție este in esența o matrice de forma
0 0 0
0 1 0
0 0 0 / 1 + 0
aplicata **punctului (pixelului) curent** si punctelor din jurul lui. Fiecare punct capătă astfel o pondere, iar suma valorilor ponderate este împărțita la un factor si câteodată se aduna si un deplasament.
Matricea de mai sus se numește filtru identitate deoarece lasă neschimbata imaginea originala.
De obicei factorul este suma tuturor ponderilor, astfel încât valoarea finala sa fie in intervalul 0..255. Sunt cazuri in care suma ponderilor este 0 (de exemplu filtrele *emboss*). Atunci factorul va fi 1 iar deplasamentul va fi 127 (o valoare obișnuita), pentru a lumina imaginea finala.

Filtrele de convoluție sunt de dimensiuni variate, 3x3 este o dimensiune normala dar exista de exemplu si filtre 7x7. Unele din filtre sunt simetrice, aplicate uniform punctelor din jur, altele sunt nesimetrice, cum ar fi filtrele de detectare a marginilor.
Cu cat un filtru este de dimensiune mai mare cu atât va rămâne o margine mare care nu poate fi prelucrata. Pentru o matrice 3x3 precum cea din exemplu va rămâne cate un pixel neprelucrat pe fiecare margine.
In cadrul acestei teme se vor procesa toți pixelii, chiar si cei de pe marginile imaginii. Pentru acest lucru veți borda cu 0 matricea (m x n) cu imaginea.

Observație$_1$: In cazul temei un pixel poate avea valori întregi in intervalul [0..255]. Daca in urma prelucrării se ajunge la o valoare înafara acestui interval, aceasta se va aduce la cea mai apropiata valoare valida. Daca este cazul, in urma împărțirii la factor, se rotunjește la cel mai apropiat întreg.

A. Detecţia marginilor (*edge dection - Sobel*)
In detectarea marginilor nu ne interesează valoarea efectiva a punctului curent, cat diferențele intre punctele din jurul lui. De aceea aceste filtre aplica o valoare negativa pe una din laturi si o valoare pozitiva pe latura opusa. Rezultatul este o valoare apropiata de 0 pentru punctele care au culoare apropiata si valoare mare pentru punctele care au culori diferite.
Exista mai multe filtre de acest tip, însă vom considera varianta *Sobel pe X*.
Pentru fiecare pixel se va calcula noua valoare, folosind următoarea matrice de convoluție:
1 0 -1
2 **0** -2
1 0 -1 / 1 + 127

B. Eliminarea mediilor (*mean removal*)
Mean removal este un filtru de accentuare, asemănător cu filtrul *sharpen*. Dar *sharpen* funcționează doar pe orizontala si pe verticala, pe când acest filtru acționează si pe diagonale.
Pentru fiecare pixel se va calcula noua valoare, folosind următoarea matrice de convoluție *(Valoarea centrala poate fi modificata pentru a modifica gradul de accentuare)*:
-1 -1 -1
-1 **9** -1
-1 -1 -1 / 1 + 0

4. Formatul datelor de intrare/ieșire.

Programul va primi ca argumente in linia de comanda: fișierul ce conține topologia, fișierul de listare a imaginilor de procesat si fișierul de ieșire cu statistica pe linii.

Rularea aplicației:

mpirun -np N ./filtru topologie.in imagini.in statistica.out

Observație: Se va rula mereu cu un număr de procese **egal** cu numărul de id-uri din topologie.in

a. Fișierul de topologie < *topologie.in* >

- Fiecare linie va avea următorul format *id_nod: $vecin_1$ $vecin_2$ ... $vecin_i$*

Exemplu:
0: 1 2
1: 0 3 4 5 6
2: 0 7 8
3: 1
4: 1 9 10
5: 1
6: 1
7: 2
8: 2 11
9: 4
10: 4
11: 8

b. Fișierul de listare a imaginilor < *imagini.in* >

Acesta are următorul format:

- pe linia I: numărul NF de imagini de intrare
- pe următoarele NF linii: $filtru_i$ $nume\_imagine_i$ $nume\_imagine\_prelucrata_i$

Pentru filtru se poate pune: mean_removal, sobel

Exemplu:
3
sobel image1.pgm image1-s.pgm
mean_removal image2.pgm image2-m.pgm
sobel image3.pgm image3-s.pgm

Observații:

- Formatul unui fișier de tip imagine:
  - fișierul va avea formatul **PGM** a cărui descriere o găsiți aici si care poate fi vizualizat cu diverse viewere (*gimp*, *xnview*, etc.).
  - pentru a converti o imagine dintr-un anume format (ex. JPG) in format PGM se poate folosii următoarea metoda: deschideți poza cu GIMP. Selectați **File →Export as...** Se va alege formatul PGM. La opțiunile de **„Data formatting"** selectați formatul **ASCII** (si NU **RAW**!). Imaginea rezultata va fi in tonuri de gri.
  - Fișierul de intrare poate avea linii comentate cu #
  - Imaginea (din fisierul de intrare) este reprezentata de o matrice de dimensiunea *width* * *height*. Ca principiu general, intr-un fisier PGM delimitarea dintre pixeli se face prin "whitespace" = {blanks, TABs, CRs, LFs}
  - Considerați ca formatul unui fișier de tip imagine va fi mereu cel rezultat in urma prelucrări cu *gimp:*
    P2
    # CREATOR: GIMP PNM Filter Version x
    W H
    255
    val_pixel_1
    .....
    val_pixel_{WxH}
  - Fișierul de ieșire cu imaginea prelucrata trebuie sa respecte același format ca si fișierul de intrare (PGM)

Exemplu: Poza aceasta in format JPG s-a convertit folosind GIMP. Rezultatul este aici.

c. **Fișierul de ieșire < statistica.out >**

Exemplu:

0: 0
1: 0
2: 0
3: 500
4: 500
5: 1000

5. Teste si punctare

Pentru a verifica funcționalitatea temei folosi aceste teste. *(testele urmează sa fie adăugate)*

Punctajul temei va fi distribuit astfel:

- 60p: Implementarea corecta a filtrelor la nivelul proceselor frunza *(30 per filtru)*
- 20p: Procesarea corecta a tuturor imaginilor date ca input.
- 20p: Consistenta rezultate de statistica

6. Conținutul arhivei temei

„Fișierele și directoarele care contribuie la rezolvarea temei trebuie OBLIGATORIU împachetate într-o arhivă de tip '.zip', cu numele '**Grupa_NumePrenume_TemaX.zip**'"

Arhiva temei va conține codul sursa pentru tema împreună cu fișierele *makefile* si *Readme*.
Temele care nu conțin si aceste doua fișiere nu vor fi corectate!

Fișierul *makefile* va conține (obligatoriu) regulile „*build*" si „*clean*" (ștergere binar).
In urma compilării, **binarul rezultat e obligatoriu sa se numească „filtru"** si trebuie sa se ruleze conform cu specificațiile din **Secțiunea 4.**

7. Resurse (optional)

[1] Image Convulation http://web.pdx.edu/~jduh/courses/Archive/geog481w07/Students/Ludwig_ImageConvolution.pdf
[2] Gimp-Convulation Matrix https://docs.gimp.org/en/plug-in-convmatrix.html