Blur Filter

Maria Moșneag
343C1

Prezentarea problemei

- Aplicarea unui filtru de blur asupra unei imagini.
- Acest procedeu presupune înlocuirea fiecărei componente (R, G, B) a fiecărui pixel cu media valorilor componentei respective în pixelii vecini.
- Pentru a crește complexitatea problemei, am decis ca pixelii “vecini” să fie considerați toți pixelii care se găsesc într-un pătrat cu latura 2 * BLUR_DEGREE + 1, care are centrul în pixelul curent. BLUR_DEGREE este o constantă ce poate fi modificată (în fișierul utils/const.h).
- Am ales să folosesc imagini de tip .ppm (Portable Pixmap Format) deoarece, spre deosebire de alte formate mai des întâlnire (.jpg, .png), acestea stochează pixelii sub forma unei matrice simple, necomprimate.

Exemplu input (5184 x 3456)

Exemplu output (5184 x 3456)

Exemplu input (640 x 426)

Exemplu output (640 x 426)

Paralelizare cu OpenMP

- 4 threaduri
- Am adăugat o directivă de paralelizare pentru for-ul care parcurge liniile imaginii.
- M-am asigurat că variabilele independente în cazul calculelor corespunzătoare unui anumit pixel (neigh = numărul de vecini, sum_r = suma valorilor componentelor R ale vecinilor, sum_g = analog pentru G, sum_b = analog pentru B) sunt private.
-

```
#pragma omp parallel for private(neigh, sum_r, sum_g, sum_b)
```

Paralelizare cu MPI

- 4 procese
- Procesul coordonator (rank == 0) trimite un număr egal de rânduri (consecutive) din imagine fiecărui proces. Rândurile trimise conțin rândurile care trebuie prelucrate de fiecare proces + cele BLUR_DEGREE rânduri dinaintea acestora + cele BLUR_DEGREE rânduri de după acestea (dacă acest lucru este posibil), astfel încât procesele să primească și informația despre toți vecinii pixelilor de care sunt “responsabile”.
- După ce fiecare proces termină prelucrarea, bucata de imagine este trimisă coordonatorului care reface imaginea.
- Pentru trimiterea informației am definit o nouă structură cu ajutorul MPI_Type_create_struct.

Paralelizare cu Pthreads

- 4 threaduri
- Fiecare thread prelucrează un număr egal de rânduri.
- Pixelii modificați sunt salvați într-o nouă imagine care o înlocuiește pe cea inițială abia după ce toate threadurile și-au terminat prelucrarea.

Paralelizare cu MPI și OpenMP

- 2 procese cu câte 2 threaduri
- Procesul coordonator (rank == 0) trimite un număr egal de rânduri celorlalte procese (în cazul acesta trimite a doua jumătate a imaginii procesului 1 și “își păstrează pentru el” prima jumătate de imagine.
- După primirea rândurilor corespunzătoare din imagine, fiecare proces aplică în paralel filtrul, cu două threaduri, folosind o directivă pragma.
- La final, bucățile de imagine sunt trimise înapoi procesului coordonator care recompune imaginea.

Paralelizare cu MPI și Pthreads

- 2 procese cu câte 2 threaduri
- Abordarea este asemănătoare cu cea anterioară, doar că, în loc să fie folosită o directivă pragma pentru pornirea threadurilor, se folosesc funcții din biblioteca Pthreads.
- Din bucata de imagine corespunzătoare procesului, fiecare thread prelucrează un număr egal de linii consecutive.

Rezultate obținute (I)

- Pentru a analiza performanțele diferitelor implementări, am contorizat timpul necesar aplicării filtrului, ignorând timpul necesar operațiilor de I/O.
- Am folosit de asemenea și imagini de dimensiuni diferite.
- În rulările luate în considerare pentru rezultatele prezentate în continuare, BLUR_DEGREE = 5 (adică pentru fiecare pixel, se parcurgeau maxim 121 pixeli vecini).
- Dimensiunile primei imagini: 1920 x 1280
- Dimensiunile imaginii 2: 5184 x 3456

Rezultate obținute (II)

- Rezoluția imaginii: 1920 x 1280

serial	1.873957 secunde	6
OpenMP (4 threaduri)	0.970779 secunde	2
MPI (4 procese)	1.137925 secunde	5
Pthreads (4 threaduri)	0.950454 secunde	1
MPI & OpenMP (2 procese cu câte 2 threaduri)	1.073094 secunde	4
MPI & Pthreads (2 procese cu câte 2 threaduri)	1.026306 secunde	3

Rezultate obținute (III)

- Rezoluția imaginii: 5184 x 3456

serial	14.634868 secunde	6
OpenMP (4 threaduri)	6.902738 secunde	1
MPI (4 procese)	8.499982 secunde	5
Pthreads (4 threaduri)	6.937866 secunde	2
MPI & OpenMP (2 procese cu câte 2 threaduri)	7.831352 secunde	4
MPI & Pthreads (2 procese cu câte 2 threaduri)	7.498362 secunde	3

Concluzii

- Din rezultatele anterioare se observă că:
  - Paralelizările cu OpenMP și cu Pthreads sunt cele mai rapide și, în același timp, rezultatele lor sunt comparabile.
  - Variantele hibride sunt următoarele ca viteză de rulare; dintre acestea, implementarea cu MPI și Pthreads este puțin mai rapidă decât cea cu MPI și OpenMP.
  - Paralelizarea cu MPI este cea mai lentă.
  - Toate variantele paralelizate sunt semnificativ mai rapide decât varianta serială.