CSSO - Tema 6

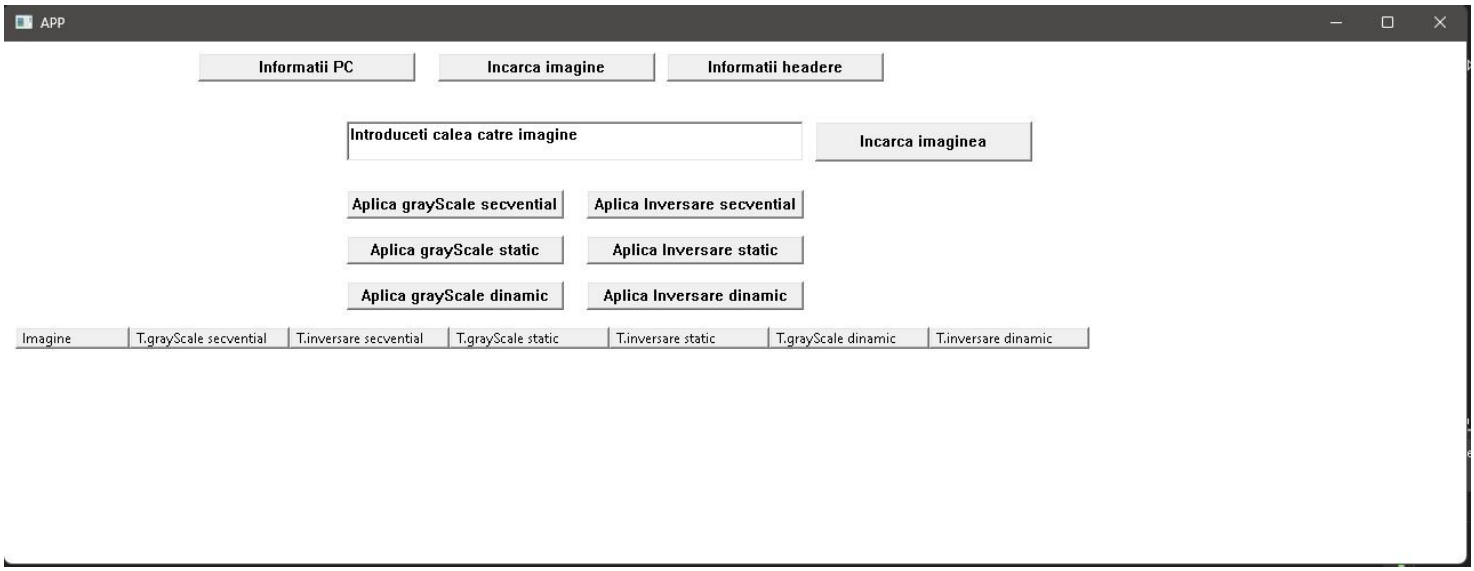
Ghimpu Ioana - Ingrid
Ceobanu Alexandra

Introducere

Am realizat un proiect care modifica imagini în format BMP în doua moduri inversare și grayscale.

Proiectul folosește trei variante de calcul diferite pentru a realiza operațiile enumerate anterior secvential, paralelizat static și paralelizat dinamic.

Pentru a face utilizarea aplicației mai accesibilă am construit o interfata grafica care contine diverse campuri de diferite functionalitati, atât care facilitează - informații utile în realizarea operațiilor date -sau - rezultatele obținute după realizarea acestora - cat și câmpuri de input necesare pentru obținerea rezultatelor.



Screenshot cu interfata aplicatiei

Informații PC personal:

- **SID-ul** utilizatorului curent, a grupului everyone si al grupului administrator - identificatori de securitate unic pentru fiecare utilizator sau grup

    - SID utilizator curent - **S-1-5-21-1917181282-2593414177-3942959740-1001**

    - SID grup everyone - **S-1-1-0**

    - SID grup administrator - **S-1-5-21-1917181282-2593414177-3942959740-500**

- **Hyperthreaded systems**:

    - ProcessorMask - nucleele logice - **63**

    - Relationship - tipul de informație furnizată pentru un anumit procesor sau grup de procesoare.

- **NUMA API**:

    - lpProcessAffinityMask - **0x3f**

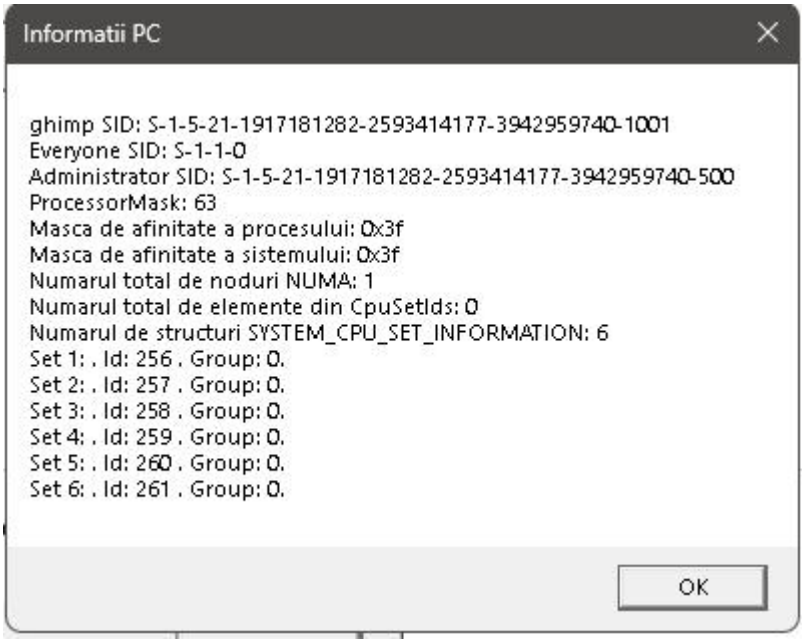
    - lpSystemAffinityMask - **0x3f**

    - Număr total noduri NUMA - **1**

- **CPU Sets** - seturi de procesare:

    - Elemente în CPUSetIds - **0**

    - Structuri SYSTEM-CPU-SET-INFORMATION - **6**



Exemplu de afișare a informațiilor PC-ului

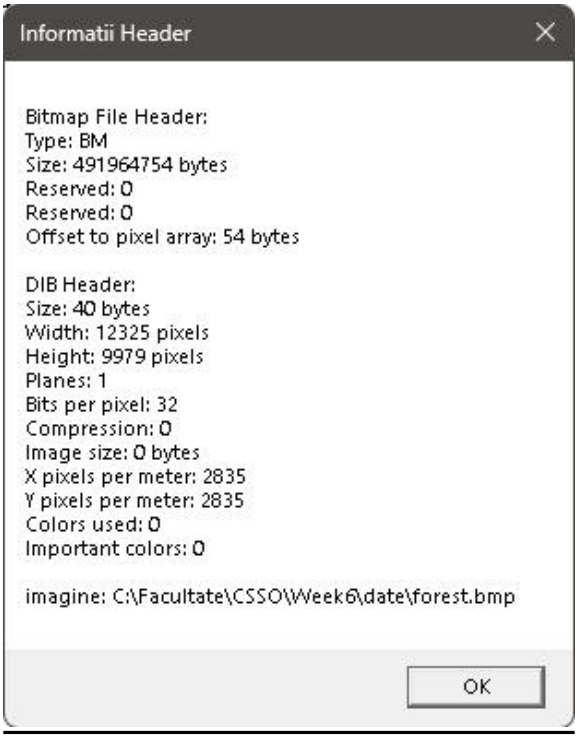
Mediul de lucru

Programul realizat face operațiile de schimbare asupra unor imagini în format BMP. Toate imaginile în acest format au un Bitmap File Header care contine următoarele informații:

- Type - BM (extensia fișierului)
- Size - dimensiunea în bytes a fișierului
- Reserved 1, 2 - 0 ()
- Offset - 54 bytes (adresa de start a vectorului de pixeli)

și un DIBHeader:

- Size - 40 bytes (dimensiunea headerului)
- Width - lățimea imaginii
- Height - înălțimea imaginii
- Planes - 1
- Bits per pixel - 32
- Compression - 0
- Image size - 0
- X pixels per meter - 2835
- Y pixels per meter - 2835
- Colors used , Important colors - 0



Screenshot din interfață cu informațiile din header-ul imaginilor

Testarea

Pentru fiecare imagine am aplicat cate 2 operații, în cadrul cărora am aplicat o formula pentru fiecare byte i, cu i de la 0 la numărul total de bytes al imaginii:

- inversarea - formula:

  0xFF - val_byte
- grayscale - formula:

  0.299 * R + 0.587 * G + 0.114 * B

Operațiile descrise mai sus au fost aplicate folosind 3 metode diferite de calcul: secvential, paralelizat static și paralelizat dinamic.

1. Metoda secvențială

Pentru aceasta metoda am parcurs toți bytes imaginii pe rand și pentru fiecare din ei am aplicat formulele descrise mai sus, în cadrul funcțiilor *“inversare”* și *“grayScale”*.

Aceasta metoda de calcul a transformat în totalitate imaginile și are un timp mediu de rulare de ***0.4764 secunde***.

2. Metodele paralelizate

Pentru aceasta metoda ne-am folosit de numărul total de procesoare al PC-ului și am calculat astfel un număr de **2 * nr. procesoare** thread-uri. Pentru a imparti numărul total de calcule am folosit 2 metode:

2.1. Static

Fiecare thread primește un număr de bytes pe care să-l modifice, număr calculat astfel:

nr_bytes_imagine / (2 * nr_thread-uri).

Formula de mai sus o aplicăm pentru fiecare thread, mai puțin ultimul, care va calcula noua valoare pentru numărul de bytes rămas până la finalul imaginii.

Știm ca aceasta metoda împarte numărul total de bytes corect (nu modifica pixeli pe jumătate), deoarece acesta este un multiplu de 4 (un pixel de forma RGBA) pe care il impartim la **2 * 2 * nr_procesoare**, adică la **4 * nr_procesoare** care va fi un multiplu de 4.

Pentru modificarea pixelilor se apelează aceleași funcții menționate mai sus.

Aceasta metoda a transformat corect imaginile doar prin folosirea unui mutex la accesarea acestora și are un timp mediu de rulare de ***1.901 secunde***.

2.2. Dinamic

În acest caz, calculele au fost împărțite thread-urilor pe masura ce acestea termina calculele anterioare, prin apelarea funcției *“Calculate_Chunks”*. Fiecare thread are acces la aceasta functie doar pe baza unui mutex comun, astfel ca 2 thread-uri nu vor primi aceeași bytes.

Calcularea chunk-urilor se face astfel:

nr_bytes_ramasi / (2 * nr_thread-uri)

Rezultatul obtinut trebuie acum verificat, iar dacă nu este un multiplu de 4 îl incrementam pană devine unul. Dupa aceasta, scadem din numarul de bytes ramasi numarul alocat.

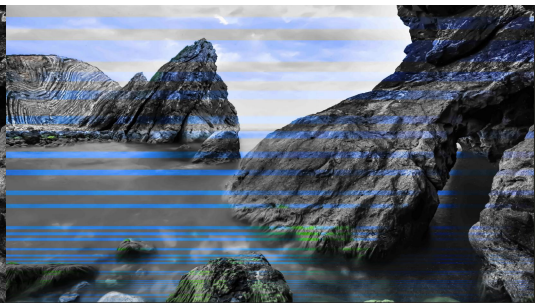
Pentru a evita blocarea programului atunci cand rezultatul impartirii este 0, cand se ajunge în acest punct thread-ul care a apelat funcția va primi toți pixelii ramai si *nr_bytes_ramasi* devine 0.

Aceasta metoda nu transforma imaginile 100% corect, la fiecare ramanand o linie de pixeli nemodificata si are un timp mediu de rulare de ***86.6464 secunde***.

Grayscale secvential

Grayscale static

Grayscale dinamic dacă <u>nu</u> este prima operație făcută



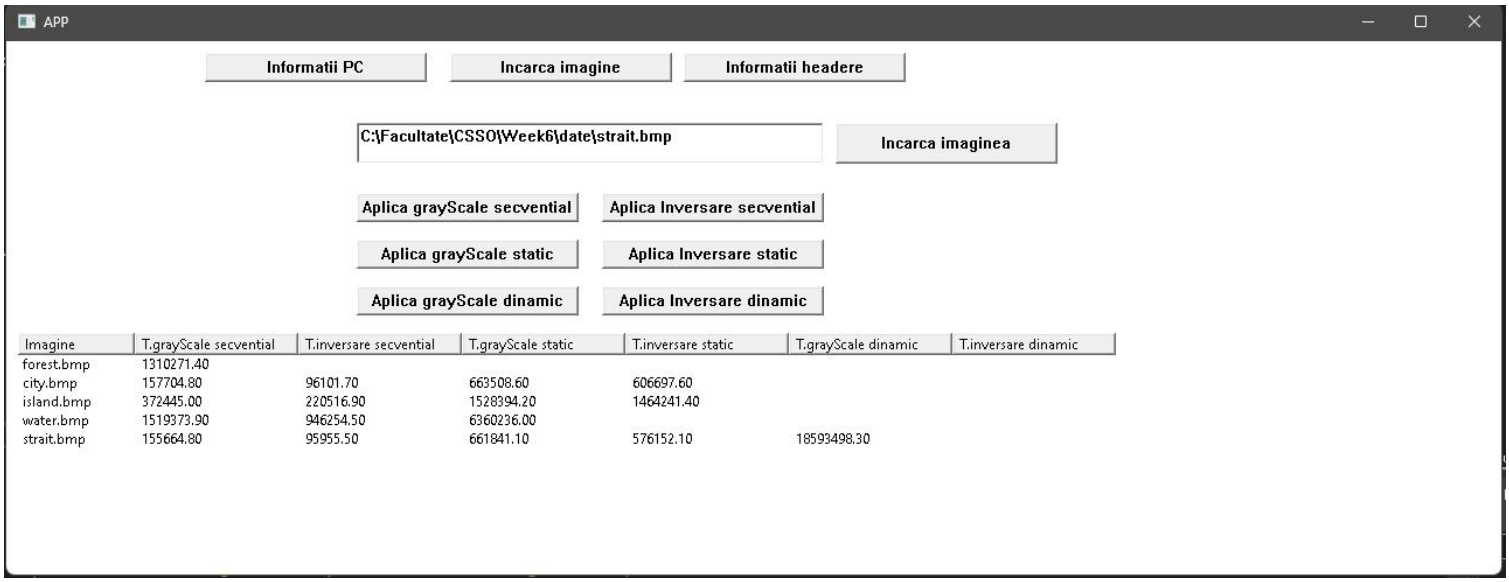
Grayscale dinamic dacă este prima operație făcută

Pe imaginea de mai sus am aplicat aceeași operație în același mod de calcul ca și ultima poza de pe pagina anterioara (*grayscale dinamic*, prima operație executată de la deschiderea programului), dar pe un dispozitiv care are numărul de structuri SYSTEM_CPU_SET_INFORMATION **16**, în loc de **6**, deci **32 de thread-uri** în loc de **12**. Se poate observa ca numărul de pixeli netransformati este mult mai mic.

Diferenta dintre rezultate în funcție de momentul aplicării operațiilor în mod dinamic (prima operație sau cel puțin a doua din momentul deschiderii programului) este valabilă la ambele dispozitive.

De asemenea, o altă diferență notabilă este faptul ca, pe dispozitivul **6** seturi de procesoare nicio operatia calculata in mod static nu rulează pentru imaginea *forest.bmp* (asumare făcută după ce am așteptat timp de 15 minute și programul nu raspundea)(de aceea lipsa valorilor în tabelul de la concluzii), însă pe dispozitivul din urma, acestea rulează, avand un timp de **10.0257** secunde petru grayscale, respectiv **9.442** secunde pentru inversare.

Concluzii



screenshoturi din interfață după transformarea mai multor imagini

	nr bytes	**operație secvențială**		**operație statică**		**operație dinamica**	
		grayscale	inversare	grayscale	inversare	grayscale (ultima op)	inversare (prima op)
forest.bmp	*491.964.754*	1.3589 s	0.8324 s	-	-	172.02 s	181.09 s
city.bmp	*58.982.454*	0.1550 s	0.1154 s	0.6510 s	0.5837 s	44.1504 s	43.6197 s
island.bmp	*132.710.454*	0.3563 s	0.2135 s	1.4146 s	1.2751 s	45.2627 s	52.5605 s
mountain.bmp	*132.710.454*	0.3518 s	0.2142 s	1.4719 s	1.3581 s	41.6464	37.1104
stones.bmp	*132.710.454*	0.3527 s	0.2083 s	1.4493 s	1.2907 s	39.6546 s	43.1044 s
strait.bmp	*58.982.454*	0.1547 s	0.0938 s	0.6442 s	0.5746 s	17.7831 s	19.7963
water.bmp	*530.841.654*	1.4063 s	0.8573 s	6.2861 s	5.8126 s	234.2487 s	242.0024 s

Timpii obtinuti pentru fiecare imagine, utilizand toate cele trei metode de calcul

După modificarea mai multor imagini am observat ca metoda secvențială este cea mai rapidă în cadrul programului nostru (de aproximativ 5 ori mai rapidă decat metoda statică), iar metoda statică este de 30-40 de ori mai rapidă decat cea dinamica.

Acest rezultat eronat este probabil o urmare a faptului ca în cadrul aplicației noastre imaginea modificata nu este adusa in memoria virtuala inainte de inceperea operatiilor de transformare.

Am observat, de asemenea, faptul ca metoda dinamica nu transforma mereu imaginile in totalitate corect. Atunci cand celelalte metode sunt aplicate inainte de cea dinamica, aceasta modfica imaginea in totalitate dar, cand este prima metoda aplicată, nu transforma toți pixelii imaginii. Celelalte 2 metode, însă, funcționează corect.

Rezultatele la inversare sunt aceleași cu cele obținute la Grayscale.