Descriere Detaliata Date rulari Cuda

1. Tabel valori

Caz (Dimensiune)	CPU (Secvențial)	GPU (Block 64)	GPU (Block 128)	GPU (Block 256)	GPU (Block 512)
10 x 10	0.0016 ms	0.3123 ms	-	-	-
1000 x 1000	9.3043 ms	27.6259 ms	26.8311 ms	29.7167 ms	28.5252 ms
10000 x 10000	922.3883 ms	285.5845 ms	**267.7944 ms**	293.6017 ms	291.98 ms

2. Analiza pe cazuri

[Case 1] Matrice 10 x 10 (mica)

- **Rezultat:** CPU (0.0016 ms) vs GPU (0.3123 ms).
- **Câștigător: CPU** (de ~195 de ori mai rapid).
- **Explicație:**
  - **Fenomenul:** La dimensiuni atât de mici (100 de pixeli total), timpul de execuție nu este dominat de *calcule*, ci de **overhead** (timpul de pregătire).
  - **CPU:** Datele încap complet în cache-ul L1 al procesorului. Execuția este aproape instantanee.
  - **GPU:** Pentru a rula codul, driverul video trebuie să trimită comanda, să aloce resurse și să pornească nucleele. Acest "ritual" de pornire durează constant aproximativ 0.3 ms, indiferent dacă calculezi 1 pixel sau 1000.
  - **Concluzie:** Nu folosi GPU pentru date minuscule; costul transportului datelor și lansării execuției este mai mare decât timpul de calcul efectiv.

[Case 2] Matrice 1000 x 1000 (medie)

- **Rezultat:** CPU (~9 ms) vs GPU (~27 ms).
- **Câștigător: CPU** (de ~3 ori mai rapid).

- **Explicație Generală (De ce pierde GPU?):**
  - Deși 1 milion de pixeli pare mult, algoritmul tău are o constrângere specifică: **procesarea linie cu linie**.
  - Asta înseamnă că GPU-ul este pornit și oprit de **1000 de ori** (câte o lansare de kernel pentru fiecare linie).
  - Suma timpilor morți (overhead-ul de 1000 de lansări) depășește timpul câștigat prin paralelizarea calculului pe acea linie.
- **Analiza pe Block Size (GPU):**
  - **Block 64 & 128 (26-27 ms):** Sunt cele mai eficiente aici. Deoarece o linie are 1000 de pixeli, ai nevoie de puține blocuri (ex: 1000 / 128 ≈ 8 blocuri). Scheduler-ul GPU gestionează rapid acest număr mic.
  - **Block 256 & 512 (28-29 ms):** Ușor mai lente. Când folosești blocuri mari pentru o dimensiune fixă (1000), poți ajunge la o utilizare ineficientă a thread-urilor (ex: ultimul bloc e pe jumătate gol) sau la o rigiditate în planificare (occupancy mai mic).

[CASE 3] Matrice 10000x10000 (Mare)

- **Rezultat:** CPU (922 ms) vs GPU (267 ms).
- **Câștigător: GPU** (de ~3.44 ori mai rapid).
- **Explicație Generală:**
  - Acesta este punctul de inflexiune. Avem **100 de milioane de pixeli**.
  - Chiar dacă avem overhead-ul celor 10.000 de lansări de kernel, **puterea brută de calcul** a GPU-ului intră în scenă. GPU-ul procesează o linie întreagă (10.000 pixeli) simultan, în timp ce CPU-ul trebuie să ia pixelii la rând. Timpul masiv economisit la calcul acoperă și depășește timpul pierdut cu lansările.
- **Analiza pe Block Size (GPU) - De ce diferă timpii?**
  - **Block 64 (285.5 ms):** *Prea mic.* GPU-urile NVIDIA lucrează în grupuri de 32 de fire (Warps). Un bloc de 64 are doar 2 Warps. Pentru a ascunde latența memoriei, un Multiprocesor (SM) are nevoie de multe Warps active. Blocurile prea mici cresc overhead-ul de gestionare al scheduler-ului hardware.
  - **Block 128 (267.8 ms - OPTIM):** *"Sweet Spot".* Acesta oferă cel mai bun echilibru. 128 thread-uri (4 Warps) este o dimensiune care permite GPU-ului

să încarce eficient multiprocesoarele, maximizând **Occupancy** (gradul de ocupare).

- **Block 256 (293.6 ms):** *Ineficient.* O linie are 10.000 pixeli. Împărțirea la 256 nu este la fel de curată ca la 128 în termeni de alocare pe hardware.
- **Block 512 (291.9 ms):** *Prea rigid.* Blocurile mari consumă mulți regiștri și memorie shared per bloc. Asta înseamnă că pe un singur Multiprocesor (SM) vor încăpea mai puține blocuri simultan. Dacă un SM așteaptă după memorie, are mai puține blocuri "de rezervă" la care să comute, deci stă degeaba mai mult timp.

3. Explicatie diferente timpi. GPU - CPU cine e mai rapid.

caz	matrice	pixeli totali	timp CPU	timp GPU	de ce?
1	10 x 10	100	0.0016ms	0.31ms	Date prea putine.
2	1000 x 1000	1.000.000	9.30ms	26.83ms	Algoritmul lansand linie cu linie thread uri incetineste timpul. Se porneste si se opreste de prea multe ori
3	10000 x 10000	100.000.000	922.39ms	267.79ms	Datele multe fac folosirea CUDA si a GPU-ului echitabil.