**CUDA**

**Rusu Dana**

**Stan Ariana-Maria**

**1. Descriere generală**

Acest laborator prezintă implementarea unei operații de **convoluție** asupra unei matrici (implementare lab. 2), utilizând **CUDA** pentru a paraleliza procesarea pe GPU. Algoritmul se bazează pe împărțirea calculelor între multiple fire (**threads**) și blocuri (**blocks**) pentru a accelera operația.

**2. Funcționalitatea principală**

Programul:

1. Citește o matrice și un kernel de convoluție dintr-un fișier de intrare.
2. Extinde matricea prin adăugarea unui padding (prin dublare) pentru a evita pierderea de informații la marginile matricei.
3. Aplica convoluția utilizând un **kernel CUDA**, rulând paralel pe GPU.
4. Salvează rezultatul într-un fișier și verifică corectitudinea față de un fișier de referință.

**3. Integrarea CUDA**

**3.1 Configurarea grilei și a blocurilor**

* **Threads per block**: Numărul de fire per bloc este definit prin:
  + Fiecare bloc conține 256 fire (16x16).
  + Aceste fire procesează simultan elemente din matrice.
* **Number of blocks**: Numărul de blocuri necesare pentru a acoperi întreaga matrice este calculat astfel:
  + Formula rotunjește în sus pentru a acoperi întreaga matrice.
  + Ex.: O matrice de 50x50 cu 16x16 fire per bloc necesită:
    - numBlocks.x=⌈50/16⌉=4
    - numBlocks.y=⌈50/16⌉=4

**3.2 Alocarea memoriei pe GPU**

Datele necesare pentru convoluție sunt copiate pe GPU înainte de lansarea kernel-ului:

* **cudaMalloc** alocă memorie globală pe GPU.
* Matricea cu padding (padded), kernel-ul (conv) și matricea rezultată (original) sunt transferate pe GPU.

**3.3 Copierea datelor CPU → GPU**

* **cudaMemcpy** transferă datele între memoria CPU și GPU.
  + **cudaMemcpyHostToDevice**: Copierea de pe CPU (host) către GPU (device).

**3.4 Lansarea kernel-ului CUDA**

Kernel-ul CUDA este lansat utilizând configurația de grilă și blocuri:

* **<<<numBlocks, threadsPerBlock>>>** definește dimensiunea grilei și a blocurilor:
  + **numBlocks**: Numărul total de blocuri.
  + **threadsPerBlock**: Numărul de fire per bloc.

**4. Kernel-ul CUDA**

Kernel-ul CUDA efectuează calculele de convoluție pentru fiecare element al matricei rezultate.

**Funcționare:**

1. **Poziția fiecărui fir**:
   * Se calculează rândul și coloana pe care firul trebuie să le proceseze utilizând:

row=blockIdx.y×blockDim.y+threadIdx.y

col=blockIdx.x×blockDim.x+threadIdx.x

1. **Convoluția**:
   * Fiecare fir calculează suma ponderată a vecinilor unui element, utilizând valorile din kernel.
2. **Stocarea rezultatului**:
   * Rezultatul calculat de fiecare fir este scris în matricea originală la poziția corespunzătoare.

**5. Copierea rezultatelor GPU → CPU**

După execuția kernel-ului, rezultatele sunt copiate din memoria GPU înapoi pe CPU.

**6. Avantaje ale utilizării CUDA**

* **Paralelizare masivă**: Fiecare fir procesează simultan un element al matricei.
* **Timp redus de execuție**: CUDA accelerează procesarea prin utilizarea mai multor unități de calcul ale GPU-ului.
* **Scalabilitate**: Configurarea grilei și blocurilor permite procesarea matricelor de orice dimensiune.

**7. Output**

1. **Timp de execuție**:
   * Timpul necesar pentru aplicarea convoluției este măsurat utilizând biblioteca chrono.
2. **Fișier rezultat**:
   * Matricea rezultată este salvată într-un fișier output.txt.
3. **Verificarea corectitudinii**:
   * Rezultatul este comparat cu un fișier de referință GT.txt utilizând funcția compareFiles.

**Concluzie**

Acest laborator demonstrează eficiența utilizării **CUDA** pentru a accelera operații compute-intensive, cum ar fi **convoluția matricială**, prin distribuirea sarcinilor între multiple unități de calcul ale GPU-ului. Integrarea CUDA implică alocarea memoriei pe GPU, transferul datelor, configurarea grilei și blocurilor pentru execuție paralelă și lansarea unui kernel care procesează simultan toate elementele matricei. Prin acest laborator, am învățat:

* Cum să împărțim datele între **blocuri** și **fire**.
* Cum să utilizăm memoria GPU în mod eficient.
* Cum să evaluăm și să optimizăm performanța utilizând **kernel-uri CUDA**.

Timpi pentru matrice de 10x10 si kernel de 3x3

|  |  |  |
| --- | --- | --- |
| Metoda | GPU (CUDA) | CPU(Thread-uri) |
| Timp estimat (ms) | ~0.1-0.5 | ~5-15 |