1 Bevezetés

1.1 Parancssori fordítás

Készíts egy-egy 5 elemű tömböt a host és a device memóriában. Másold át a GPU-ra az adatokat, ott szorozd meg az összes számot egy megadott értékkel, majd az eredmény tömböt másold vissza.

Használd a parancssori fordítót az elkészített kód lefordításához!

Vizsgáld meg az elkészült segédállományokat!

1.2 Visual Studio környezet használata

Készítsd el az előző feladatot a Visual Studio környezetben!

Fordítsd és futtasd le az elkészített programot!

Vizsgáld meg a környezet adta lehetőségeket!

2 Egyszerű kernelek

2.1 Egy és többdimenziós indextér használata

Készíts egy alkalmazást, ami egy előre megadott hosszúságú (N) szövegben megkeres egy előre megadott hosszúságú (M) szövegrészt. Amennyiben több találat is van, akkor ezek közül elég ha csak az egyiket adja vissza. Az alkalmazást az alábbi lépésekben készítsd el:

- CPU implementáció
- GPU implementáció 1 szállal
- GPU implementáció N szállal
- GPU implementáció NxM szállal

Vizsgáljuk meg a megoldásokat a szinkronizáció szempontjából!

2.2 Kétdimenziós indextér

Készíts egy alkalmazást, ami egy tömbben eltárolt „kép” pixeleit elmossa. Ehhez hozzál létre egy WxH méretű tömböt (WxH legyen kisebb mint 512), töltsd fel adatokkal (0..255), majd minden pixelt cserélj ki a körülötte lévő pixelek átlagával (középső + 4 szomszéd). A széleken kilógó pixelek helyett mindig a középső pixel színét vegyük figyelembe.

- CPU implementáció
- GPU implementáció 1 szállal
- GPU implementáció WxH szállal

Miként lehetne gyorsítani a széleknél látható kivételes esetek kezelését?

3 Több blokk használata

3.1 Rendezés

Készíts egy alkalmazást, amely egy blokk segítségével rendezni tud egy N elemű számsorozatot. Az alkalmazást az alábbi lépésekben készítsd el:

- GPU implementáció egy blokkon belül (tehát N <= 1024)
- GPU implementáció több blokk segítségével
    - Blokkonkénti rendezés a fenti módszerrel

- GPU implementáció shared memória használatával
- Blokkok összefuttatása a GPU-n

3.2 Kép elmosás több blokk segítségével

Az előző alkalommal készített „kép” pixeleinek elmosását végző alkalmazást egészítsük ki úgy, hogy a blokkméretnél nagyobb tömbök esetén is működjön.

4 Memória architektúra

4.1 Jelszó törés

Készíts programot, ami létrehoz egy TEXT_N hosszú szöveget, illetve egy KEY_N hosszú kulcsot. A megadott kulcscsal titkosítsd a szöveget valamilyen egyszerű módon. Ezt követően:

- Az eredeti és a titkosított szöveg ismeretében törd fel a kulcsot a CPU-n
- Végezd el ugyanezt a GPU-n
- Futtassuk le ugyanezt a kernelt equals és L1 cache elsőbbségi beállítással
- Végezd el ugyanezt a GPU-n úgy, hogy a nem változó adatok a konstans memóriában vannak
- Egészítsük ki ezt úgy, hogy a generált kulcsokat a dinamikusan foglalt shared memóriában tároljuk

Hasonlítsd össze az egyes módszerek időigényét!

4.2 Mátrix szorzás megvalósítása

Hozz létre két NxN méretű mátrixot (A és B), töltsd fel őket véletlen adatokkal. Ezt követően:

- Számold ki az A*B szorzatot a CPU-n
- Számold ki az A*B szorzatot a GPU-n. Használj dinamikus memóriafoglalást
- Valósítsd meg a shared memóriával való gyorsítást

Hasonlítsd össze az egyes módszerek időigényét!

5 Atom

5.1 Megszámlálás és kiválogatás

Hozz létre egy N elemű vektort (A) és töltsd fel véletlen számokkal. Ezt követően végezd adj választ az alábbi kérdésekre:

- Van-e az elemek között páros szám (CPU-n és GPU-n)?
- Hány darab páros szám van az elemek között (CPU-n és GPU-n)?
- Hol vannak ezek a páros számok (CPU-n és GPU-n)?
- Az utolsó GPU implementációt próbáljuk gyorsítani shared memóriával!

5.2 Redukció

Hozz létre egy N elemű vektort (A) és töltsd fel véletlen számokkal. Keressük ki a számok közül a legkisebb értékét. Valósítsuk meg az alábbiakat:

- CPU megvalósítás
- GPU megvalósítás 1 szállal
- GPU megvalósítás több szállal, atomi műveletekkel
- GPU megvalósítás több szállal, atomi műveletekkel, shared memóriával gyorsítva
- GPU megvalósítás a tanult gyors redukciós módszerrel egy blokkon belül

- GPU megvalósítás redukcióval, blokkok között atomi műveletekkel
- GPU megvalósítás többszintű redukcióval

6 Stream, event, multiGPU

6.1 Kép feldolgozása részletekben (stream)

Készíts CUDA alkalmazást, amely beolvas egy 8 bites képet, majd csökkenti annak kontrasztját.
Ehhez:

- Fileból kép betöltése, fejléc elhagyása
- Áttöltés GPU memóriába, majd pixelenként a kontraszt csökkentése, és visszamásolás
- Ugyanezt oldjuk meg úgy, hogy a képet darabonként küldjük át a GPU-ra, azonnal megkezdjük ennek feldolgozását, majd visszamásoljuk az eredményt
- Mérjük meg az időkülönbséget az egyes megvalósítások között

6.2 Kép feldolgozása részletekben (stream és event)

Egészítsük ki a fenti alkalmazást egy elmosás lépéssel.

Ehhez:

- Fileból kép betöltése, fejléc elhagyása
- Darabokban áttöltés a GPU memóriába, majd kontraszt csökkentése
- Az egyes darabokon elmosás művelet alkalmazása megadott sugárral
- Ügyeljünk rá, hogy az elmosás csak akkor indulhat meg, ha a szükséges képszeletek előfeldolgozása már megtörtént!

6.3 Kép feldolgozása részletekben (stream és event)

Módosítsuk úgy a fenti alkalmazást, hogy több GPU-n is működőképes legyen.

Ehhez:

- Fileból kép betöltése, fejléc elhagyása
- Darabokban áttöltés a GPU-k memóriába (szükséges esetén alul-felül sáv szabadon hagyása), majd kontraszt csökkentése
- GPU-k közötti szükséges adatcsere megvalósítása
- Elmosás végrehajtása minden GPU-n
- Ügyeljünk rá, hogy a GPU-k között adatcsere, és az elmosás is csak akkor indulhat meg, ha a szükséges képszeletek előfeldolgozása már megtörtént!

7 Optimalizáció

7.1 GPU adatok lekérdezése

7.2 Különböző blokkméretek ellenőrzése

7.3 Memória hozzáférés

7.4 Warp divergencia csökkentés

8 Beépített könyvtárak

8.1 cuBLAS

A CUBLAS könyvtárak segítségével számold ki egy térbeli pont helyzetét az alábbiak szerint:

- Kiindulópont: P(1,2,0)
- Forgatás a CZ(2,3,0) ponton áthaladó Z-vel párhuzamos egyenes körül 180 fokkal
- Forgatás a CX(3,2,0) ponton áthaladó X-el párhuzamos egyenes körül 90 fokkal
- Hol lesz a pont a forgatások után? (3,2,2)
- Segítség:

RotX=

1	0	0
0	cos α	sin α
0	-sin α	cos α

RotY=

cos α	sin α	0
-sin α	cos α	0
0	0	1

8.2 cuRandom

Készíts programot, ami 100x100-as block-kernel méretet használva, minden szál által 100 darab véletlenszámot generálva, megadja az alábbi véletlenszámgenerátorok által adott eloszlásokat (0..99 felbontásban):

- Lineáris
- Normális
- Poisson

8.3 Thrust

Készíts Thrust segítségével egy programot, ami megvalósítja az alábbiakat:

- Generál N darab véletlenszámot, ezeket eltárolja egy vectorban
- Kikersi a legkisebb illetve legnagyobb számot
- Megszámolja, hogy ezen tartományon belül melyik számból hány darab van
- Ezt megjeleníti
- Ugyanezt oldjuk meg CPU-val és GPU-val, hasonlítsuk össze a futásidőket

9 OpenCL

9.1 Platform adatok lekérdezése

Készíts OpenCL alkalmazást, amely lekérdezi az alábbi adatokat:

- Elérhető platformok száma
- Elérhető platformok részletes adatai
- Platformonként az elérhető eszközök listája

9.2 Vektor skalár szorzás

Készíts OpenCL alkalmazást, amely összeszoroz egy 5 elemű vektort egy paraméterként megadott számmal

- Platform, kontextus, parancs sor, program, kernel létrehozása
- Adatok átmásolása
- Kernel indítása
- Adatok visszamásolása

9.3 Kép elmosás

Készíts OpenCL alkalmazást, amely beolvas egy 8 bites képet, majd elmossa annak pixeleit. Ehhez:

- Fileból kép betöltése, fejléc elhagyása
- Pixelek intenzitásának átmásolása a grafikus kártyára
- Minden pixel kicserélése a körülötte K távolságon belül lévő pixelek átlagára
- Adatok visszamásolása, majd kiiírás fileba
- Program kiegészítése lokális memória használatával
    - A globális -> lokális memóriába másolást végezze egy szál
    - A globális -> lokális memóriába másolást végezze minden szál

Hasonlítsd össze a GPU és CPU megvalósítást futásidő szempontjából