Programació Paral·lela

Pràctica 3: CUDA

Memòria de la pràctica, per Johnny Nuñez i Joan Travé

Introducció

La principal motivació d'aquesta pràctica és conèixer l'entorn de programació de CUDA, així com familiaritzar-se amb el mateix. I dit d'una manera més concreta: d'ésser conscient del hardware d'una GPU moderna de la casa NVIDIA i programar un seguit de solucions a un problema proposat, dividit en diferents seccions, cadascuna d'aquestes condicionada per un codi proporcionat i una metodologia indicada.

El problema és el següent: partim d'una imatge en format BRG i l'hem de transformar per a obtenir la mateixa imatge, però en format RGBA.

Cada problema escriu el temps total per realitzar el còmput sobre 1000 iteracions del procés a un fitxer anomenat dades.txt, que després utilitzarem per realitzar una comparativa de temps entre cada secció.

Secció 1

Per aquesta secció se'ns demana completar el codi del kernel per tal que calculi l'índex en ambdós eixos de la posició que el nostre thread aplica a la imatge. Quin píxel és. Per passar de la posició relativa al bloc, a la absoluta al grid, hem de realitzar la següent transformació:

```
int x = threadIdx.x + (blockIdx.x * blockDim.x);
```

Vista a teoria.

Un cop realitzat, compilem i executem i obtenim així uns resultats OK.

Secció 2

Aqui hem de modificar la secció anterior per tal d'usar un grid unidimensional en comptes de bidimensional. Com que tenim un grid ja configurat sense dimensions en l'eix de les y's i de les z's, simplement hem de calcular la posició absoluta del thread amb la fórmula de l'exercici anterior.

Secció 3

Aquesta implementació ha estat cortesia del professor.

La solució d'aquesta secció, on se'ns demanava optimitzar el rendiment del codi utilitzant un registre temporal per llegir el bloc de 16 bytes i després utilitzar el registre en comptes d'accedir a memoria tantes vegades com necessitavem utilitzar el bloc, ha estat millorada llegint de cop 3 blocs de 16 bytes, que contenen en total 4 píxels (Cada bloc té 4 enters i en necessitem 3 per la nostra transformació, i el mínim comú múltiple entre 3 i 4 és 12, per tant 3 blocs) i guardar-lo en un registre temporal, després guardar en més registres temporals cada pixel dels 4 que acabem de llegir (en principi mentre no utilitzem més de 32 registres no hauria de suposar pèrdua de rendiment) i assignar-los a la matriu output RGBA a les posicions corresponents.

Secció 4

L'objectiu d'aquesta secció és iniciar-nos a la memòria compartida. Ara llegirem de la imatge BRG com si fossin elements de 4 bytes i ho colocarem a un vector de memòria compartida. la dificultat d'aquesta part és calcular la posició de cada thread a partir d'aquesta transformació. Per això haurem de modificar la fórmula de la secció primera, per tal de tenir en compte la nova dimensió del bloc.

```
int position = threadIdx.x + (blockIdx.x * N_ELEMS_3_4_TBLOCK);
```

A continuació sincronitzarem threads i llegirem de la memòria compartida. Per això haurem de calcular la posició del thread dins la imatge, aquest cop igual que a la secció primera i escriure el pixel tansformat a la matriu RGBA.

Secció 5

Continuem explorant la matèria de la memòria compartida, aquest cop donant-li una volta més al kernel. Partirem de l'exercici 4, a partir del punt on hem assignat la memòria compartida i havíem sincronitzat threads.

Ara crearem un altre vector de memoria compartida, i per cada thread llegirem tres blocs d'elements, que tal i com hem vist a la secció tercera, equivalen a quatre pixels. En aquest moment assignarem tres blocs a registres temporals, i mentre fem les transformacions per passar de BRG a RGBA assignem els pixels transformats a les quatre posicions consecutives del nou vector de memòria compartida.

A continuació assignem tots 4 pixels directament a les quatre posicions del vector RGBA.

Notem que aquesta secció ha de ser més lenta que la secció anterior, doncs estem accedint dos cops a memoria compartida, mentre que a la secció anterior, només un.

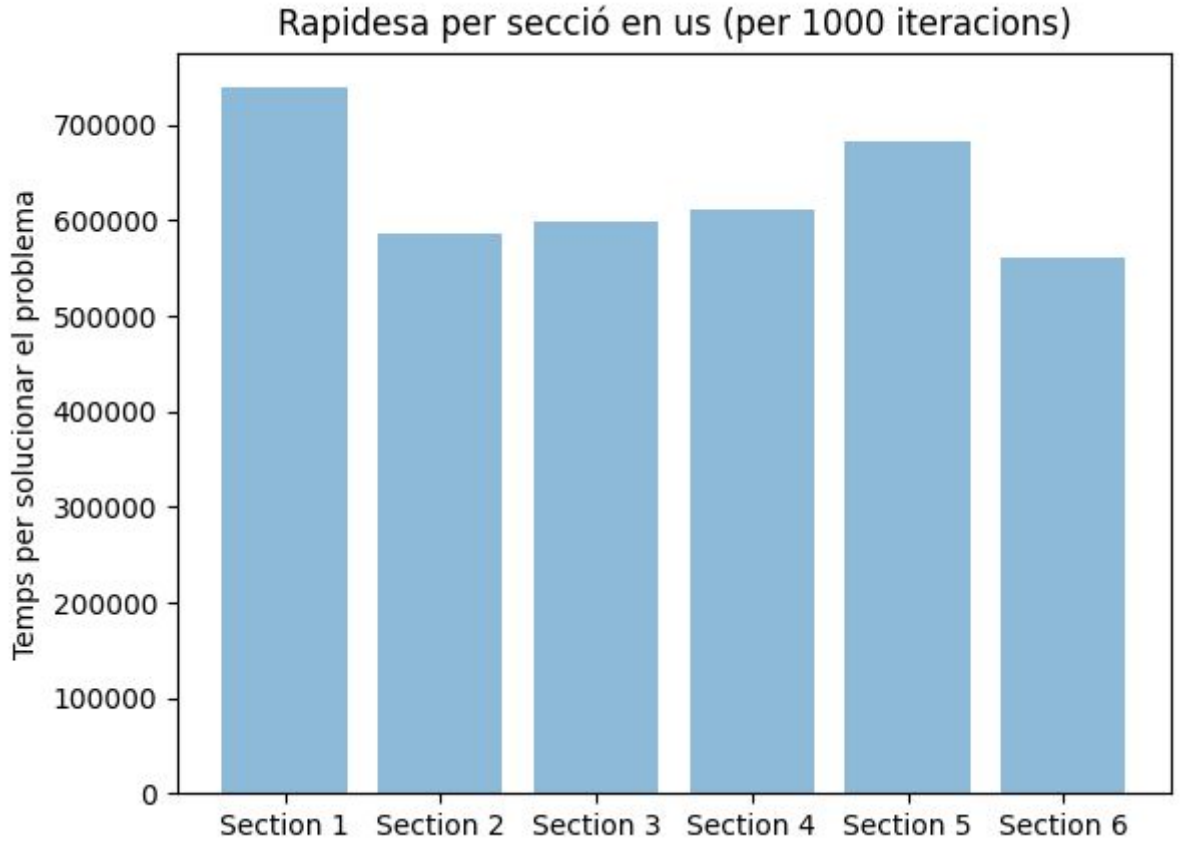
Secció 6

Hem de canviar el codi per a usar streams. Per primera vegada no cal modificar el kernel, doncs usarem el kernel de l'exercici 4. Per a resoldre això ens hem basat en el codi d'exemple, i hem seguit els mateixos passos.

Al final ens adonem que no hem de canviar pas gaire el codi d'exercicis anteriors, excepte afegir les funcions per copiar la memoria de CPU a GPU per a que les crides a memoria de CPU no siguin bloquejants.

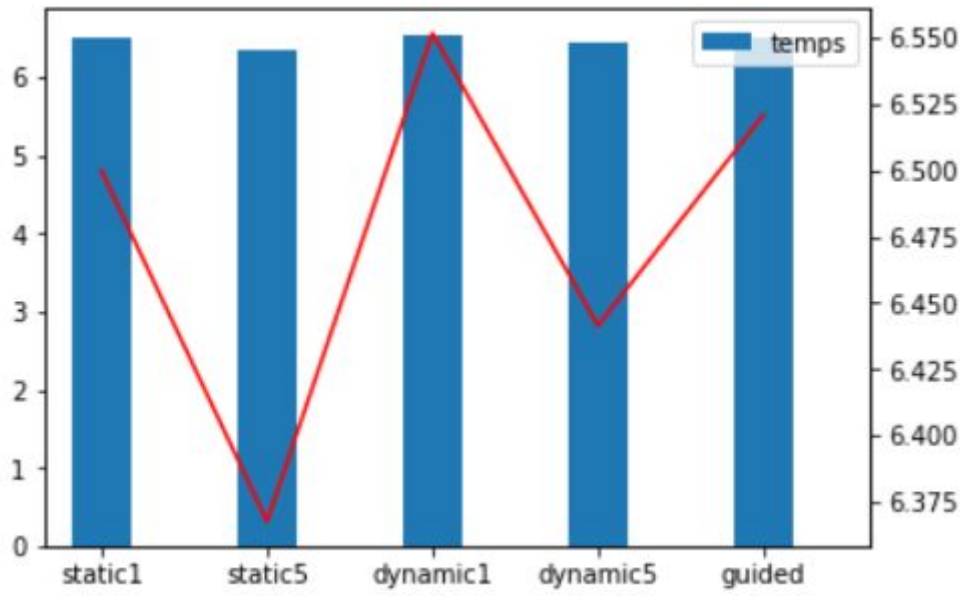
Comparació de resultats

Hem implementat un petit script en python per a llegir del fitxer dades.txt i així poder comparar resultats. El resultat ha sigut:



On podem veure una millora de les seccions 4, 5 i 6 respecte la 1. Això és degut a l'ús de memòria compartida. No veiem, però, una millora respecte la secció 3, ans al contrari, perquè les optimitzacions de la secció tres son exactament les mateixes conceptualment que a la secció 4, però sense haver de sincronitzar threads, la qual cosa sempre serà més ràpida. Al final destacar que de la secció 4 i la 6 hi ha diferència en el temps, i partint de que ambdues usen el mateix kernel, en podem deduir que les millores de streams de la secció 6 han suposat una millora en el rendiment.

Si comparem els resultats amb Open MP (les unitats estan en 10e+6), amb els diferents schedulings ens adonem que sigui quina sigui la comparació, CUDA és sempre de l'ordre d'una desena menys, aproximadament. És a dir, per a aquest cas de paral·lelisme de dades, CUDA és la opció més òptima entre aquestes dues.



Conclusions finals

Podem concloure la pràctica mirant enrere, a la introducció i avaluant el compliment dels objectius de la mateixa. Després d'aquestes setmanes d'implementar codi en CUDA i d'estudiar-ne l'entorn, hem acabant sent capaços de familiaritzar-nos amb el llenguatge i començar a desgranar els inicis del seu funcionament. D'aquesta manera aprofitar-lo per incorporar-lo als possibles problemes de paral·lelisme de dades que ens trobem en un futur.

A més, val a dir que a tots dos integrants de l'equip ens ha interessat especialment el llenguatge i és molt possible que en continuem estudiant en un futur, doncs som conscients que això n'és només una introducció.