# Lab 2

## Programació paral·lela

Oscar Amoros
2019-2020

# Conversió d'espai de color

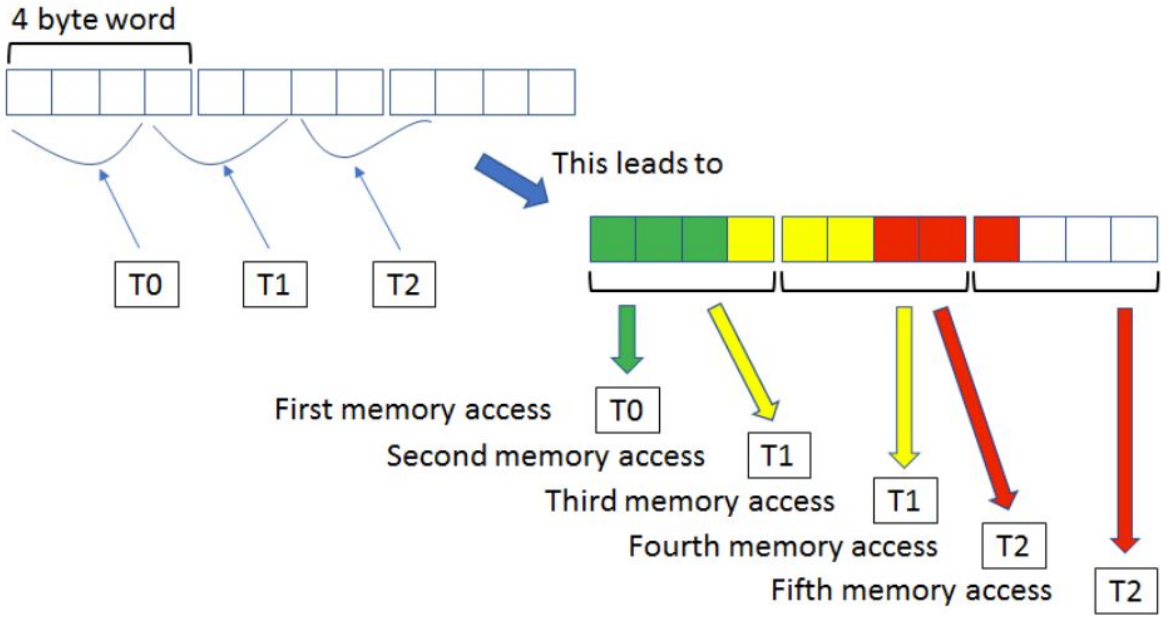
Seguint amb el mateix algorisme que a la pràctica 1, ara anem a implementar una versió en CUDA, i a comprar el rendiment amb les versions en CPU.

El codi proporcionat es incomplet, i l'heu de completar amb el que hem vist a teoria.

Com a l'anterior pràctica, entregueu el codi i documenteu per a cada un dels punts següents:

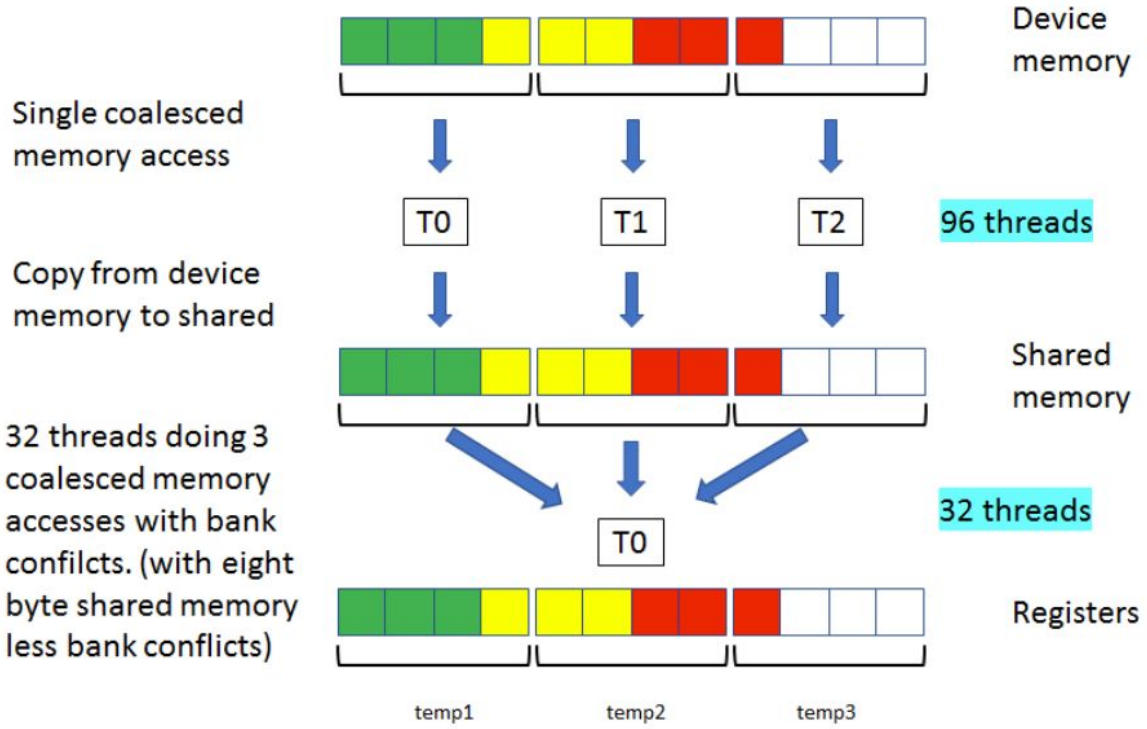
1. Utilitzant els apunts de teoria, completeu el codi que falta. Un cop funcioni:
    a. Mireu el temps que surt per pantalla al executar ./main, i apunteu-lo.
    b. Executeu: nvprof ./main
    c. Obseveu i apunteu el temps per al kernel convertBRG2RGBA
    d. Expliqueu el perquè de les diferències entre un temps i l'altre, seguint el que hem vist a teoria.
2. Implementeu una versió de l'algorisme en CUDA, utilitzant un GRID d'una sola dimensió. És a dir, utilitzant només l'eix de les X. Mesureu el temps i veieu si hi ha alguna diferencia.
3. Partint del punt 2, intenteu optimitzar les accesos a memoria d'alguna manera basica, sense utilitzar shared memory. Comenteu els problemes que hi veieu. Esta la memoria alineada? Recordeu que a diferencia del compilador de CPU, aquí el compilador no optimitza les lectures i les escriptures. Recordeu també, que la memória de la GPU, s'organitza en blocs de 4 bytes.
4. Optimitzeu els accesos a memoria utilitzant shared memory, de manera que sempre puguem llegir i escriure blocks de 4 bytes a la device memory.
5. Intenteu aplicar la optimització explicada a la pàgina següent.
6. Ara, creeu un stream de CUDA, i feu totes les còpies i memset asincrons, utilitzant aquest stream. Utilitzeu aquest stream per al kernel. Compareu els temps totals d'execució amb la comanda: time ./main. Compareu els temps amb i sense stream, modificant també la macro EXPERIMENT_ITERATIONS, fins a 10000 iteracions.
7. Com a la pràctica anterior, creeu un informe, i en aquest cas, compareu el rendiment de la millor versió en serie i la millor versió amb OpenMP, amb les versions que feu amb CUDA.

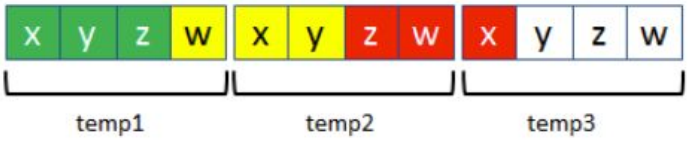
El problema de rendiment d'aquest codi en CUDA és el següent:




Com podeu veure, els threads d'un Warp no estan fent les lectures coalescents, i per tant, el nombre d'accessos a memoria no es un per warp, si no molts més.

A continuació, teniu ilustrada la solució més eficient al problema. La comentarem a classe.

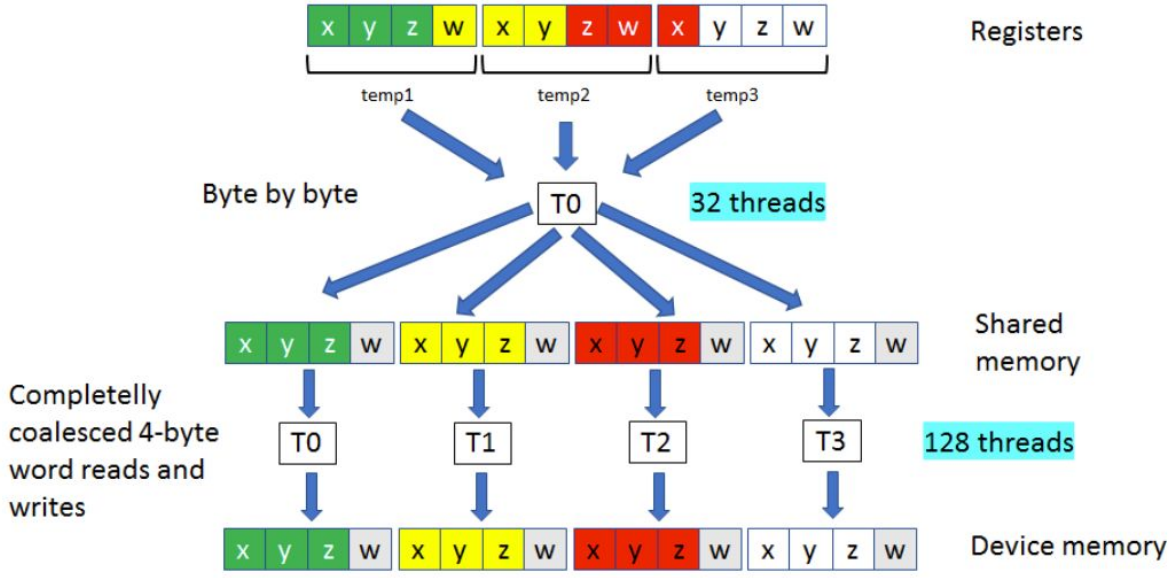





T0

```
pix_write[threadIdx.x * 4]       = make_uchar4(temp1.z, temp1.y, temp1.x, (uchar)0);
pix_write[(threadIdx.x * 4) + 1] = make_uchar4(temp2.y, temp2.x, temp1.w, (uchar)0);
pix_write[(threadIdx.x * 4) + 2] = make_uchar4(temp3.x, temp2.w, temp2.z, (uchar)0);
pix_write[(threadIdx.x * 4) + 3] = make_uchar4(temp3.w, temp3.z, temp3.y, (uchar)0);
```

32 threads generate 128 uchar4 values and write them in shared memory again.




Fixeu-vos que a la part de codi que apareix, l'ordre de les x, y, z i w no son correctes. Les heu de canviar.

La data d'entrega d'aquesta pràctica és el dimecres 13 de Maig.