Universitat de Barcelona

Software Concurrent

Pràctica 1 – OpenCL

Albert Folch i Xavi Moreno

29/03/2015

Descripció

El que volem en aquesta pràctica és paral·lelitzar la multiplicació de matrius per tal d'optimitzar-la en temps d'execució. Gràcies a OpenCL podem executar codi a la GPU i programar cada unitat de processament per a que treballi de forma paral·lela amb les altres.

Per simplificar el problema la multiplicació de matrius la fem amb matrius quadrades, per tant si la entrada són dos matrius de NxN, la sortida serà una matriu de NxN.

Part tècnica

```
#define DBG 0
#define BLOCK_SIZE_H 1
#define BLOCK_SIZE_V 1
#define ALPHA 0.0001
#define MATRIX_SIZE 512
#define DEVICE 1
```

Constants

DBG

Aquesta constant defineix si el codi s'executarà en mode *debug* o no. En cas d'executar-se en mode *debug* (DBG 1) farà la multiplicació en CPU (sense OpenCL) i comprovar si la matriu calculada amb OpenCL és la mateixa que la calculada en CPU. A més mostra els resultats per pantalla. En cas d'executar sense mode *debug* (DBG 0), farà els càlculs amb OpenCL, però no mostrarà cap resultat, excepte el temps que ha tardat.

BLOCK_SIZE_*

Defineix la mida que pren en horitzontal i vertical cada *work-group*, és a dir, el número de *work-items* que té cada *work-group*.

ALPHA

Alpha s'utilitza per comparar la matriu calculada amb OpenCL i amb CPU amb codi C.

MATRIX_SIZE

Aquest paràmetre ens indica la dimensió de les matrius d'entrada

DEVICE

Defineix el dispositiu amb el qual s'executarà el codi d'OpenCL.

- 0 -> CPU
- 1 -> GPU

Inicialització del software i del hardware

```
/* Inicialitzar hardware i software */
found=0;
hardware = sclGetAllHardware(&found);
software = sclGetCLSoftware("matmul_kernel.cl","MatMulKernel",hardware[DEVICE]);
```

Calcular el temps d'execució

```
cl_ulong time = sclGetEventTime(hardware[DEVICE], event);
printf("\nAmb OpenCL ha tardat: %f segons\n", time/1000000000.f);
```

Proves

Totes les proves fetes per mostrar el rendiment de cada exercici s'ha fet tant en CPU com en GPU, però ambdós utilitzant OpenCL. Les marques de temps de les taules es mostren en segons. Les proves de rendiment de la multiplicació de matrius a nivell de CPU en codi C no les hem contemplat.

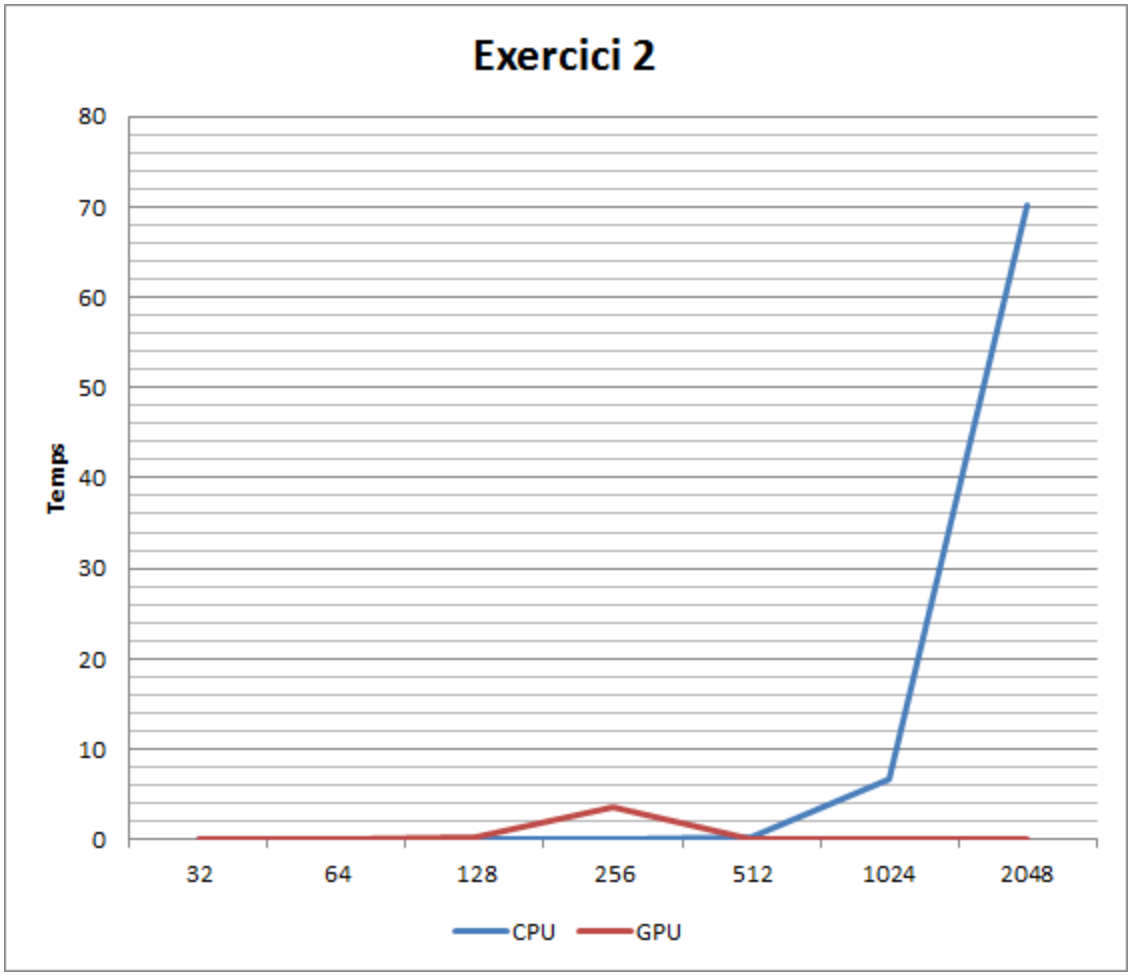
Exercicis

Exercici 2

Aquí ens demanaven que provéssim d'executar la multiplicació de matrius en un sol *work-item*.

dimensió	cpu	gpu
32	0.000083	0.002144
64	0.00035	0.016531
128	0.006322	0.16788
256	0.034149	3.65874
512	0.241054	
1024	6.576024	
2048	70.116531	

A partir de la mida de matriu 512 inclosa, en la GPU falla, suposem que és perquè carreguem massa la càrrega de un *work-item* Al gràfic es mostra com si fos 0 en aquest interval.



Exercici 3

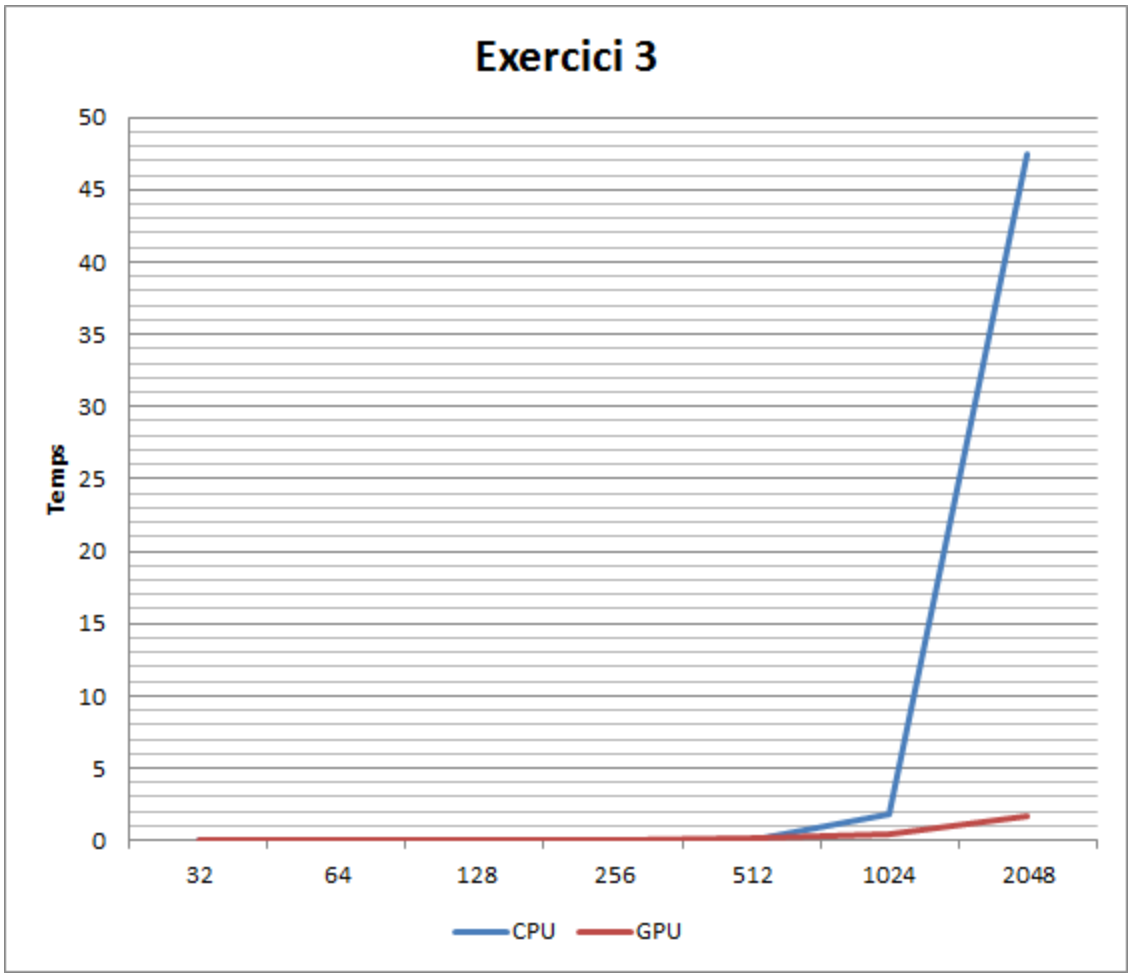
En aquest exercici ens demanaven que utilitzéssim tants *work-items* com files o columnes tingués les matrius d'entrada. Per tal d'optimitzar al màxim els resultats, el número de work-items per work-group seguirà la següent fórmula:

$$min\ (\ Matrix\ Size,\ Block\ Size\ H)$$

on *Block_Size_H* és el número de work-items que té un *work-group* en les GPUs dels ordinadors de l'aula IA que són 128 (32x4). Amb aquesta fórmula aconseguim que el màxim de work-items dels work-groups estiguin ocupats.

Cada work-item calcularà una columna de la matriu C. Mitjançant 2 bucles, calcularà el resultat final d'una posició recorrent la fila i-éssima de A i la columna amb el seu mateix id global. Quan hagi acabat de calcular la posició, anirà a la següent fila de la mateixa columna i tornarà a calcular-ho amb la corresponent fila de A. Finalment, es farà una barrera de la memòria global per a que no es comenci a transmetre la matriu C al host fins que no s'hagin acabat els càlculs.

dimensió	cpu	gpu
32	0.000099	0.000087
64	0.00072	0.000331
128	0.006394	0.003817
256	0.017512	0.015423
512	0.064104	0.097336
1024	1.84811	0.39201
2048	47.501163	1.629827

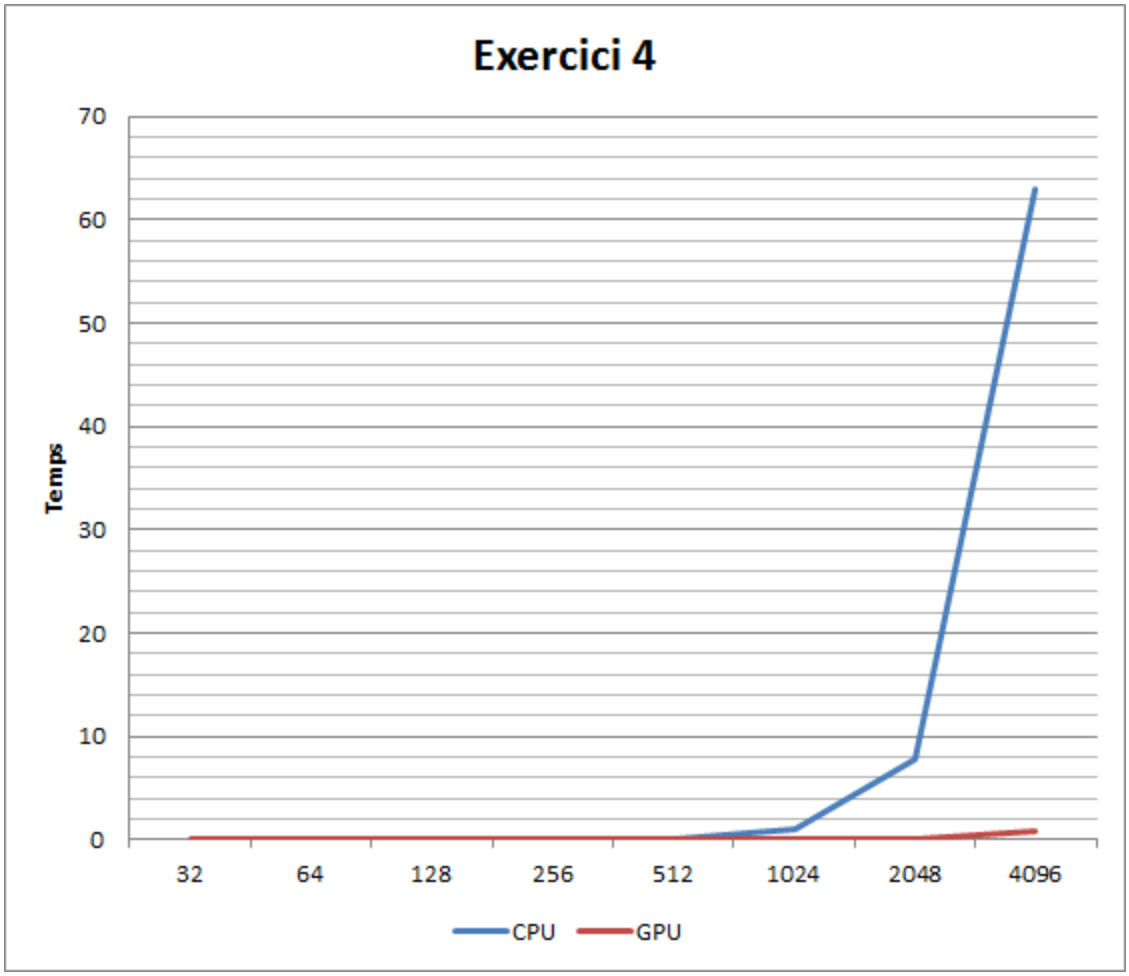


Exercici 4

Aquí fem servir un total de *MATRIX_SIZE* work-items en els 2 eixos. El número de work-items per work-group pot variar sempre que sigui igual en els 2 eixos. Per les proves hem utilitzat 16x16 work-items per work-group, ja que és la mida més òptima.

Per optimitzacions, i perquè ho demana a l'enunciat, fem servir memòria local per estalviar accessos de memòria a la memòria global ja que és molt lent. Hi guardarem espai pels 2 blocs que farem servir per multiplicar. Una manera de carregar les dades del bloc per multiplicar seria que un work-item s'encarregués de fer-ho, seria una manera vàlida però no la òptima. Això és el que havíem fet al principi, en canvi nosaltres al final hem fet que cada work-item s'encarregui d'agafar el valor corresponent d'A i de B i de posar-lo als blocs. Fem servir una barrera just després per a que es faci la multiplicació per guardar a C assegurant-nos que els blocs s'han carregat. Després de que el work-item corresponent hagi calculat la seva posició de C recorrent els 2 blocs, farem una altra barrera de la memòria local per a esperar a que s'hagin utilitzat els blocs abans de modificar-los de nou en la següent iteració del bucle. Al acabar el bucle, fem una barrera per asegurar-nos que C s'ha acabat de calcular abans de que es comenci a carregar al host.

dimensió	cpu	gpu
32	0.000151	0.000016
64	0.000301	0.00002
128	0.003192	0.000047
256	0.015358	0.000249
512	0.124984	0.001789
1024	0.973841	0.014021
2048	7.85764	0.111558
4096	62.860981	0.893125



Exercici 5

Aquest exercici no l'hem pogut implementar a falta de temps, però si ho haguéssim fet, notaríem un millor rendiment al executar la multiplicació de matrius en la GPU dels ordinadors de l'aula IA.

En aquest exercici ens basem que les *comput units* de les GPUs de l'aula IA, tenen 32x4 unitats de processament. Per tant la mida òptima de *work-items* per *work-group* ha de ser 32x4. Tenint en compte aquesta mida, s'hauria de tocar el codi de l'exercici 4 ja que no accepta blocs que no siguin quadrats.

De la mateixa manera que en l'exercici 4 anem desplaçant els blocs en una sola direcció, ara el que hauríem de fer es moure el blocs 8 cops en vertical, d'aquesta manera ho podríem mirar com si tinguéssim blocs de 32x32, i un cop hem desplaçat el bloc 8 cops en vertical, podem fer el moviment que toca per a cada matriu.

D'aquesta manera podríem aprofitar al màxim els *work-items* de cada *comput unit* de les GPUs de l'aula IA, i així millorar el rendiment.

Conclusions

En conclusió, hem pogut comprovar que per exemple en l'exercici 2 el mateix codi era més ràpid en la CPU que no pas en la GPU. Això és degut a que 1 *core* de la CPU treballa a més freqüència i està més optimitzat que 1 sol work-item de la GPU.

En l'exercici 3, des del principi fins que *MATRIX_SIZE* és 1024 a vegades triga més la CPU i a vegades més la GPU. Però si la matriu augmenta més la diferència es multiplica mostrant que la GPU calcula més ràpidament.

En l'exercici 4, abans d'acabar d'optimitzar que teniem programat que la càrrega dels blocs la fes 1 work-item amb *MATRIX_SIZE* de 4096 trigava 10 segons. Després, al canviar-ho a que ho fessin tots els work-items en paral·lel tarda 0.89 segons.

Finalment, ens hem adonat que l'execució del codi OpenCL a la GPU depèn molt d'aquesta, ja que cada GPU és diferent de l'altre.