Software Concurrent

OpenCL: Multiplicació de Matrius

Vicent Roig, Igor Dzinka

29/03/2015

Índex

2. Objectiu

La multiplicació de matrius es una operació matemàtica bàsica, que conté força paral·lelisme de dades, però també certs problemes de localitat de dades. En aquesta pràctica es demana implementar el codi en OpenCL per a la multiplicació de dues matrius A i B, i retornar una matriu C amb el resultat de la multiplicació.

Per a simplificar, A i B tindran la mateixa mida per a les dimensions "x" i "y". L'objectiu passa per la implementació OpenCL i posterior comparació a les diverses plataformes i dispositius amb suport (Devices) a més de contrastar el conjunt amb el codi C sense paral·lelitzar, per tal de comprovar que els resultats en OpenCL són correctes

El codi de Host ha set implementat amb **SimpleOpenCL**. El codi de Device, és a dir, el **Kernel**, ha set implementat amb OpenCL C.

1. Introducció

La codificació de la pràctica ha estat distribuïda en una estructura jeràrquica que segueix els passos a implementar. Les dades referents als temps d'execució per als diversos Devices han set recollides en fitxers de log a cada un d'aquests directoris (anomenats 'results.log'). S'ha extret el src del projecte SimpleOpenCL al directori arrel per tal de generalitzar la compilació i no duplicar codi al repositori, en conseqüència s'han modificat les directrius dels Makefile corresponents a cada part. A més es facilita l'output de clinfo a 'clinfo.log' per tal d'obtenir més detalls.

Totes les proves d'execució han set testejades a diversos dispositius. Tant als ordinadors de l'aula IA com a un portàtil amb Linux y 2 Platform amb suport pels 2 devices (Intel HD4600 Graphics Card & Intel core i3 3217U) gràcies al suport del projecte openSource **Beignet OpenCL** que ens ofereix suport per les GPU integrades "on-die" a les CPU de les series Core i Xeon.

Finalment hem trobat més interessant poder fer proves a un altre sobretaula amb gràfica dedicada, per contrastar diverses arquitectures. En aquest cas una màquina amb CPU Intel i7 4770k 3.5Ghz (overclok a 4.2GHz) i GPU AMD Ati R9 280x (940Mhz / 3GB DDR5. Hem treballat sobre un repositori propi, disponible pel controlador de versions GIT a **BitBucket** respectant les llicencies heretades de SimpleOpenCL i GPL:

- **git clone https://simorgh12@bitbucket.org/simorgh12/sc2015.git**

Donat la gran quantitat de gràfics i taules generades a les diverses plataformes hem decidit incloure a aquest document les corresponents únicament a l'arquitectura ATI Radeon + i7 4770k. En aquest cas, les proves han set realizades sobre llibreries UNIX **Cygwin** a una arquitextura WIN64 amb suport OpenCL (2 platforms) Intel i **AMD OpenCL™ Accelerated Parallel Processing**. Diexem aquí un breu resum del sistema retornat per SimpleOpenCL:

```
PLATFORM/DEVICES GENERAL:
============================================================
Group 1 with 1 devices
Group 2 with 1 devices
Group 3 with 1 devices
 Device 0
 Platform name: Intel(R) OpenCL
 Vendor: Intel(R) Corporation
 Device name: Intel(R) Core(TM) i7-4770K CPU @ 3.50GHz
 Device 1
 Platform name: AMD Accelerated Parallel Processing
```

```
Vendor: Advanced Micro Devices, Inc.
Device name: Tahiti
Device 2
Platform name: AMD Accelerated Parallel Processing
Vendor: Advanced Micro Devices, Inc.
Device name: Intel(R) Core(TM) i7-4770K CPU @ 3.50GHz
```

(*) detalls poden ser consultats al fitxer clinfo.log.

Cal a dir, com podem observar, que el Device 2 és el mateix component hardware que el 0, la variació ve donada per la plataforma que estan utilitzant. Com bé hem vist durant les classes magistrals, a un mateix sistema poden conviure diverses Platform OpenCL, i cadascuna d'aquestes plataformes poden o donar suport al mateix dispositiu, esdevenint Devices diferents. Aquest és el cas de la CPU, s'han realitzat les mateixes proves als Devices 0 i 2 per poder comparar les dues plataformes.

3. Descripció detallada de la implementació

3.1 Estructuració del projecte:

La estructuració del projecte, els directoris part1, part2, part3, part4 corresponen a:

- **/part1**:

  Corresponent al punt 2 de l'enunciat de la pràctica. Creació d'un kernel OpenCL, que utilitzant un sol Work Item, implementi el mateix algorisme que el codi C estàndard. (*)

- **/part2**:

  Corresponent al punt 3 de l'enunciat de la pràctica. Modificació per a que el kernel utilitzi, tants WorkItems com elements hi ha l'eix de les "x". En aquest cas no es fan servir variables de tipus __local, només __global. Proves d'execució del kernel d'OpenCL tant a la CPU com a la GPU. (*)

- **/part3**:

  [Opcional]. Partint del punt 3 de la pràctica, modificació per a que el kernel utilitzi tants WorkItems com elements a la matriu . No es fan servir variables de tipus __local, només __global. (*)

- **/part4**:

  Corresponent al punt 4 de l'enunciat de la pràctica. Acceleració de l'execució, ajustant el numero de Work Items i fent servir variables de tipus __local en comptes de __global, per reduir el nombre d'accessos a la memòria __global. Tants Work Items com dades a la matriu C tant a la dimensió de les x com les y. Work Groups de 16x16 Work Items. (*)

** La comparació del rendiment s'il·lustrarà i discutirà més endavant (apartat **4.Resultats de funcionament i discussió**).*

3.2 Paral·lelisme sense memòria local:

Com s'ha comentat a la introducció aquesta pràctica està dedicada al processament de multiplicació de matrius utilitzant la GPU.

La part més interessant resideix en desenvolupar dues implementacions conceptualment divergents pel que respecta a la compartició de memòria i temps d'accés a recursos compartits. Una implementació multiplica dues matrius sense utilitzar les variables __local, i en contraposició una d'altra agilitzarà accessos fent ús de còpies de variables __local per tal d'accelerar-ho. D'aquesta manera podrem comparar el rendiment de sincronització local front el rendiment del processament dels WorkItems.

La idea de fons, és que si aconseguim estalviar accessos reiteratius a la mateixa posició de memòria i aconseguim portar a nivell local un problema més petit en forma de submatrius (BA, BB), aquestes còpies romandran accessibles per tots els WI estalviant cicles de rellotge dintre d'un WorkGroup, molt valuosos.

3.3 Paral·lelisme amb memòria local:

Aquest és el punt principal d'aquest estudi, ja que mostra com prendre avantatge dels grups de treball mitjançant l'ús de barreres i les variables __local. El que volem en aquest moment és dividir el problema de la multiplicació de matrius en submatrius que puguin cabre en la grandària de grup de treball local.

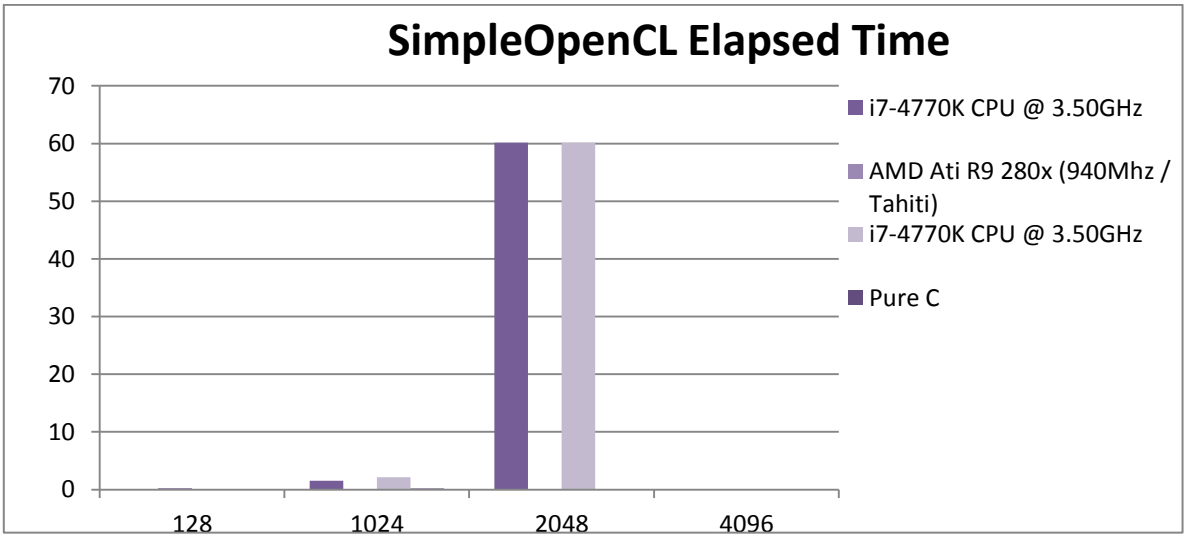
Hem aconseguit dividir el problema en un de submatrius 16×16, on podrem calcular cada submatriu resultat com la suma dels productes de les primeres, com si les submatrius fossin realment elements primitius a computar.

Això però, ens presenta altres inconvenients, cal assegurar que les còpies locals s'han acabat de realitzar abans de continuar, això ho podem implementar mitjançant l'ús d'una barrera. Donat que necessitem sincronització a memòria __local, el tipus de sincronització utilitzada és la definida com 'CLK_LOCAL_MEM_FENCE'.

4.Resultats de funcionament i discussió

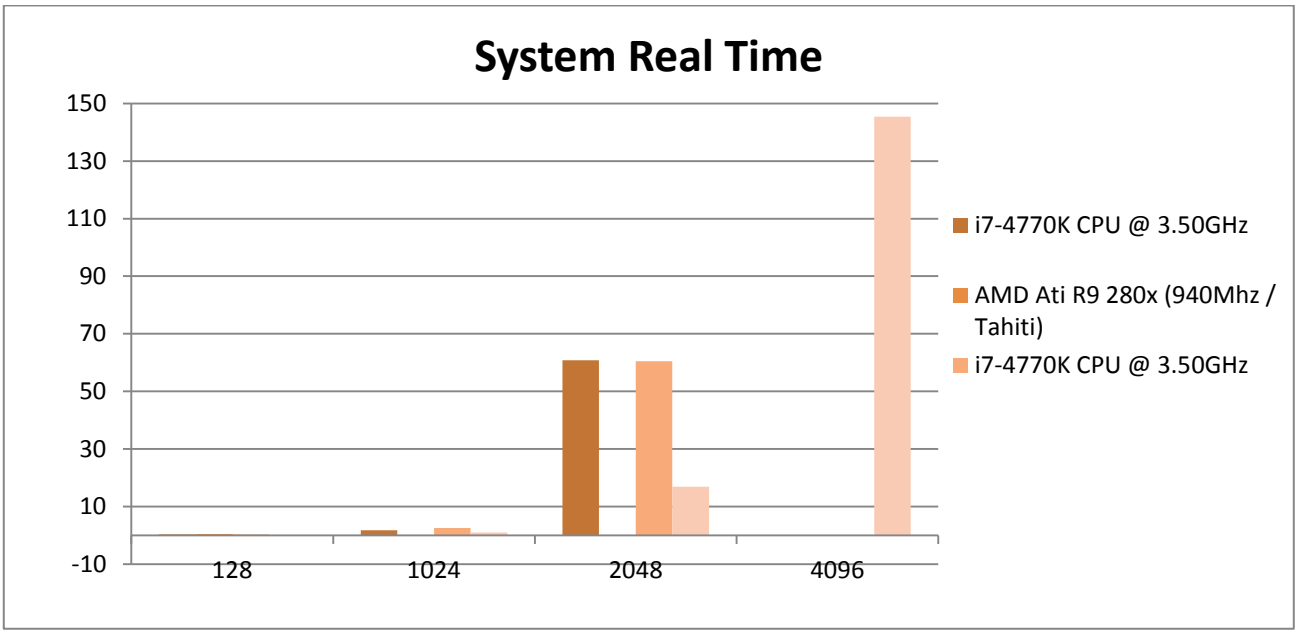
Part 2: 1 workitem per calcular tota la matriu.

Quan fem servir un sol workitem no es produeix cap paral·lelització a l'hora de realitzar el càlcul. La capacitat de un workitem de la unitat gràfica es inferior a la capacitat de la CPU. En el nostre cas, quan hem intentat calcular el producte de matrius de mida 1024 o superior, a la GPU, l'ordinador no ha sigut capaç d'acabar el càlcul, resultant en errors gràfics o temps d'execució massa llargs.



Elapsed Time

Workgroup	i7-4770K CPU @ 3.50GHz	AMD Ati R9 280x (940Mhz / Tahiti)	i7-4770K CPU @ 3.50GHz	Pure C
128	0,0023	0,2810	0,0016	0,0000
1024	1,5279	0,0000	2,1600	0,1870
2048	60,1661	NA	60,1290	0,0000
4096	NA	NA	NA	0,0150



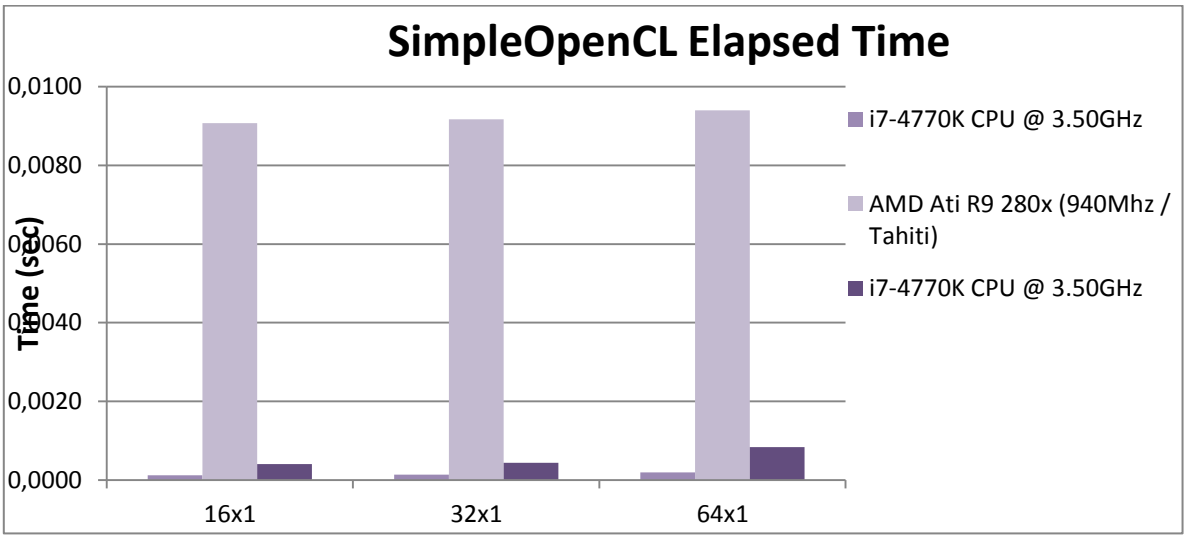
Real time

workgroup	i7-4770K CPU @ 3.50GHz	AMD Ati R9 280x (940Mhz / Tahiti)	i7-4770K CPU @ 3.50GHz	Pure C
128	0,3010	0,5850	0,3150	0,0040
1024	1,8070	NA	2,5490	1,0530
2048	60,8190	NA	60,4700	16,8980
4096	NA	NA	NA	145,3740

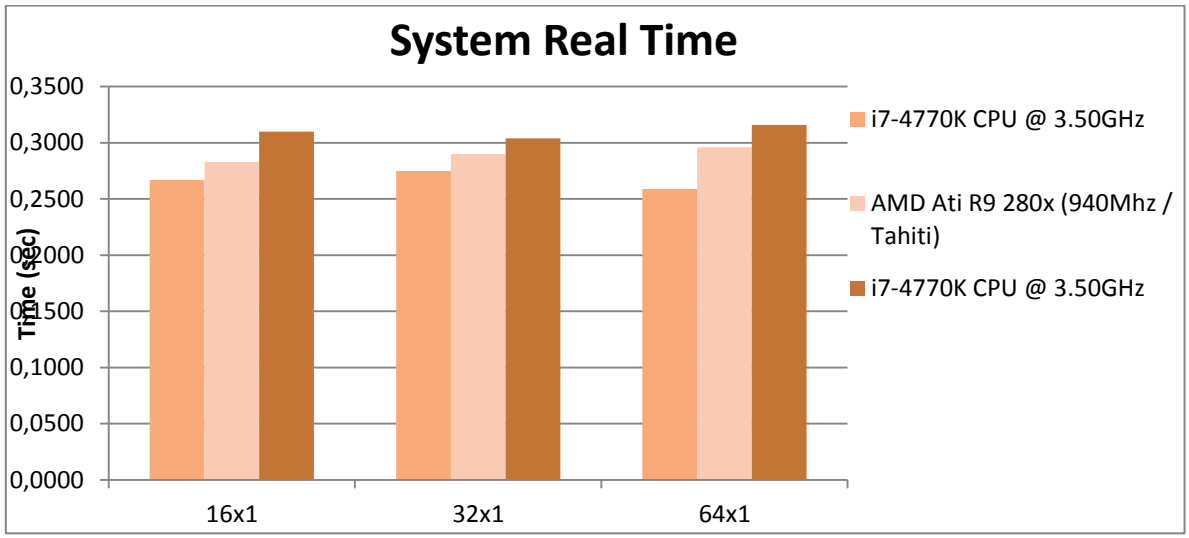
Part 3: Tants workitems com elements a la fila

En aquesta part per primer cop apliquem una paral·lelització a l'hora de realitzar els càlculs. A continuació posem els resultats del les proves amb diferents mides de les matrius. Com podem apreciar a les gràfiques dels *Elapsed time,* que representen els temps d'execució mesurats amb la funció de **SimpleOpenCL *sclGetEventTime***, quan mes gran és el volum de dades a calcular, es destaca més l'eficiència de càlcul a la unitat gràfica. Cal destacar que a aquesta part de la practica no utilitzem encara les variables __local, per tant no estem realitzant els càlculs de la forma mes eficient possible, ja que estem realitzant moltes lectures de memòria __global, cosa que fa que es ralentitzi el càlcul.

128x128

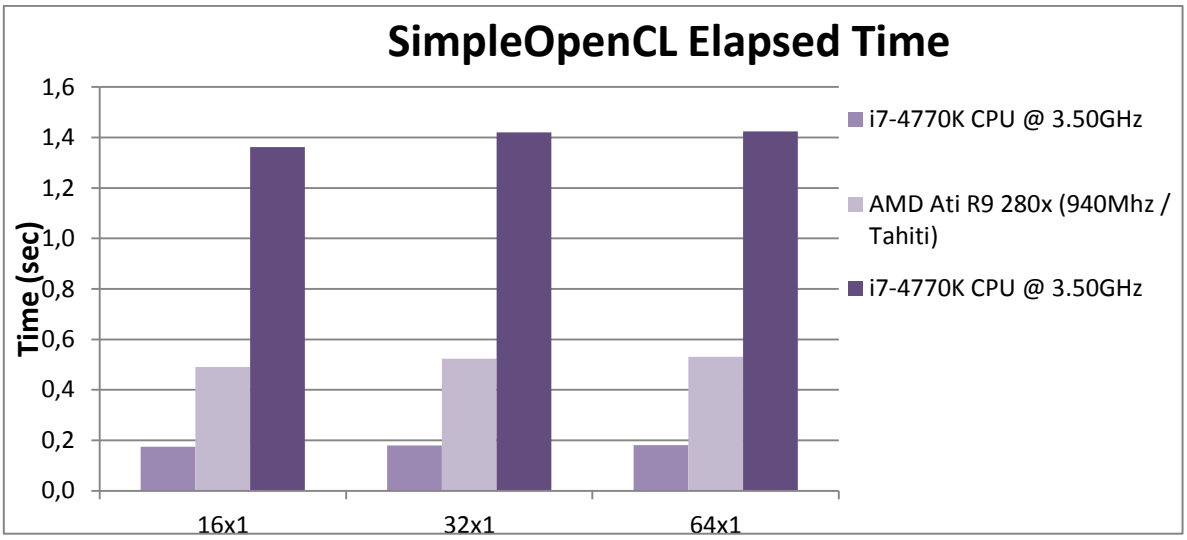


workgroup	i7-4770K CPU @ 3.50GHz	AMD Ati R9 280x (940Mhz / Tahiti)	i7-4770K CPU @ 3.50GHz
16x1	0,0001	0,0091	0,0004
32x1	0,0001	0,0092	0,0004
64x1	0,0002	0,0094	0,0008

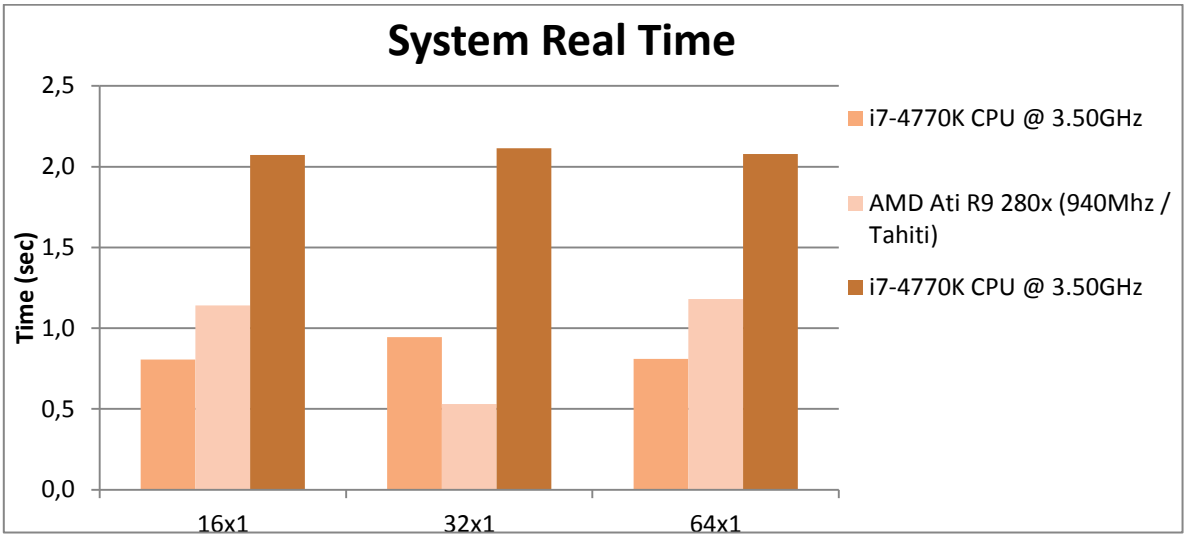


workgroup	i7-4770K CPU @ 3.50GHz	AMD Ati R9 280x (940Mhz / Tahiti)	i7-4770K CPU @ 3.50GHz
16x1	0,2670	0,2830	0,3100
32x1	0,2750	0,2900	0,3040
64x1	0,2590	0,2960	0,3160

1024x1024

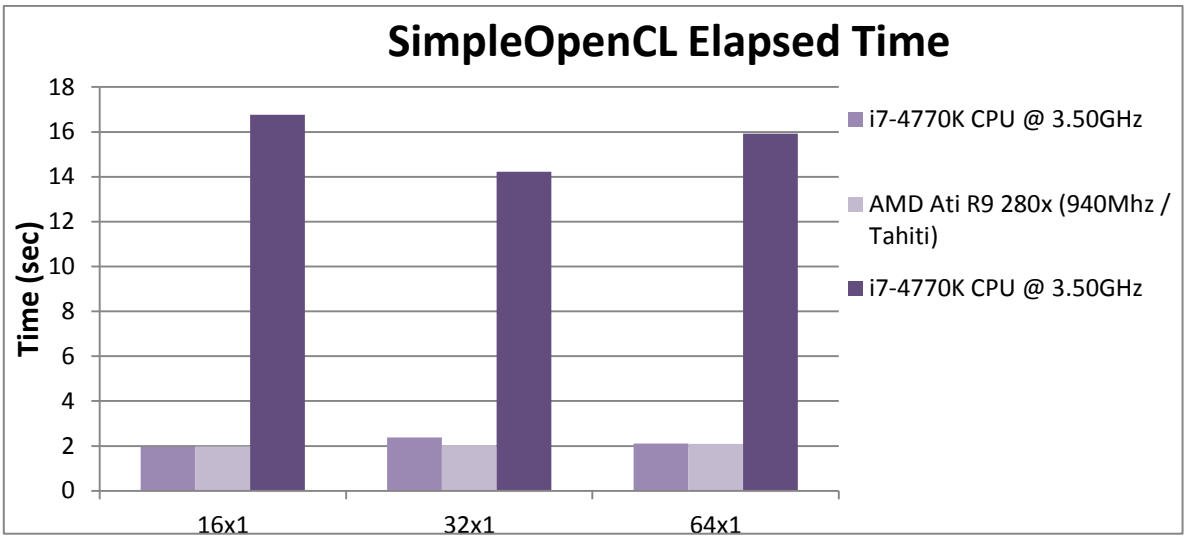


workgroup	i7-4770K CPU @ 3.50GHz	AMD Ati R9 280x (940Mhz / Tahiti)	i7-4770K CPU @ 3.50GHz
16x1	0,1741	0,4902	1,3617
32x1	0,1798	0,5238	1,4203
64x1	0,1806	0,5309	1,4237

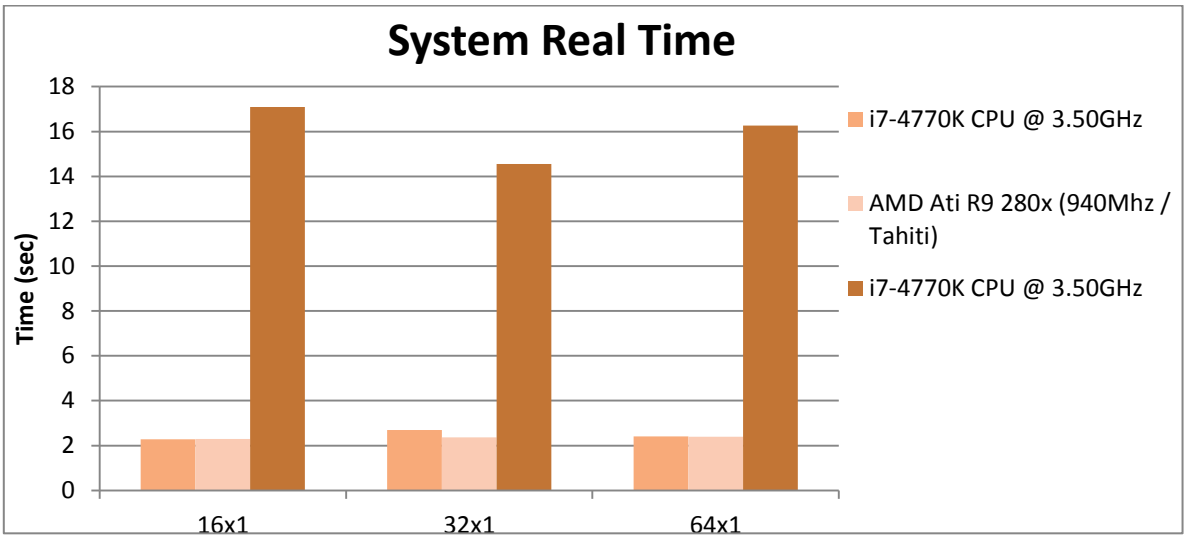


workgroup	i7-4770K CPU @ 3.50GHz	AMD Ati R9 280x (940Mhz / Tahiti)	i7-4770K CPU @ 3.50GHz
16x1	0,8060	1,1410	2,0730
32x1	0,9440	0,5310	2,1140
64x1	0,8110	1,1810	2,0790

2048x2048

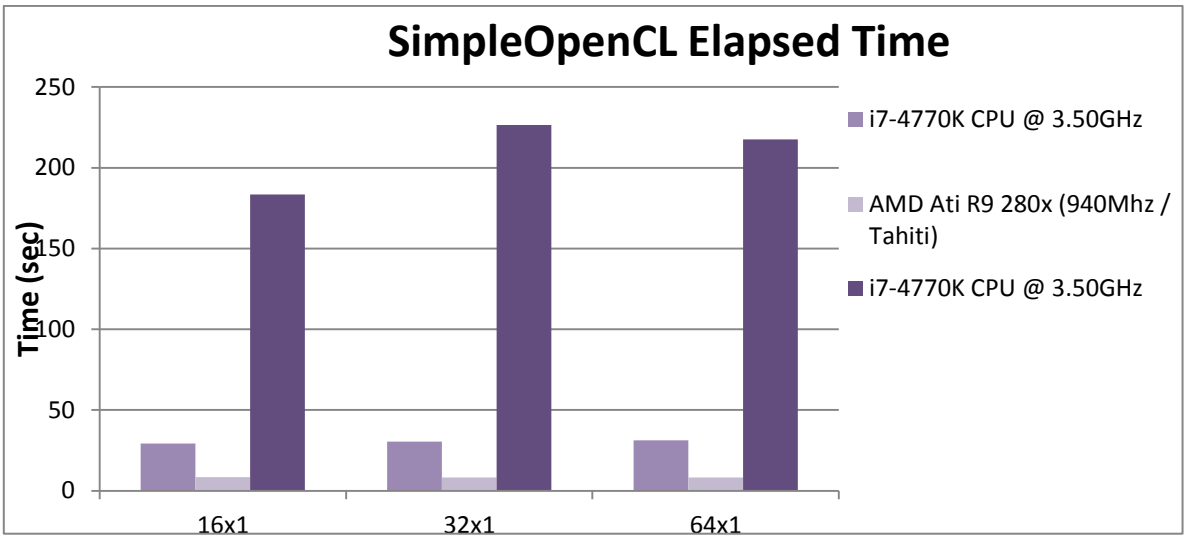


workgroup	i7-4770K CPU @ 3.50GHz	AMD Ati R9 280x (940Mhz / Tahiti)	i7-4770K CPU @ 3.50GHz
16x1	1,9699	1,9827	16,7712
32x1	2,3776	2,0347	14,2287
64x1	2,1015	2,0866	15,9253

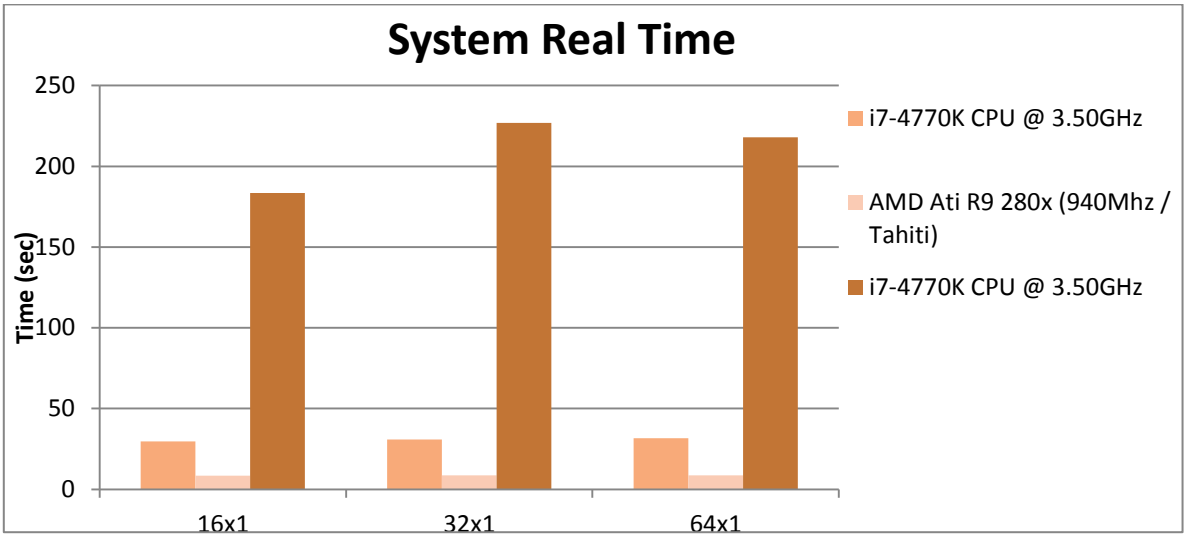


workgroup	i7-4770K CPU @ 3.50GHz	AMD Ati R9 280x (940Mhz / Tahiti)	i7-4770K CPU @ 3.50GHz
16x1	2,2740	2,3000	17,0970
32x1	2,6870	2,3660	14,5590
64x1	2,4140	2,3900	16,2660

4096x4096



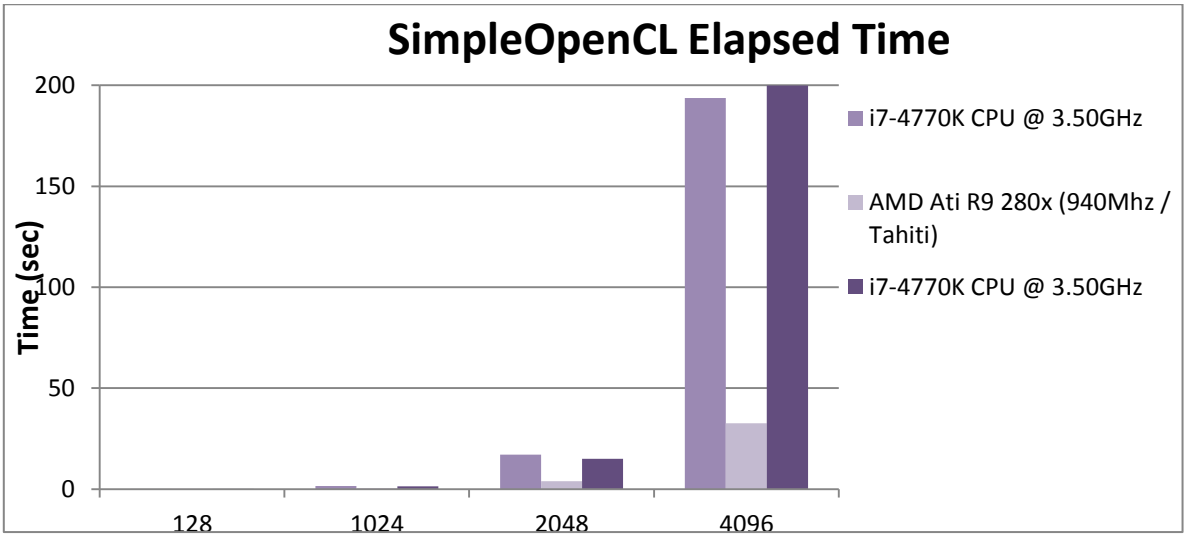
workgroup	i7-4770K CPU @ 3.50GHz	AMD Ati R9 280x (940Mhz / Tahiti)	i7-4770K CPU @ 3.50GHz
16x1	29,2453	8,4462	183,5730
32x1	30,3803	8,2261	226,4029
64x1	31,2456	8,2434	217,6437



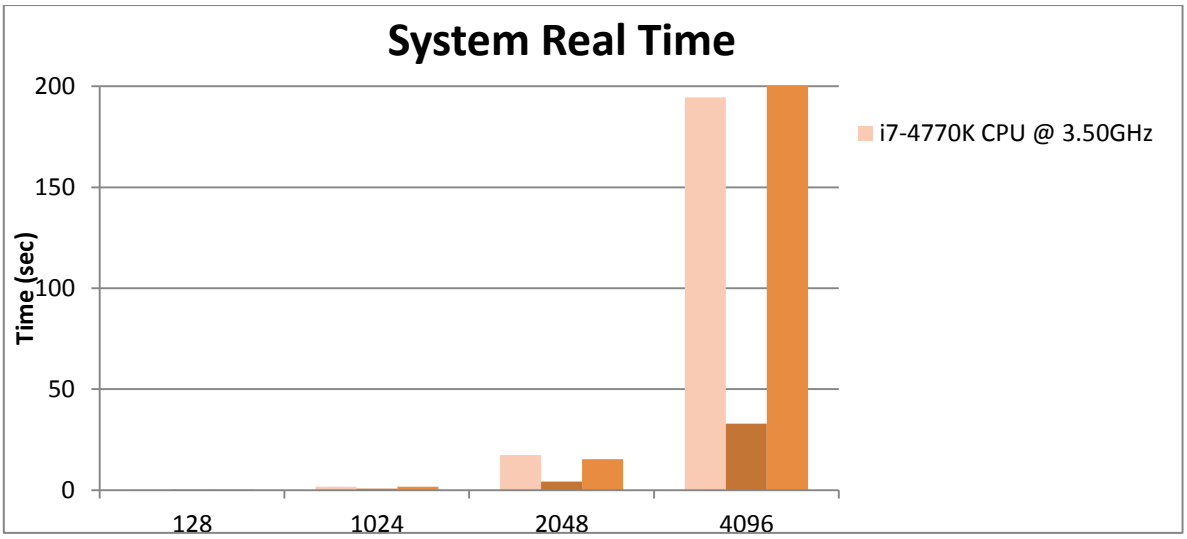
workgroup	i7-4770K CPU @ 3.50GHz	AMD Ati R9 280x (940Mhz / Tahiti)	i7-4770K CPU @ 3.50GHz
16x1	29,6870	8,4462	183,5729
32x1	30,8300	8,5900	226,8200
64x1	31,7000	8,5860	218,0310

Extra: Càlcul, fent servir tants workitems com elements a la matriu

En aquest apartat hem intentat realitzar el càlcul repartint cada element de la matriu a un workitem. Teòricament amb aquesta aproximació s'hauria d'obtenir el resultat mes òptim. Però, a la pràctica veiem que no es així. Es deu a que el Device en qüestió disposa d'un nombre determinat de fils d'execució. A l'hora de paralelitzar, hem de tenir en compte aquesta limitació. Quan estem quan el nombre de workitems assignats esdevé massa gran, es perd el factor de coalescència i l'execució es ralentitza.



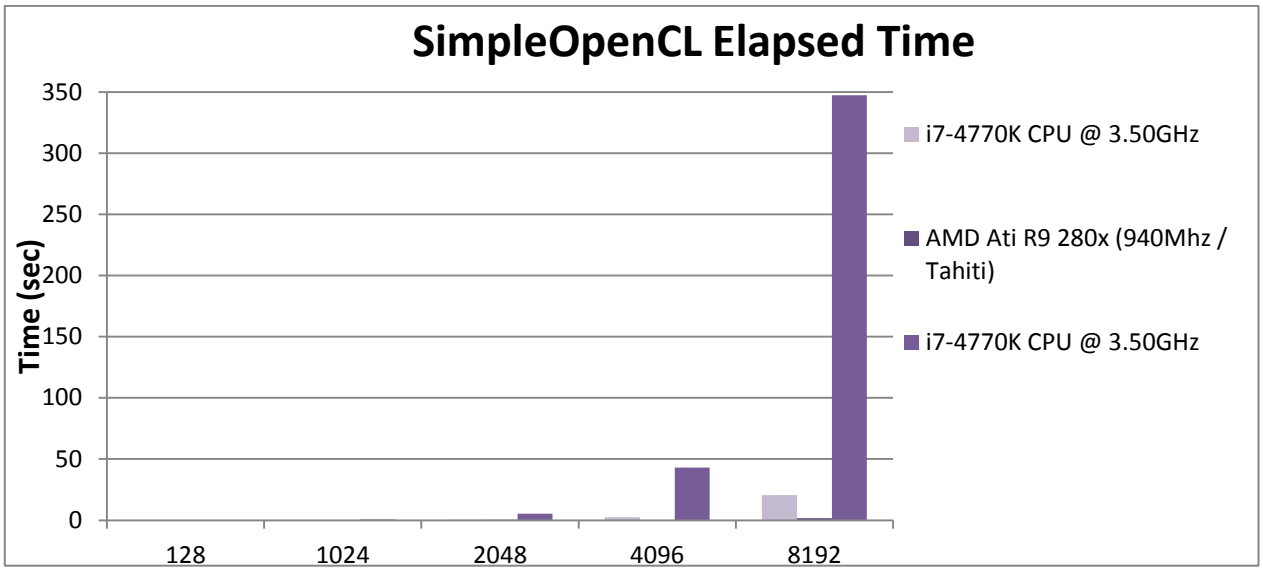
workgroup	i7-4770K CPU @ 3.50GHz	AMD Ati R9 280x (940Mhz / Tahiti)	i7-4770K CPU @ 3.50GHz
128	0,0005	0,0014	0,0006
1024	1,4597	0,4884	1,4388
2048	17,0751	3,9810	15,0019
4096	193,6963	32,5435	199,8518



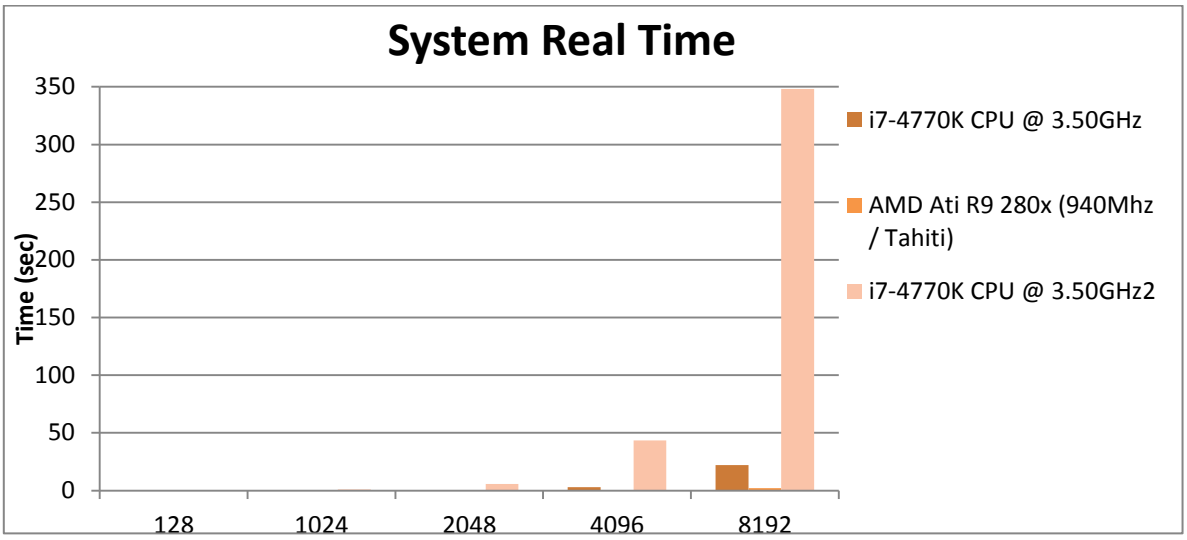
workgroup	i7-4770K CPU @ 3.50GHz	AMD Ati R9 280x (940Mhz / Tahiti)	i7-4770K CPU @ 3.50GHz
128	0,2730	0,2980	0,3090
1024	1,7320	0,7920	1,7440
2048	17,4100	4,2880	15,3390
4096	194,4930	32,9120	200,2550

Part 4: Acceleració utilitzant les variables de tipus __local

Per últim posem els resultats del càlcul paral·lelitzat fent servir, a més les variables __local. Comparant el resultats d'execució amb variables __local respecte a execució amb __globals veiem que la millora d'eficiència es molt notable: Per a matrius de 4096x4096 aconseguim un factor x44 d'augment de rendiment.



Matrix size	i7-4770K CPU @ 3.50GHz	AMD Ati R9 280x (940Mhz / Tahiti)	i7-4770K CPU @ 3.50GHz
128	0,0001	0,0000	0,0014
1024	0,0255	0,0057	0,6548
2048	0,2775	0,2822	5,3596
4096	2,4056	0,1936	43,1180
8192	20,6285	1,5942	347,4320



Matrix size	i7-4770K CPU @ 3.50GHz	AMD Ati R9 280x (940Mhz / Tahiti)	i7-4770K CPU @ 3.50GHz
128	0,0001	0,0000	0,0014
1024	0,0255	0,0057	0,6548
2048	0,2775	0,2822	5,3596
4096	2,4056	0,1936	43,1180
8192	20,6285	1,5942	347,4320

5. Conclusions

Al llarg d'aquesta pràctica hem sigut capaços d'entendre els conceptes de paral·lelisme i coalescència gracies als mecanismes que ens facilita OpenCL per explotar la concurrència que ens brinda la pròpia arquitectura de les GPU's.

Els conceptes clau per a la implementació del producte han set resumides al material docent facilitat a l'assignatura i es resumeixen bàsicament en els següents punts:

- Els Work Items llegeixen tots un mateix element a la matriu A, i coalescentment a la matriu B
- Si el Work Group te 2 o mes WI a la dimensió Y, llavors, el Work Group llegeix repetides vegades de la matriu B.
- Utilitzant memòria local per a la matriu A, podem dividir les lectures a memòria global, corresponents a la matriu A, pel nombre de WI que hi hagi al WG.
- Utilitzant memòria local per a la matriu B, podem reduir el nombre de lectures a memòria global, corresponents a la matriu B, a la meitat.

Cal destacar que per obtenir un bon rendiment a l'hora de paral·lelitzar cal tenir en compte els recursos disponibles i la manera de repartir la feina entre els workers disponibles.

Encara que és obvi que no tots els problemes poden ser tan fàcilment paral·lelitzables (en el sentit de que es pugui maximitzar el benefici sobre arquitectures GPU, com ara al cas del producte de matrius), som conscients de que en funció de com estructurem el problema i les dades a computar, en molts casos podrem obtenir uns guanys en rendiment considerables.