PARCIAL I

High Performance Computing

DANIEL DIAZ GIRALDO

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
PEREIRA, RISARALDA
2014

Introducción

A lo largo de la evolución del hardware se han presentado problemas de rendimiento para la solución de algoritmos de alta demanda de recursos. Las diferentes implementación han evolucionado dependiendo de las necesidades estrictas del entorno, se han pasado de entornos secuenciales a procesadores con capacidades mucho más potentes, pero con arquitecturas no tan óptimas para enfoques de convergencia a problemas de nuevo índole.Nace así, el pensamiento paralelo, aprovechando las grandes capacidades de cómputo de la unidades de procesamiento gráfico (GPU), ayudando a la resolución eficaz de problemas algorítmicos que tardarían mucho tiempo en computarse en una arquitectura tradicional.

La intención del documento, es comparar desde aspectos claves (aceleraciones,tiempos) la computación de algoritmos, implementados directamente desde enfoques distintos (CPU,GPU...) con diferentes técnicas de optimización.

El diseño de los algoritmos planteados están compuestos por el lenguaje de programación c++ y la implementación de la compañía ensambladora Nvidia con su módulo para este lenguaje (“.cu”)

Proceso Aplicado

El propósito del documento es tratar de implementar el algoritmo de multiplicación de matrices pasando por los diferentes niveles de implementación y arquitectura en dónde correrá.

A continuación se explicarán los pasos aplicados en la tarea investigativa.

- Análisis de algoritmos (Secuencial - Paralelo - Paralelo optimizado): Planteamiento de la solución “normal” del algoritmo de multiplicación de matrices.
- Construcción de algoritmos - Implementación (Secuencial - Paralelo - Paralelo optimizado) : Aplicación de conocimientos para la construcción de los algoritmos paralelos, en conjunto a la utilización de la plataforma en donde se implementarán.
    - Implementación tipo de dato INT
    - Implementación tipo de dato Float
- Comparativa de parámetros- Resultado (Tiempo - Aceleración): Dentro del modelo planteado para cada tipo de implementación, se establecerá un benchmarking para los 3 tipos de implementación bajo un mismo archivo ejecutable.
- Conclusiones : Análisis interpretativo de los valores obtenidos por la realización de pruebas.

Implementación

Ver: https://github.com/DanDayz/Hpc/tree/master/Tests/Test1

Datos obtenidos

A continuación se mostrarán los datos obtenidos durante la ejecución del algoritmo.

Especificaciones técnicas de la máquina (GPU):

SELECT THE RIGHT TESLA GPU

Features	Tesla K40
GPU	1 Kepler GK110B
Peak double precision floating point performance	1.66 Tflops (GPU Boost Clocks) 1.43 Tflops (Base Clocks)
Peak single precision floating point performance	5 Tflops (GPU Boost Clocks) 4.29 Tflops (Base Clocks)
Memory bandwidth (ECC off)[2]	288 GB/sec
Memory size (GDDR5)	12 GB
CUDA cores	2880

(link: http://www.nvidia.com/object/tesla-servers.html)

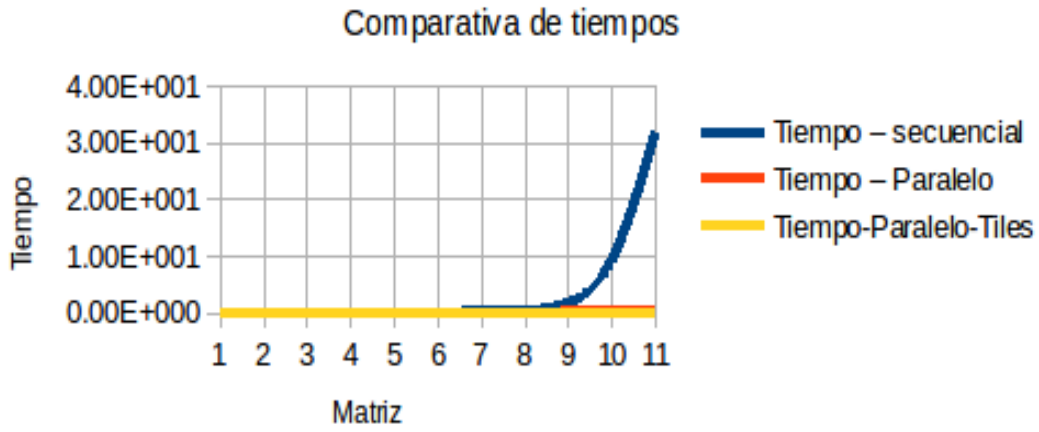
Especificaciones técnicas de la máquina :

- I7 3770k
- Ram 16GB DDR3

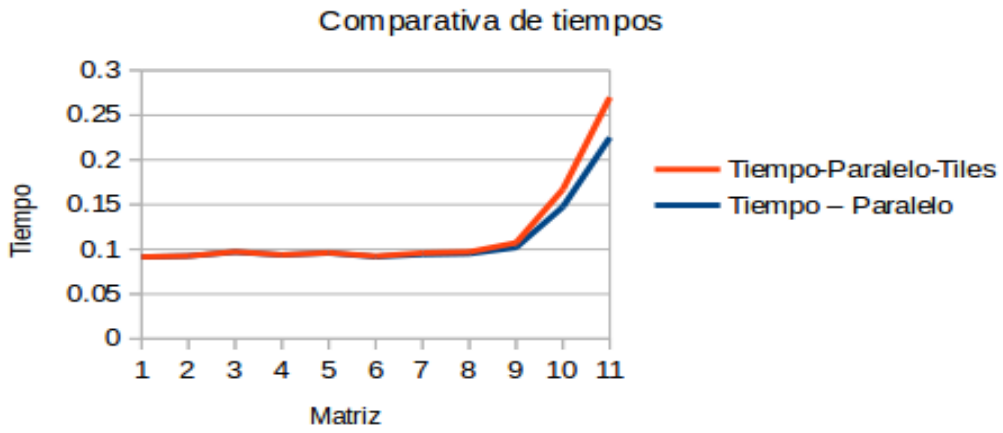
Implementación tipo de dato INT

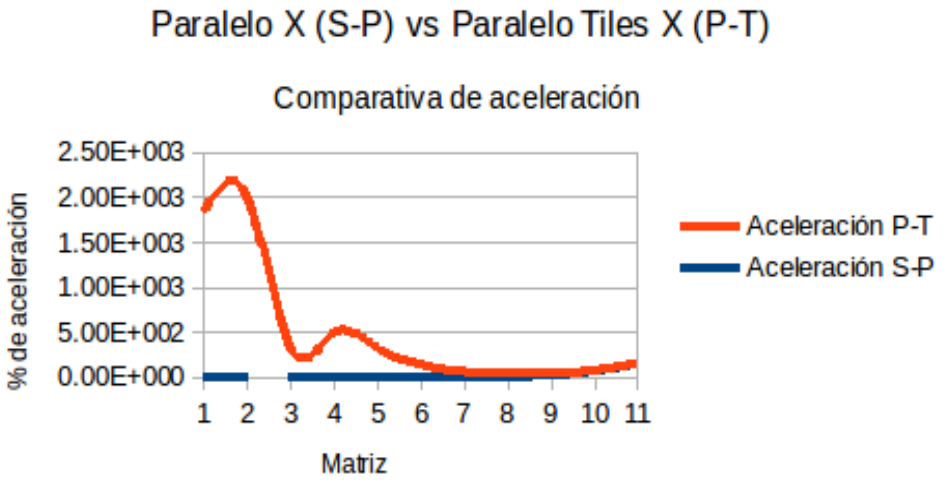
	Tiles=32	Blocksize=32						
	Mat1	Mat2	Mat3	Tiempo – secuencial	Tiempo – Paralelo	Tiempo-Paralelo-Tiles	Aceleración S-P	Aceleración P-T
1	**4x8**	**8x10**	**4x10**	3.00E-006	0.091483	5.00E-005	3.28E-005	1.83E+003
2	**32x16**	**16x32**	**32x32**	5.70E-005	0.092504	4.60E-005	0.00061619	2010.96
3	**64x32**	**32x128**	**64x128**	0.000697	0.09685	0.000287	0.0071967	337.456
4	**128x128**	**128x256**	**128x256**	0.01624	0.093676	0.000193	0.216509	485.368
5	**256x128**	**128x160**	**256x160**	0.016691	0.095845	0.000296	0.174146	323.801
6	**320x256**	**256x240**	**320x240**	0.051699	0.091659	0.000679	0.564036	134.991
7	**400x200**	**200x600**	**400x600**	0.135828	0.094192	0.001735	2.50907	54.2893
8	**512x360**	**360x600**	**512x600**	0.315207	0.095076	0.002036	3.31532	46.6974
9	**1024x512**	**512x960**	**1024x960**	1.60406	0.102027	0.005278	15.7219	19.3306
10	**2048x1024**	**1024x1200**	**2048x1200**	9.3385	0.147428	0.019629	63.3428	7.51072
11	**2048x2048**	**2048x1500**	**2048x1500**	31.9431	0.224963	0.044892	141.993	5.0112

Secuencial vs Paralelo vs Tiles

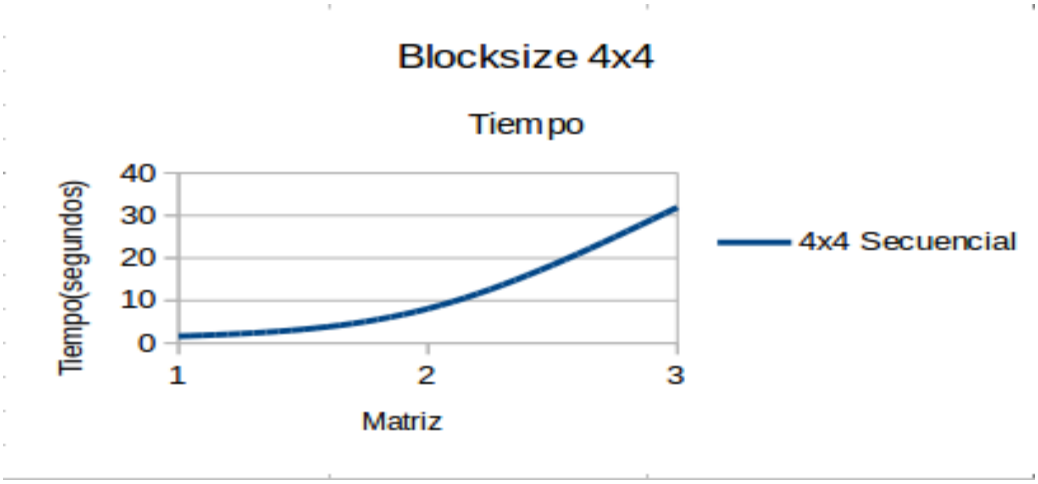


Paralelo vs Paralelo Tiles

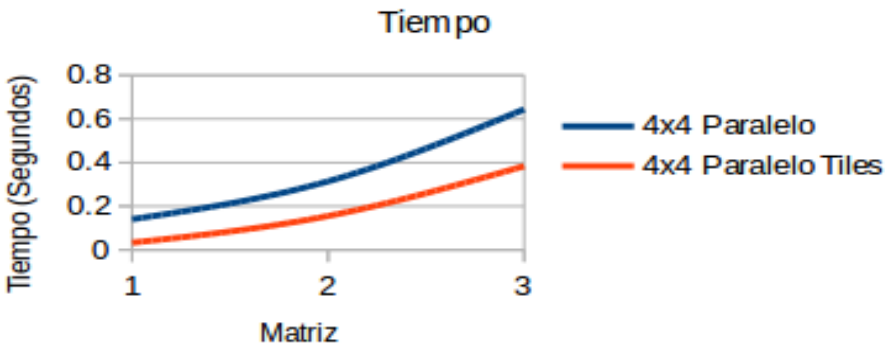



Paralelo X (S-P) vs Paralelo Tiles X (P-T)
Comparativa de aceleración
% de aceleración
2.50E+003
2.00E+003
1.50E+003
1.00E+003
5.00E+002
0.00E+000
1 2 3 4 5 6 7 8 9 10 11
Matriz
Aceleración P-T
Aceleración S-P

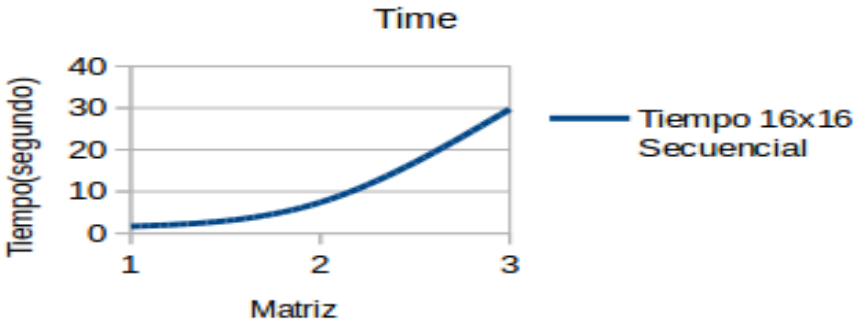
Blocksize


Blocksize 4x4
Tiempo
Tiempo(segundos)
40
30
20
10
0
1
2
3
Matriz
4x4 Secuencial

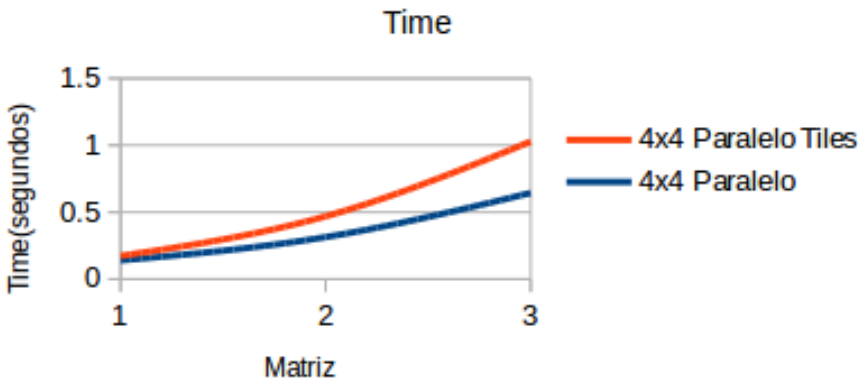
Blocksize 4x4



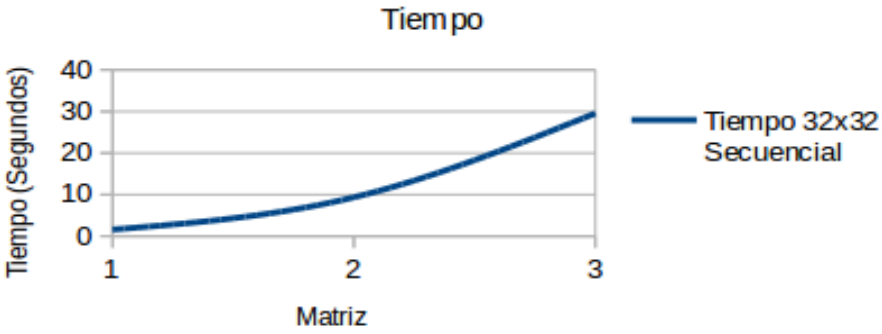
Blocksize 16x16



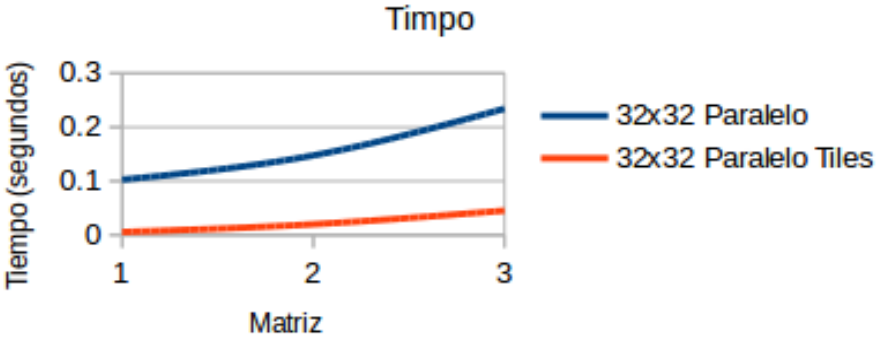
Blocksize 16x16



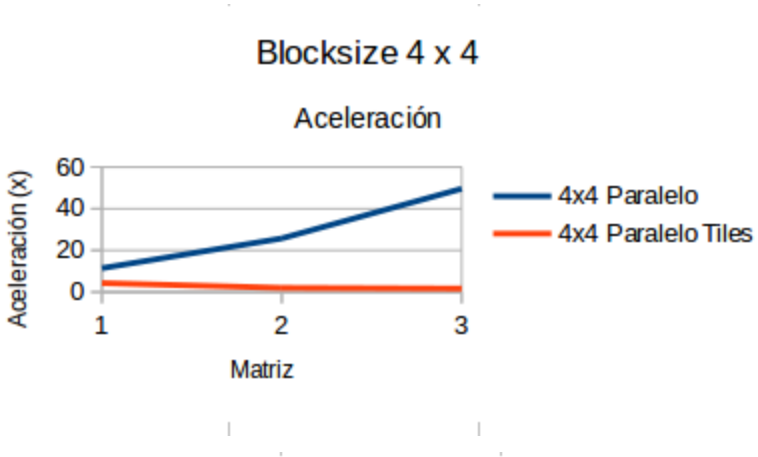
Blocksize 32 x 32

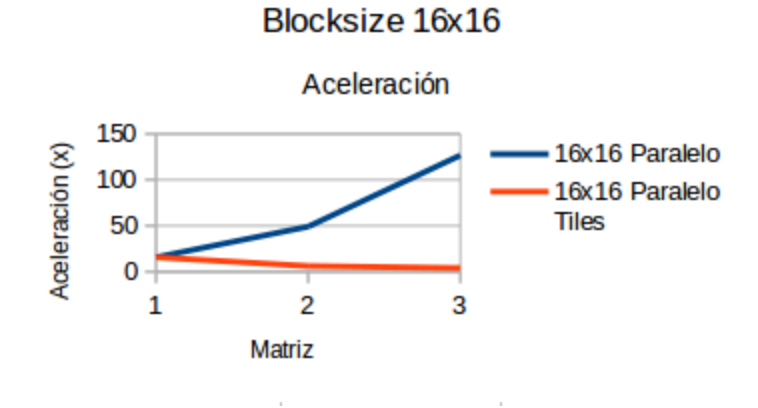


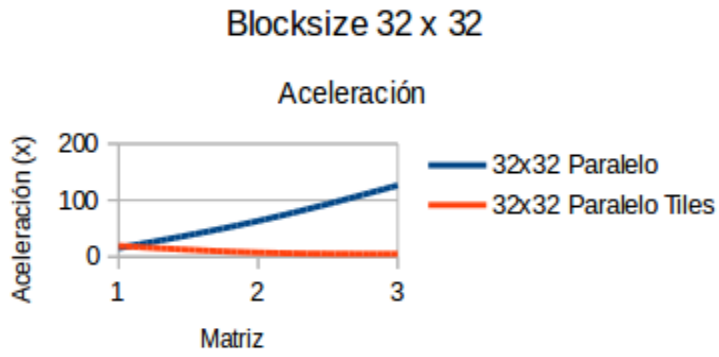
Blocksize 32 x 32



Aceleración





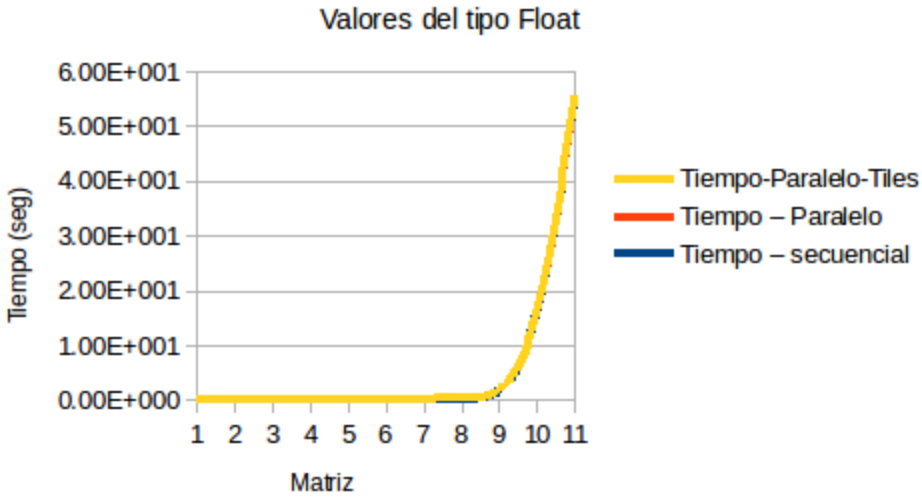


Nota: La aceleración de la implementación “paralelo con tiles” fue tomada respecto a la implementación paralela.

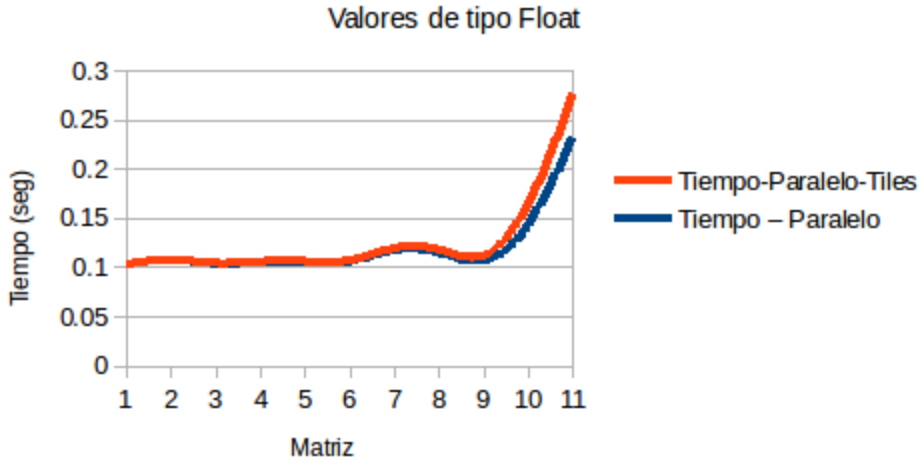
Implementación Float

	Tiles=32	Blocksize=32						
	Mat1	Mat2	Mat3	Tiempo – secuencial	Tiempo – Paralelo	Tiempo-Paralelo-Tiles	Aceleración S-P	Aceleración P-T
1	**4x8**	**8x10**	**4x10**	3.00E-006	0.102484	9.20E-005	2.93E-005	1.11E+003
2	**32x16**	**16x32**	**32x32**	1.07E-004	0.107572	8.20E-005	0.000994683	1311.85
3	**64x32**	**32x128**	**64x128**	0.000704	0.10367	0.000308	0.00679078	336.591
4	**128x128**	**128x256**	**128x256**	0.018474	0.105106	0.000605	0.175765	173.729
5	**256x128**	**128x160**	**256x160**	0.018787	0.105637	0.000335	0.177845	315.334
6	**320x256**	**256x240**	**320x240**	0.06026	0.106239	0.00049	0.567212	216.814
7	**400x200**	**0.145748**	**400x600**	0.145748	0.117095	0.002041	1.2447	57.3714
8	**512x360**	**360x600**	**512x600**	0.31939	0.114927	0.00284	2.77907	40.4673
9	**1024x512**	**512x960**	**1024x960**	1.8035	0.106792	0.005225	16.888	20.4387
10	**2048x1024**	**1024x1200**	**2048x1200**	16.3538	0.144399	0.021306	113.254	6.77739
11	**2048x2048**	**2048x1500**	**2048x1500**	55.3183	0.233736	0.043972	236.67	5.31556

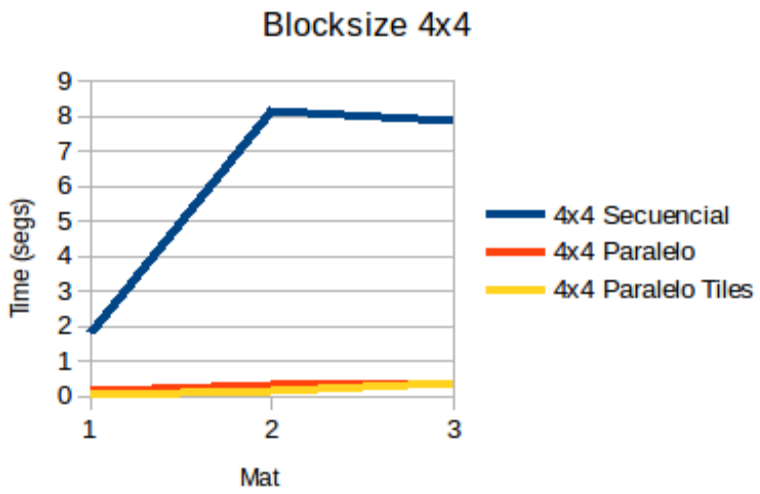
Serial vs Paralelo vs Tiles

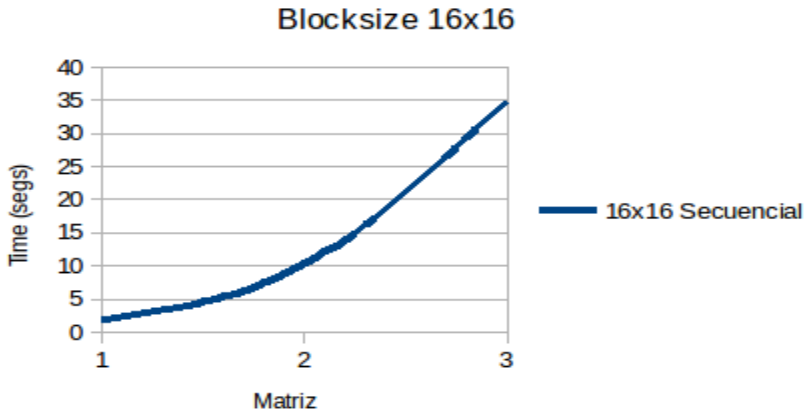


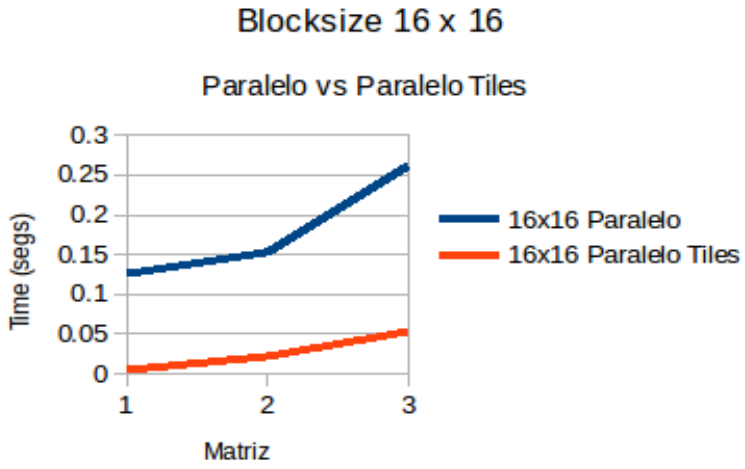
Paralelo vs Tiles



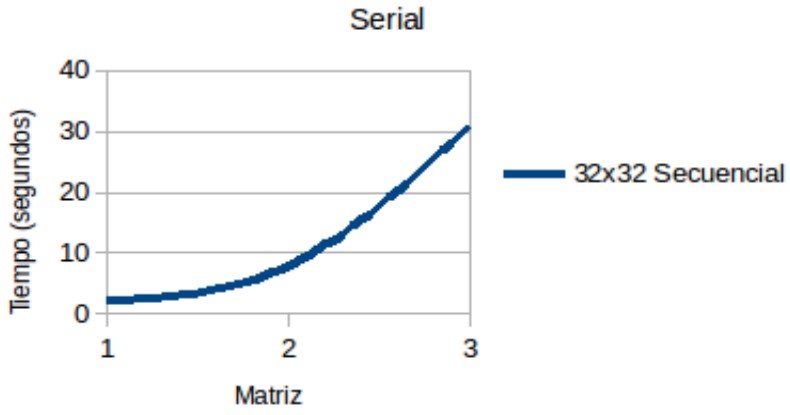
Blocks


Blocksize 4x4
Time (segs)
9
8
7
6
5
4
3
2
1
0
1
2
3
Mat
4x4 Secuencial
4x4 Paralelo
4x4 Paralelo Tiles

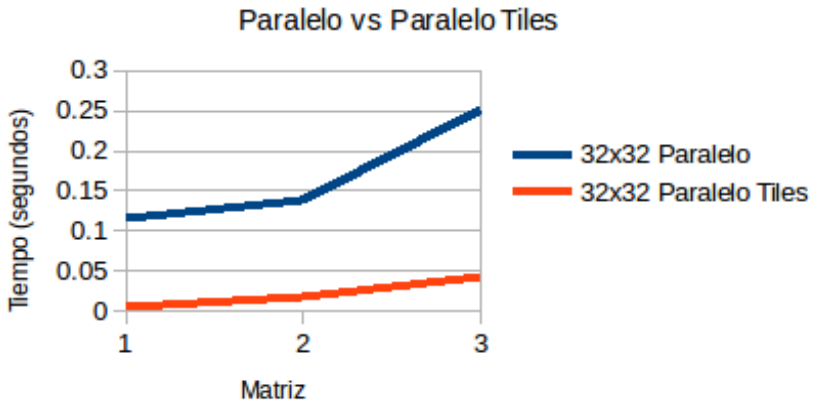

Blocksize 16x16
Time (segs)
40
35
30
25
20
15
10
5
0
1
2
3
Matriz
16x16 Secuencial


Blocksize 16 x 16
Paralelo vs Paralelo Tiles
Time (segs)
0.3
0.25
0.2
0.15
0.1
0.05
0
1
2
3
Matriz
16x16 Paralelo
16x16 Paralelo Tiles

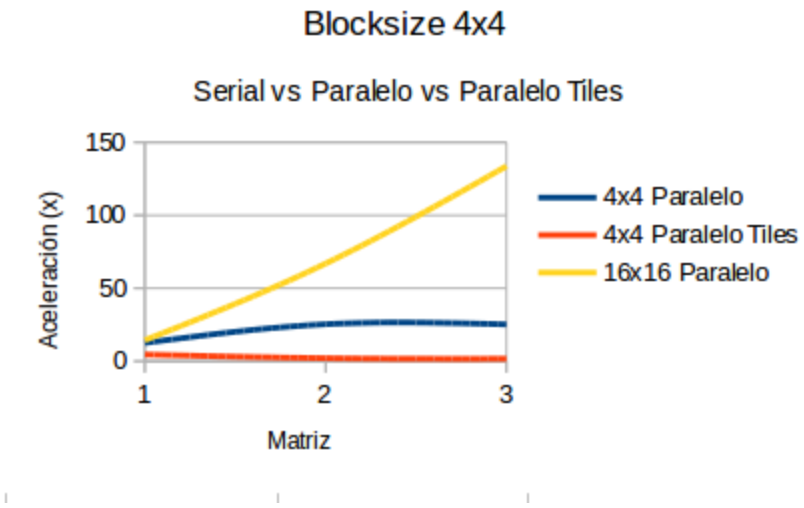
32 X 32

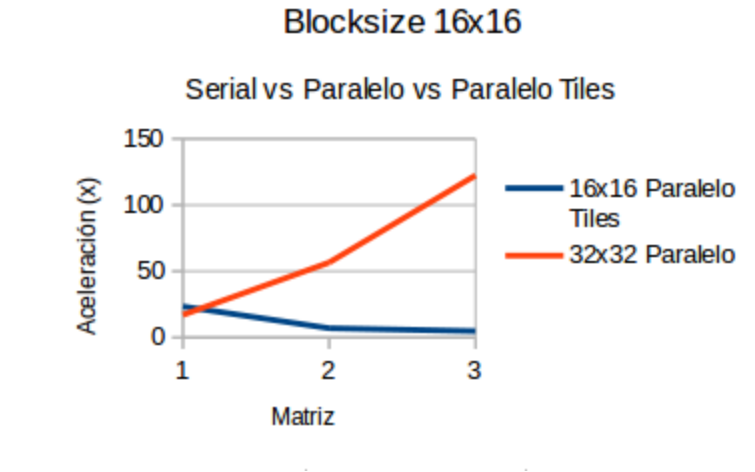

Serial
40
30
20
10
0
Tiempo (segundos)
1
2
3
Matriz
32x32 Secuencial

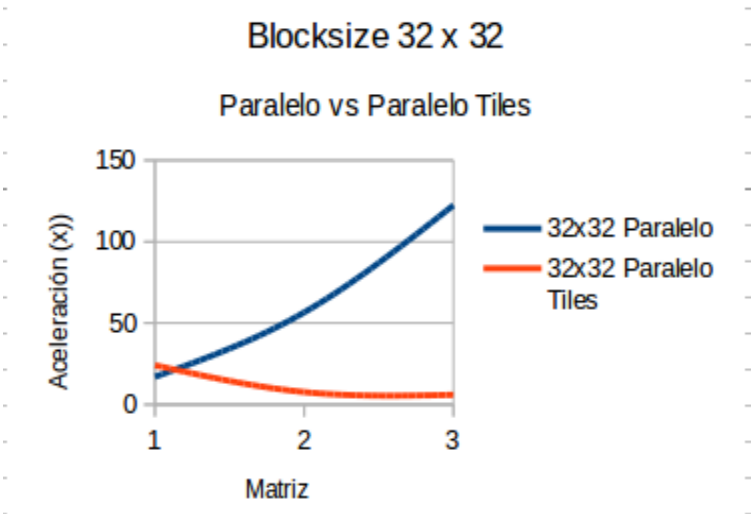
32 x 32


Paralelo vs Paralelo Tiles
0.3
0.25
0.2
0.15
0.1
0.05
0
Tiempo (segundos)
1
2
3
Matriz
32x32 Paralelo
32x32 Paralelo Tiles

Aceleración







Conclusiones

Dentro del análisis concreto de resultado, se encontró que:

- La implementación del algoritmo secuencial en la máquina es relativamente “eficiente” cuando se trata de dimensiones estimables.
- La implementación del algoritmo paralelo es muy eficiente a medida que se eleva el tamaño de las matrices a multiplicar.
- La memoria compartida es una fuerte herramienta de optimización hasta cierto punto, cuando esta se ocupa en su totalidad , tiende a perder velocidad de aceleración.
- Respecto a la utilización de “Tiles” , se evidenció que a un menor número de éstos (también de bloques) , el tiempo de convergencia era limitado debido a que su tamaño no era el mejor, abarcando pocos registros por cada tile,aumentando el ciclo de cálculo en la matriz; contrario a tener un número de bloques más grandes (tamaño), que aprovechan mejor los recursos estipulados por la arquitectura.
- La utilización de una plataforma compartida denota cambios importantes en la obtención de datos, dada la naturaleza de los trabajos y de la compartición de recursos.
- Las aceleraciones estimadas de la implementación con tiles está limitada a la memoria compartida en el device.