TECHNICAL REPORT
SEBASTIÁN GALGUERA ORTEGA
OCTOBER 2018
MULTICORE PROGRAMMING

Abstract

The following document describes the process that was followed in order to apply the tiling technique to a matrix multiplication, using sequential and parallel approaches with OpenMp and Nvidia's CUDA in order to compare speedups. The aim of the task was to provide a way to accelerate a rather trivial problem that can have many real world applications.

1. Introduction

Matrix multiplication is a process that involves many calculations. This kind of operation provides a way to solve different real world problems not only on the domain of mathematics, but also on many aspects of engineering and software development. Of course, the solution to this calculation is trivial in the sense that the process is well understood and known. Nevertheless, many things can be learned from making this calculation more efficient, potentially speeding up entire programs like image processors and simulation software.

For this practice, the speed up was done using tiling technique, in which thread blocks are stored in shared memory in a tiled fashion. Shared memory inherently has less latency than global memory, but it is also much smaller. Because of this, shared memory must be used wisely.

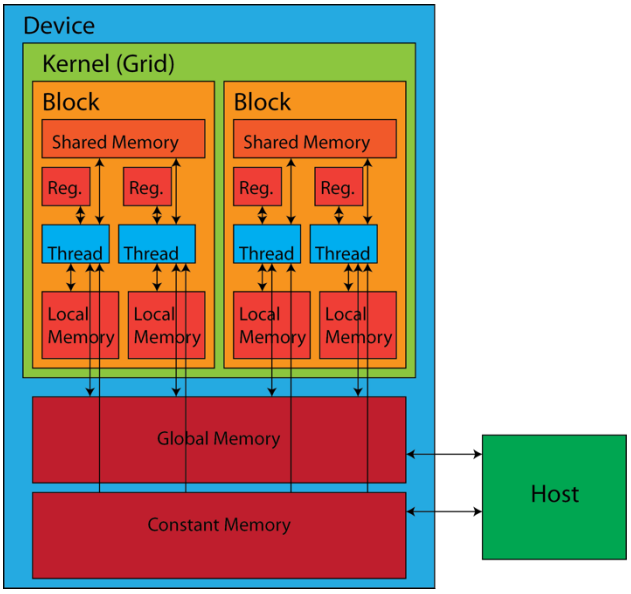


FIGURE 1.0 Memory architecture visualization.

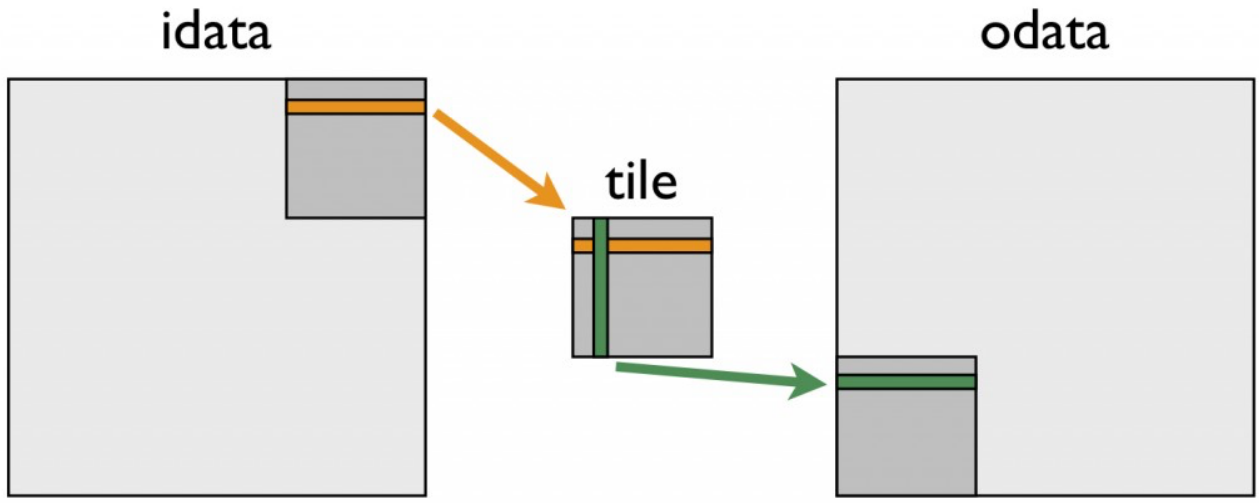


FIGURE 2.0 Tiling technique visualization.

2. Development

The main tests were executed on a remote server via secure shell. A size of 2000 x 2000 for the matrix was chosen. Also, the block size configured in the kernel was specified as the size of the tile. 8x8, 16x16 and 32x32 tiles were used for the practice.

The computer specifications:
Brand: Alienware 32 GB RAM
Model name: Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz
CPU: 8
CPU: 3.9 GHZ
CPU Cores: 6
Threads per Core: 2
GPU: NVIDIA Corporation GK104 [GeForce GTX 670] [10de:097a]
GPU Cores: 1344
Architecture: x86_64

The following table specifies the average elapsed time measured in milliseconds of each function for each image size. Chart provided as appended content in section 4.

TABLE OF GENERAL AVERAGE ELAPSED TIME (ms)	2000 x 2000
Host Sequential	75296.062500
Host with Threads	18883.210938
CUDA GPU	224.717697
CUDA GPU TILES	74.247330

TABLE OF IMPROVEMENT (S(n)=T(1)/T(n))	8x8	16x16	32x32
NORMAL	382.6390365	261.6701854	225.7649881
TILING	169.7812412	85.51025826	75.33961637
SPEEDUP	2.253155666	3.058479098	2.9947033

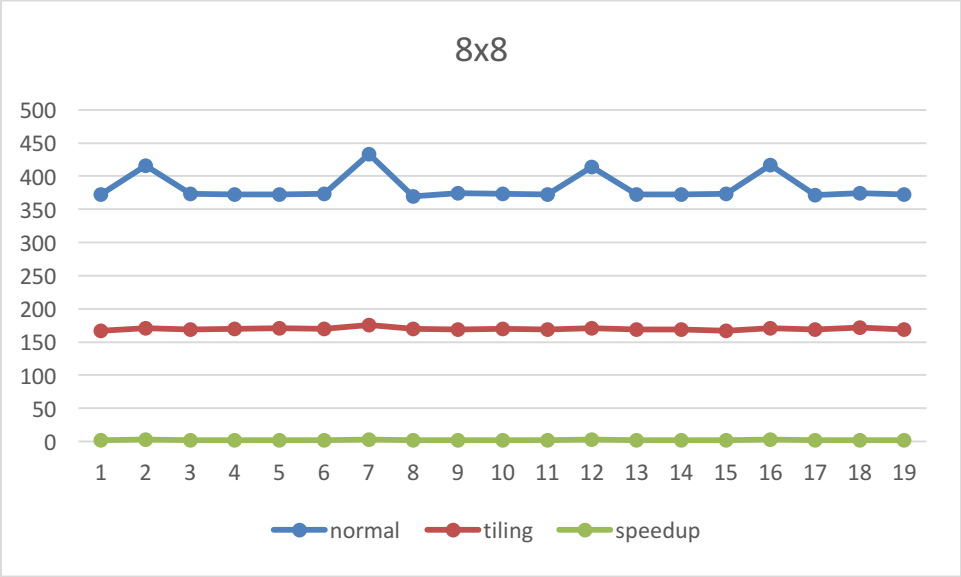
As it can be seen, tile size represents different performance.

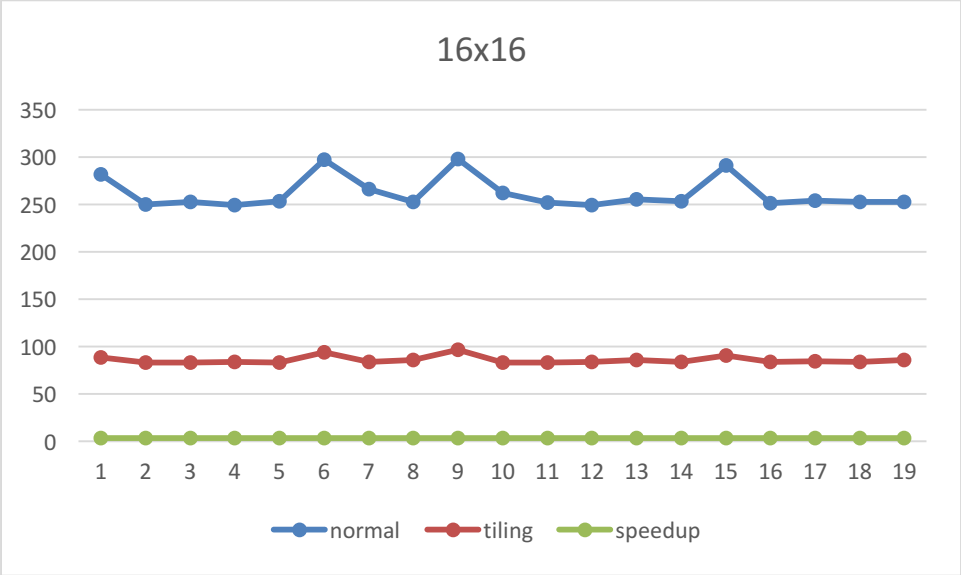
3. Conclusion

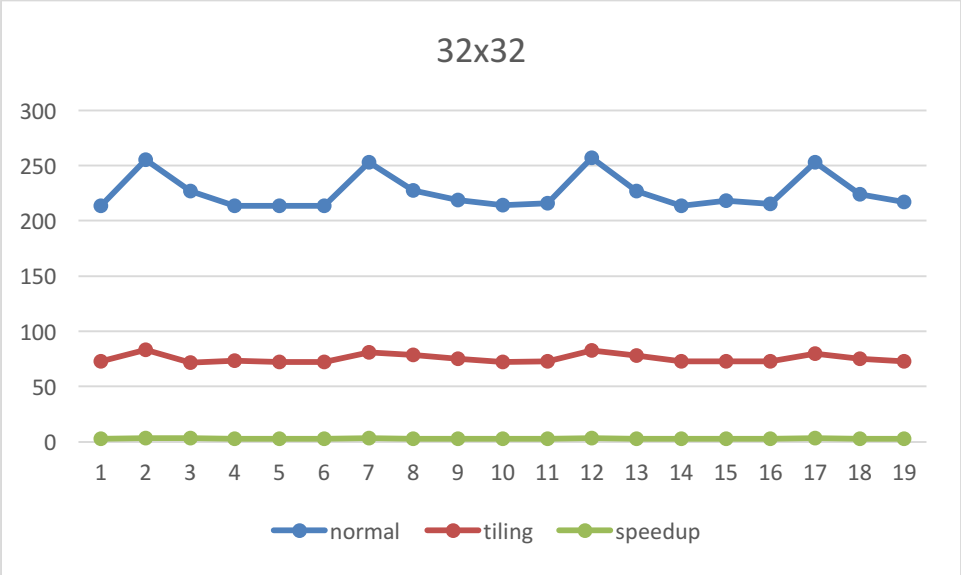
With this, it can be concluded that the vast superiority of CUDA emerges in this kind of problems. Of course, even while using CUDA, there can be several configurations and architectures that can speed up even more the performance of the whole application. Because of this, we can implement new tools and get better overall times of execution.

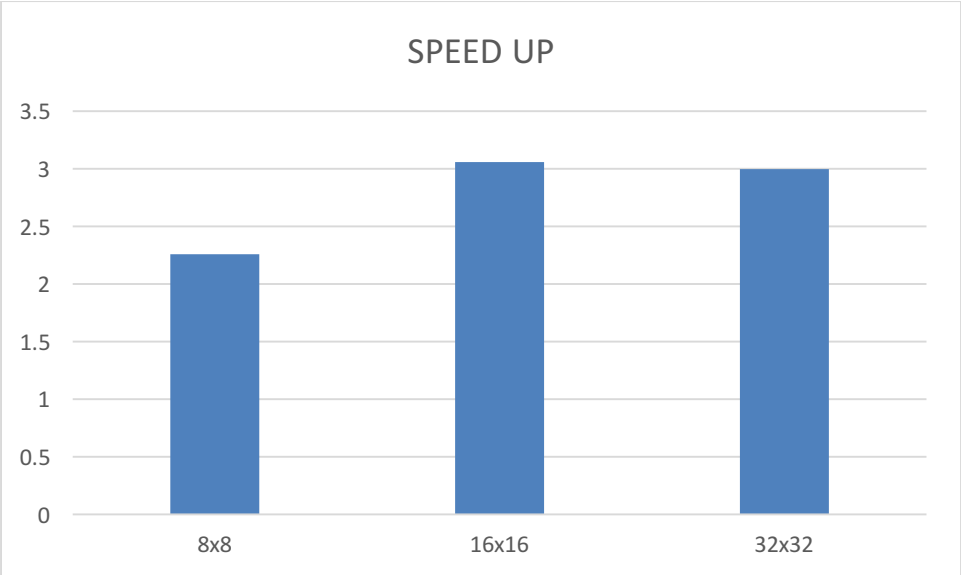
4. References and Figures

"GEFORCE RTX™." *NVIDIA*, NVIDIA, 1 Sept. 2018, www.nvidia.com/es-la/.
Podlozhnyuk, Victor. "Image Convolution with CUDA ." NVIDIA, 11 June 2007.




16x16
350
300
250
200
150
100
50
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
normal
tiling
speedup


32x32
300
250
200
150
100
50
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
normal
tiling
speedup


SPEED UP
3.5
3
2.5
2
1.5
1
0.5
0
8x8
16x16
32x32