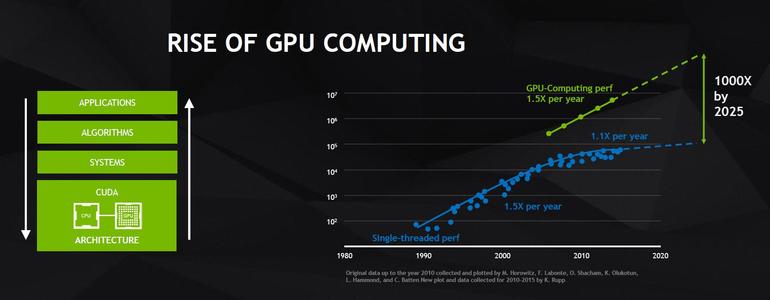
CUDA Material:

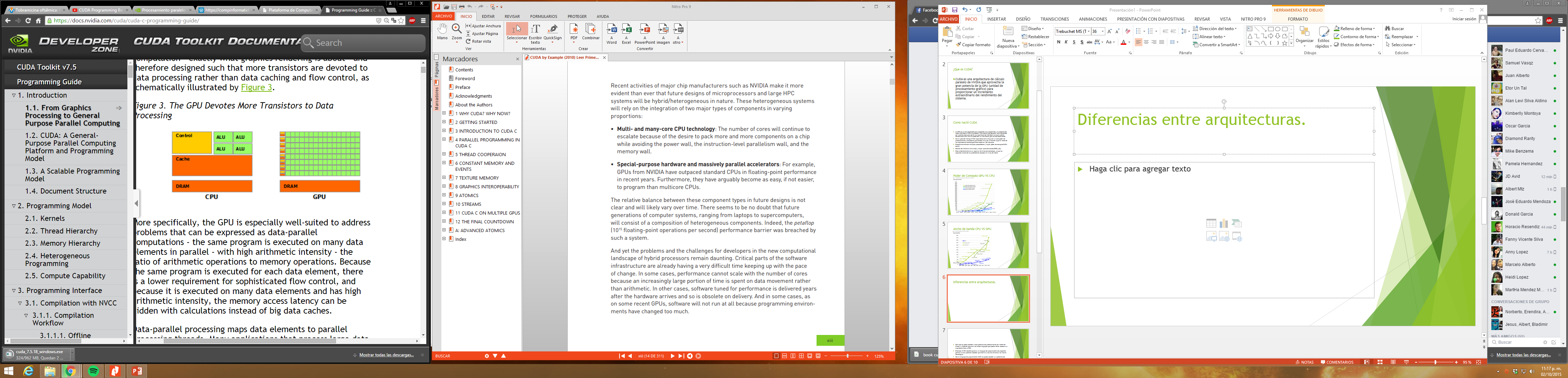
**Key Concepts for GPU Parallel Programming:** Memory Management,Resource Allocation, and Race Conditions.

**Before class:** configure your access to the server following these instructions <https://benjaminva.github.io/server/> Nvidia GeForcecard,Cuda 8.0, Visual Studio community edition 13 or 15 (Windows) or use preconfigured server (ssh)

**During Class:**

**Note:(**All of the following diagrams come directly from Nvidia CUDA courses)

* Programming on GPU
* History Nvidia the underdog, (Riva 128). <https://en.wikipedia.org/wiki/RIVA_128>
* Nvidia 1999 first GPU <https://en.wikipedia.org/wiki/GeForce_256>
* 2006 birth of Cuda architecture, GPUs for general purpose computing.
* Overview of cards and the things they can do <https://en.wikipedia.org/wiki/CUDA>
* 2008 Tegra for mobile, stolen by Samsung. (for gossip) <https://www.theinquirer.net/inquirer/news/2380961/samsung-countersues-nvidia-for-false-and-misleading-tegra-k1-benchmark-scores>
* 2008 buys ageia developer of Physx which is used for video games and lets CUDA be included in GForce graphix cards.
* G780 512 CUDA. GTx 1080 2000 cores. 1 thread per core (16,000,000 thread with virtualisation per card)
* Hardware: Silicio vs grafeno
* Current status nVidia GTC 2017 <https://www.youtube.com/watch?v=Rn73n1HYYNs>
* GPUs and CPUs architecture PerformanceGPUs and CPUs architecture



* X86 has not been displaced due to backward compatibility, therefor there are no commercial OS that run on pure GPUs.
* Pros and Cons
* GPUs are bad decision makers and are bad sequential runners (their instruction sets for handling these functionalities are incomplete).
* Setting up your environment or accessing the server.

**Reading Material 1:**

Read and make notes of chapter 3 Introduction to CUDA C (complete chapter) from the book CUDA by Example.

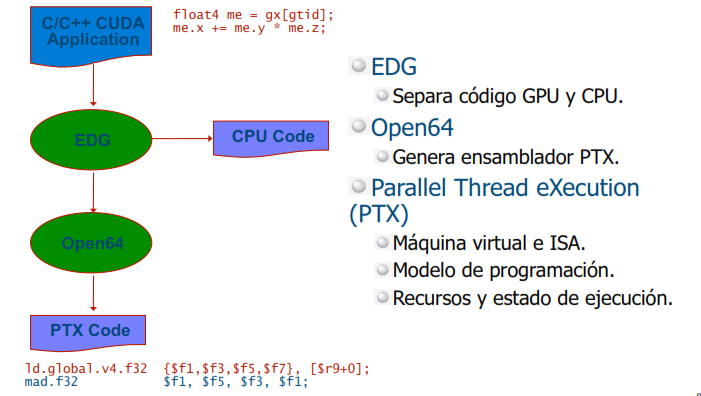
* How can you distinguish the code in CPU from the code for GPU in CUDA?

**During Class:**

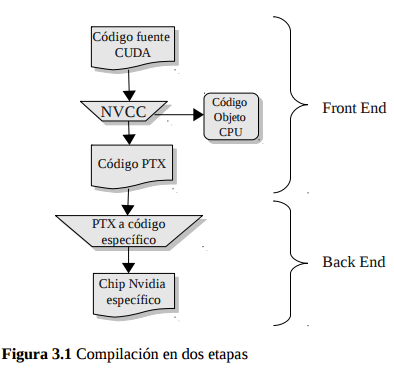
* Open CL standard (AMD used for mining, Chaos group intel, and others) vs CUDA
* Languages for CUDA:
  + Best is **C**
  + **PyCuda** not transparent
  + **Java** version still too young.

**Implement together:** *BasicSum* First CUDA example. (Programming example)

* Compilation happens in several steps:



* Two object codes are generated for the same program. One code is for GPU and the other is for CPU.



* When the GPU code is executed, it goes through a translator (the Driver) that changes each code to instructions for each exact card.

**Implement together** Getting to know your GPU (GPUtraits) (Programming example)

* Using the thread Indexes.

**Implement together** Comparing Cpu vs Gpu performance (VectorSum) (Programming example)

* See and run cuda-memcheck example.

**Lab 1:**

After doing this lab you should understand how to assign work to cpu and to gpu in CUDA parallel computing algorithms.

* The following code calculates PI in a sequential way, modify it to turn it into a CUDA program that can calculate PI value in a parallel way. (extra points if you can make it run correctly with 1,000,000 rectangles Hint: you can make minor modifications to the algorithm)

long num\_rects = 100000, i;

double mid, height, width, area;

double sum = 0.0;

width = 1.0 / (double) num\_rects;

for (i = 0; i < num\_rects; i++) {

mid = (i + 0.5) \* width;

height = 4.0 / (1.0 + mid \* mid);

sum += height;

}

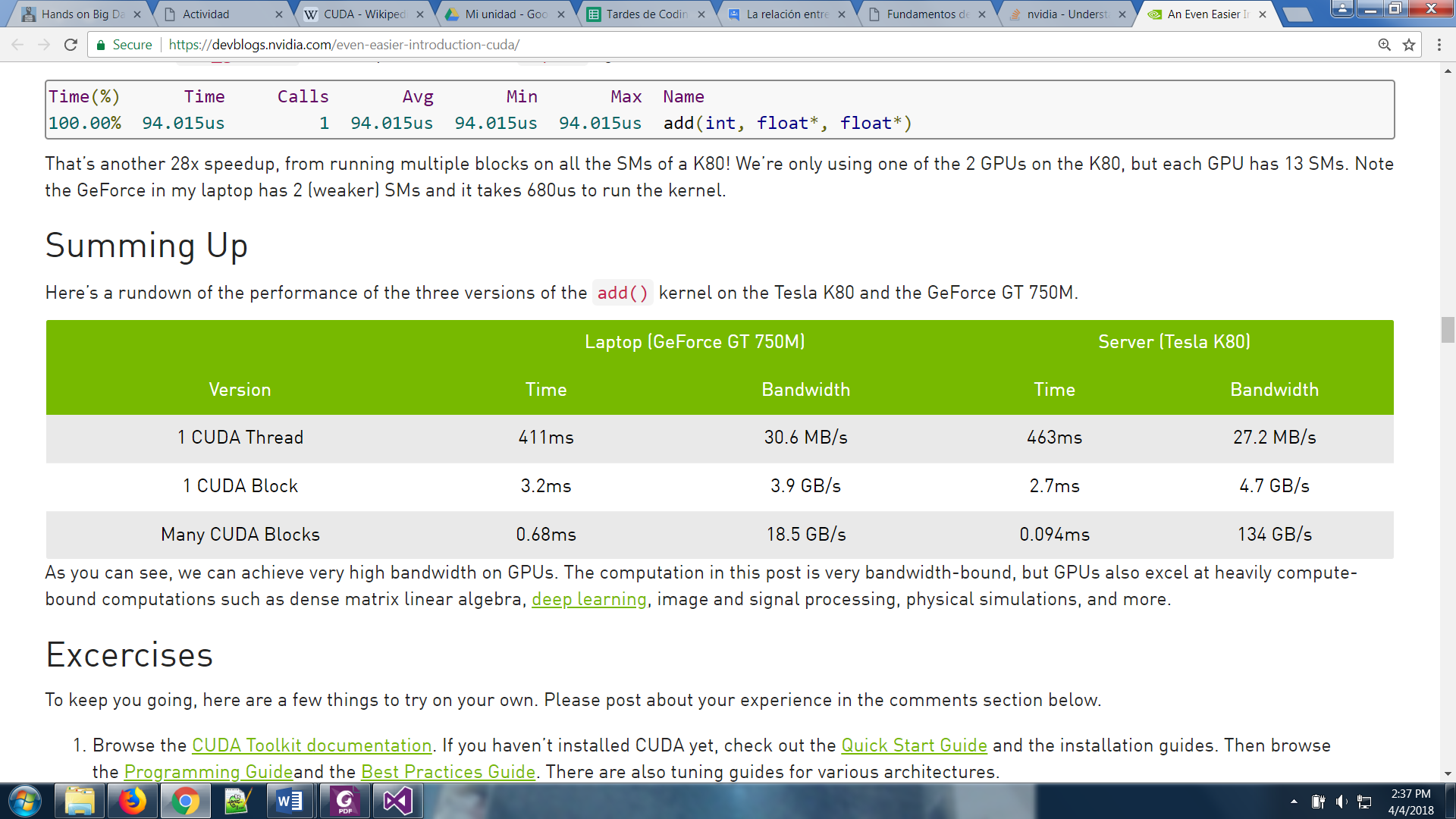
area = width \* sum;

**Reading Material 2:**

Read and make notes of chapter 4 Parallel Programming in CUDA C (complete chapter) from the book CUDA by Example.

* What happens to arrays when they entered memory in CUDA?
* How do we compensate for this change in representation (how do we now where the data is in memory)?

**During Class:**

* How do parallel programs work?
* What is thread.Idx
* What is blockDim?
* What is GridDim?
* Look at [Example of Threads and Blocks.pdf](Example%20of%20Threads%20and%20Blocks.pdf) from (2011) Samuel S. Cho course
* Comparison from Nvidia

<https://devblogs.nvidia.com/even-easier-introduction-cuda/>

* Blocks are in fact software created units. Which will be queued (similar to the threadpools) here is a nice explanation: <https://stackoverflow.com/questions/2392250/understanding-cuda-grid-dimensions-block-dimensions-and-threads-organization-s>
* Matrix Addition example
* Implement Matrix Convolution together.

**Lab 2**

After doing this lab you should understand how memory behaves in CUDA parallel computing algorithms and how to pass arrays to the GPU

* **Implement a matrix multiplication** using a GPU to solve the operation instead of a CPU. Create the matrices in the CPU pass them to the GPU calculate the answer and show the answer in the console in an ordered way. You can assume that the matrixes will be square. If your method works well for non-square matrixes you’ll get extra points.
* **Test cases :**

1)

original mats

1.00 2.00

3.00 4.00

4.00 5.00

6.00 7.00

calc mats

row 0 \* height 2 col 0 index 0 values 16.000000

row 0 \* height 2 col 1 index 1 values 19.000000

row 1 \* height 2 col 0 index 2 values 36.000000

row 1 \* height 2 col 1 index 3 values 43.000000

16.00 19.00

36.00 43.00

2)

original mats

1.00 2.00 3.00

4.00 5.00 6.00

6.00 7.00

8.00 9.00

0.00 1.00

calc mats

row 0 \* height 2 col 0 index 0 values 22.000000

row 0 \* height 2 col 1 index 1 values 28.000000

row 1 \* height 2 col 0 index 2 values 64.000000

row 1 \* height 2 col 1 index 3 values 79.000000

22.00 28.00

64.00 79.00

3)

original mats

1.00 2.00

3.00 4.00

5.00 6.00

6.00 7.00 8.00

9.00 0.00 1.00

calc mats

row 0 \* height 3 col 0 index 0 values 24.000000

row 0 \* height 3 col 1 index 1 values 7.000000

row 0 \* height 3 col 2 index 2 values 10.000000

row 1 \* height 3 col 0 index 3 values 54.000000

row 1 \* height 3 col 1 index 4 values 21.000000

row 1 \* height 3 col 2 index 5 values 28.000000

row 2 \* height 3 col 0 index 6 values 84.000000

row 2 \* height 3 col 1 index 7 values 35.000000

row 2 \* height 3 col 2 index 8 values 46.000000

24.00 7.00 10.00

54.00 21.00 28.00

84.00 35.00 46.00

Hers is a link to help you out with other test cases <http://matrix.reshish.com/multCalculation.php>

**Reading Material 3:**

Read and make notes of chapter 5 Thread Cooperation C (complete chapter) from the book CUDA by Example.

* What is the difference between a block and a thread?
* In CUDA GPUs what are the more threads per block, or blocks per grid?

**During Class:**

* Explain sharing and synchronization.
* **(Implement** DotProdcutReduction together**)**
* Analyse the code of DotProduct to see how is synchronization occurring in memory
* Making reductions efficient with CUDA.

**Helpful links:**

**How do I rename a project in visual Studio?**

<https://stackoverflow.com/questions/2043618/proper-way-to-rename-solution-and-directories-in-visual-studio>

**painful configuration of OpenCV for Visual Studio in windows**

[libs\_for\_labs\agregaropencv.docx](libs_for_labs/agregaropencv.docx)

[libs\_for\_labs\Crear proyecto de cuda con opencv.docx](libs_for_labs/Crear%20proyecto%20de%20cuda%20con%20opencv.docx)

**CUDA Graphics Extra week**

**1).-**

* Using CUDA with images
* Install cpu\_bitmap.h
  + Add **.dll** files to the **bin** in C:\Windows\SysWOW64 and in **bin** in C:\Windows\system32
  + Copy the content of the folder **common** and paste it to the folder **inculde** in C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\include
  + Copy the content of **lib** folder and paste it in C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\lib
* Julia set example
  + Command to compile in linux **nvcc julia.cu -lGL -lGLU -lglut**
  + Libraries in common must be in the server:
    - common/cpu\_bitmap.h
    - and the whole content of the common folder.
  + To run use: **vglrun ./a.out**

Play with the parameters a bit, (you might crash your graphics card)

**2).-**

* Explain difference between threads and blocks
* Assigning 1 thread per pixel
* RippleExample

**3).-**

* Graphics processing library OpenCV.
* Show OpenCV example **RealTimeVideoFilter/ProcessingVideo** (don’t go into detail, but provide the code so they can experiment)
  + opencv: nvcc [file.cu](http://file.cu) `pkg-config --cflags --libs opencv`
* Use the code from **ProcessingVideo** as a template to help you, choose from the following link 2 CUDA filters and implement them: <http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html>

**(Extra activity)**

If you really liked CUDA and want to try your hands at graphics, then repurpose or integrate any of the graphical applications provided above to your own application. **Simple replication will not get any points**. Only submit if you have done something significantly different or interesting from the basic provided examples. (points may vary from 1 to 5 on final grade)