Programming GPUs with CUDA

Tutorial at 18th IEEE CSE'15 and 13th IEEE EUC'15 conferences
Porto (Portugal). October, 20th, 2015

Manuel Ujaldón
A/Prof. @ University of Málaga (Spain)
Conjoint Senior Lecturer @ Univ. of Newcastle (Australia)
CUDA Fellow @ Nvidia

NVIDIA

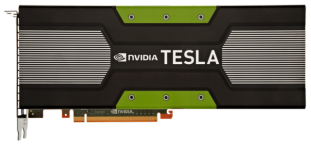
Prerequisites for this tutorial

- You (probably) need experience with C.
- You do not need parallel programming background (but it helps if you have it).
- You do not need knowledge about the GPU architecture: We will start with the basic pillars.
- You do not need graphics experience. Those were the old times (shaders, Cg). With CUDA, it is not required any knowledge about vertices, pixels, textures, ...

I. Introduction

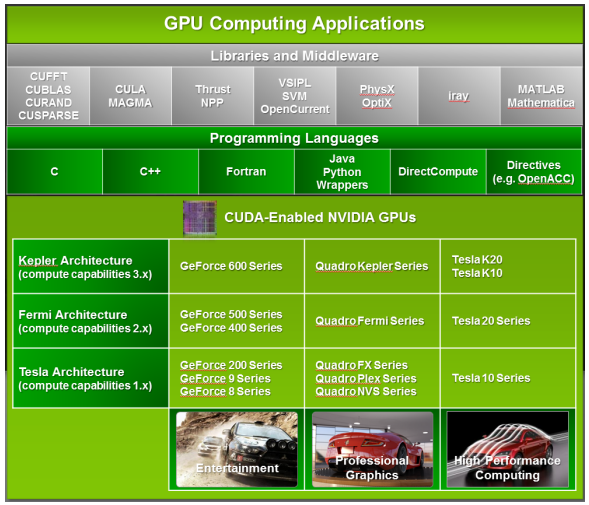
NVIDIA

Commercial models available for Kepler: GeForce vs. Tesla



- Designed for gamers:
  - Price is a priority (<500€).
  - Availability and popularity.
  - Small video memory (1-2 GB.).
  - Frequency slightly ahead.
  - Hyper-Q only for CUDA streams.
  - Perfect for developing code which can later run on a Tesla.
- Oriented to HPC:
  - Reliable (3 years warranty).
  - For cluster deployment.
  - More video memory (6-12 GB.).
  - Tested for endless run (24/7).
  - Hyper-Q for MPI.
  - GPUDirect (RDMA) and other features for GPU clusters.

The characters of this story: The CUDA family picture



The impressive evolution of CUDA

Year 2008	Year 2015
100.000.000 CUDA-capable GPUs (6.000 Teslas only)	600.000.000 CUDA-capable GPUs (and 450.000 Tesla high-end GPUs)
150.000 CUDA downloads	3.000.000 CUDA downloads per year (that is, one every 9 seconds)
1 supercomputer in top500.org (77 TFLOPS)	75 supercomputers in TOP500.org (aggregate 54.000 TFLOPS)
60 university courses	840 university courses
4.000 academic papers	60.000 academic papers

Summary of GPU evolution

- 2001: First many-cores (vertex and pixel processors).
- 2003: Those processor become programmable (with Cg).
- 2006: Vertex and pixel processors unify.
- 2007: CUDA emerges.
- 2008: Double precision floating-point arithmetic.
- 2010: Operands are IEEE-normalized and memory is ECC.
- 2012: Wider support for irregular computing.
- 2014: The CPU-GPU memory space is unified.
- Still pending: Reliability in clusters and connection to disk.

The 3 features which have made the GPU such a unique processor

- Simplified.
  - The control required for one thread is amortized by 31 more (**warp**).
- Scalability.
  - Makes use of the huge **data volume** handled by applications to define a sustainable parallelization model.
- Productivity.
  - Endowed with efficient mechanisms for **switching immediately** to another thread whenever the one being executed suffers from **stalls**.
- CUDA essential keywords:
  - Warp, SIMD, latency hiding, free context switch.

Three reason for feeling attracted to GPUs

- Cost
  - Low price due to a massive selling marketplace.
  - Three GPUs are sold for each CPU, and the ratio keeps growing.
- Ubiquitous
  - Everybody already has a bunch of GPUs.
  - And you can purchase one almost everywhere.
- Power
  - Ten years ago GPUs exceed 200 watts. Now, they populate the Green 500 list. Progression in floating-point computation:

	GFLOPS/w on float (32-bit)	GFLOPS/w. on double (64-bit)
Fermi (2010)	5-6	3
Kepler (2012)	15-17	7
Maxwell (2014)	40	12

What is CUDA? "Compute Unified Device Architecture"

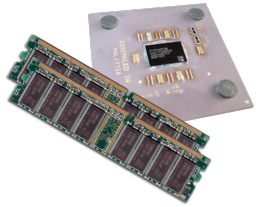
- A platform designed jointly at software and hardware levels to make use of the GPU computational power in general-purpose applications at three levels:
  - Software: It allows to program the GPU with minimal but powerful SIMD extensions to enable heterogeneous programming and attain an efficient and scalable execution.
  - Firmware: It offers a driver oriented to GPGPU programming, which is compatible with the one used for rendering. Straightforward APIs manage devices, memory, ...
  - Hardware: It exposes GPU parallelism for general-purpose computing via a number of twin multiprocessors endowed with cores and a memory hierarchy.

CUDA C at a glance

- Essentially, it is C language with minimal extensions:
  - Programmer writes the program for a single thread, and the code is automatically instanciated over hundreds of threads.
- CUDA defines:
  - An architectural model:
    - With many processing cores grouped in multiprocessors who share a SIMD control unit.
  - A programming model:
    - Based on massive data parallelism and fine-grain parallelism.
    - Scalable: The code is executed on a different number of cores without recompiling it.
  - A memory management model:
    - More explicit to the programmer, where caches are not transparent anymore.
- Goals:
  - Build a code which scales to hundreds of cores in a simple way, allowing us to declare thousands of threads.
  - Allow heterogeneous computing (between CPUs and GPUs).

Heterogeneous Computing (1/4)

- Terminology:
  - Host: The CPU and the memory on motherboard [DDR3 as of 2013].
  - Device: The graphics card [GPU + video memory]:
    - GPU: Nvidia GeForce/Tesla.
    - Video memory: GDDR5 as of 2015.

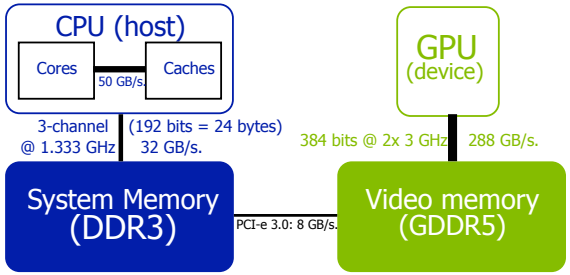


Host

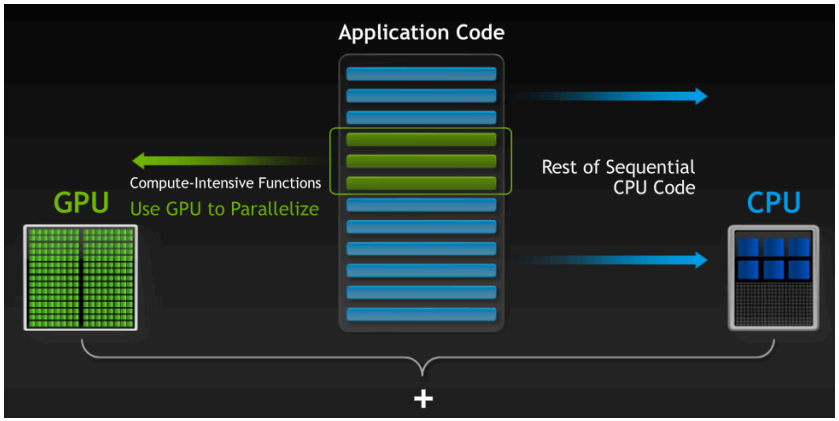
Device

Heterogeneous Computing (2/4)

- CUDA executes a program on a device (the GPU), which is seen as a co-processor for the host (the CPU).
- CUDA can be seen as a library of functions which contains 3 types of components:
  - Host: Control and access to devices.
  - Device: Specific functions for the devices.
  - All: Vector data types and a set of routines supported on both sides.

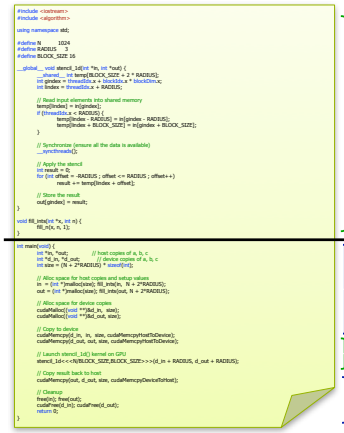


Heterogeneous Computing (3/4)



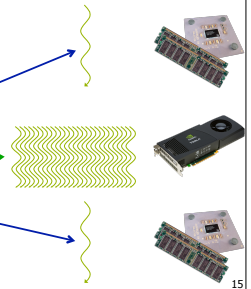
- The code to be written in CUDA can be lower than 5%, but exceed 50% of the execution time if remains on CPU.

Heterogeneous Computing (4/4)

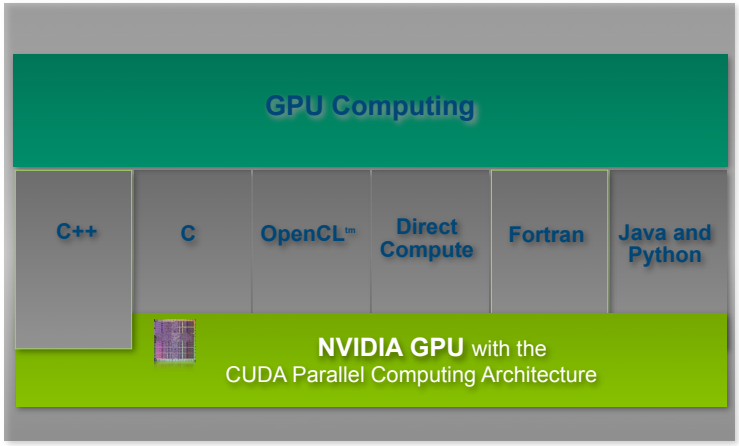


DEVICE CODE:
Parallel function
written in CUDA.

HOST CODE:
- Serial code.
- Parallel code.
- Serial code.



If we have a CUDA architecture, we can approach programming in different ways...



- ... but this tutorial focuses on CUDA C.

CUDA evolution

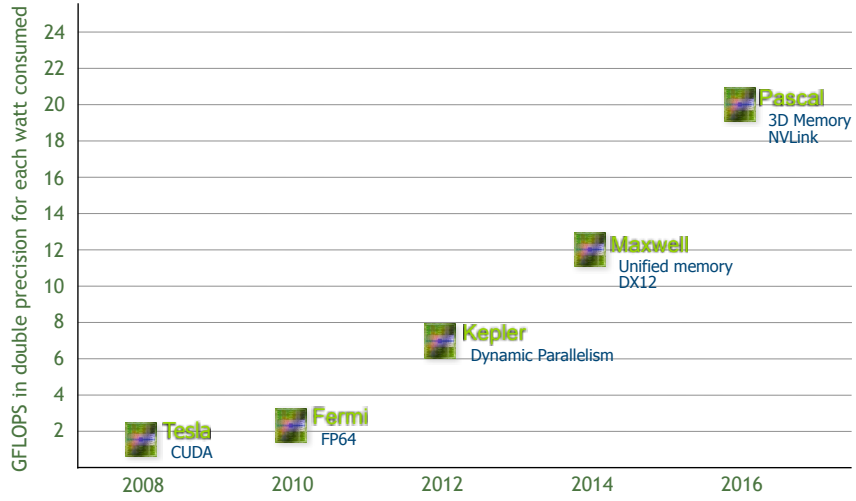
- Over the past 7 years, Nvidia has manufactured more than 500 million CUDA-enabled GPUs.
- CUDA has evolved in the opposite direction we are used to: From scientists/researchers to more generic users.

CUDA version [year]	Users and highlights
1.0 [2007]	Researchers and early adopters
2.0 [2008]	Scientists and HPC applications
3.0 [2009]	Application innovation leaders
4.0 [2011]	Broader developer adoption
5.0 [2012]	Dynamic parallelism, object linking, Remote DMA.
6.0 [2014]	Unified CPU-GPU memory.
Next	Half precision in floating-point arithmetic

II. Architecture

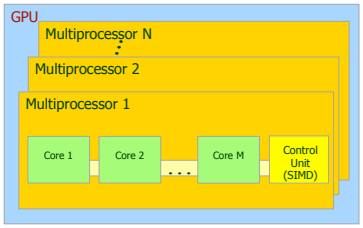
II.1. CUDA hardware model

Overview of CUDA hardware generations



The CUDA hardware model: SIMD processors structured, a tale of hardware scalability

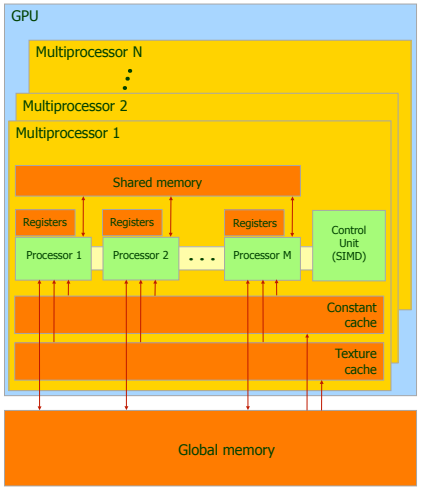
- A GPU consists of:
  - N multiprocessors (or SMs), each containing M cores (or stream procs).
- Massive parallelism:
  - Applied to thousands of threads.
  - Sharing data at different levels.
- Heterogeneous computing:
  - GPU:
    - Data intensive.
    - Fine-grain parallelism.
  - CPU:
    - Control/management.
    - Coarse-grain parallelism.



	G80 (Tesla)	GT200 (Tesla)	GF100 (Fermi)	GK110 (Kepler)	(GM200) Maxwell
Period	2006-07	2008-09	2010-11	2012-13	2014-15
N (multip.)	16	30	14-16	13-15	4-24
M (cores/mult.)	8	8	32	192	128
# cores	128	240	448-512	2496-2880	512-3072

Memory hierarchy

- Each multiprocessor has:
  - A register file.
  - Shared memory.
  - A constant cache and a texture cache, both read-only.
- Global memory is the actual video memory (GDDR5):
  - Three times faster than the DDR3 used by the CPU, but...
  - ... around 500 times slower than shared memory! (DRAM versus SRAM).

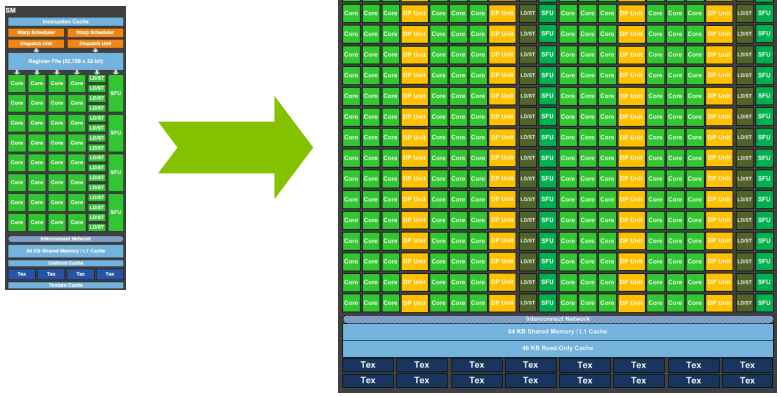


II. 2. The third generation: Kepler (GK1xx)

Kepler GK110 Block Diagram

- 7.1 billion transistors.
- 15 SMX multiprocs.
- > 1 TFLOP FP64.
- 1.5 MB L2 Cache.
- 384-bit GDDR5.
- PCI Express Gen3.

Multiprocessor evolution: From SMs in Fermi to SMXs in Kepler



The SMX multiprocessor

Instruction scheduling and issuing in **warps**

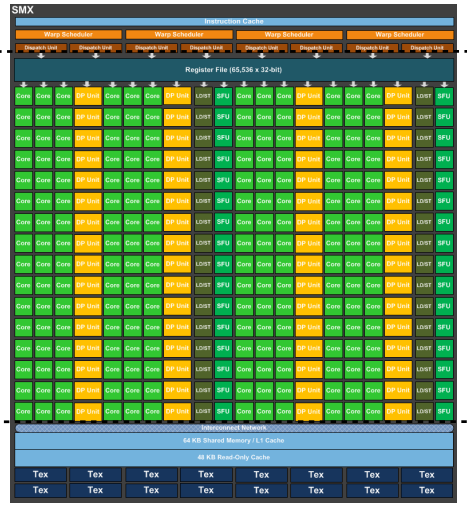
Front-end

Instructions execution.
512 functional units:
- 192 for ALUs.
- 192 for FPUs S.P.
- 64 for FPUs D.P.
- 32 for load/store.
- 32 for SFUs (log,sqrt, ...)

Back-end

Memory access

Interface



From SM multiprocessor in Fermi GF100 to SMX multiprocessor in Kepler GK110

The way GigaThread scheduling works

- Each grid provides a number of blocks, which are assigned to mult. (up to 32 blocks in Maxwell, 16 in Kepler, 8 in Fermi).
- Blocks are split into warps (groups) of 32 threads.
- Warps are issued for each instruction in kernel threads (up to 64 active warp-instructions in Kepler, 48 in Fermi). Ex:

II. 3. The fourth generation: Maxwell (GM1xx)

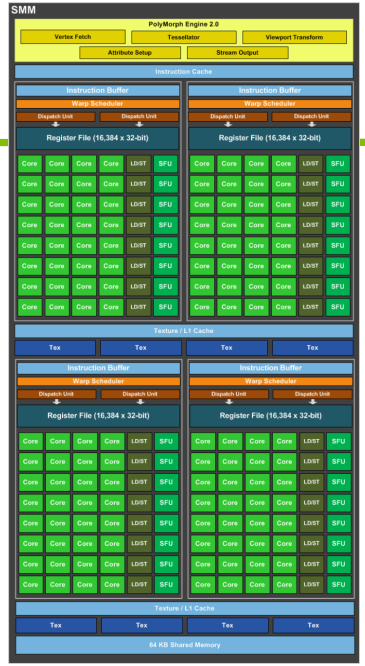
NVIDIA

Maxwell and SMM multiprocessors (for GeForce GTX 980, 16 SMMs)

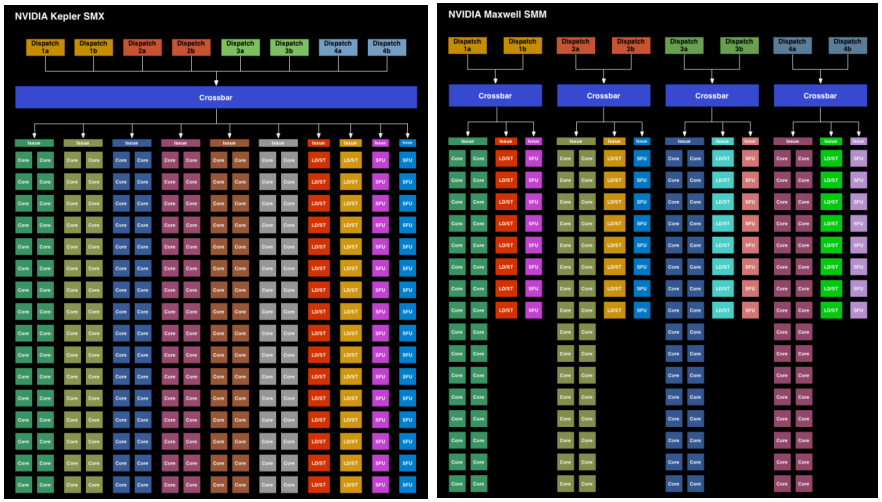
- 1870 Mt.
- 148 mm^2.

The SMMs

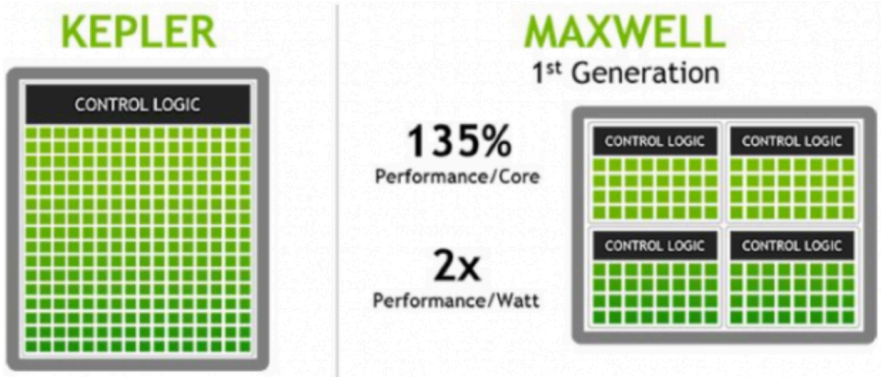
- Keep the same 4 warp schedulers, and the same LD/ST and SFU units.
- Reduce the number of cores for `int` and `float`: from 192 to 128 units.



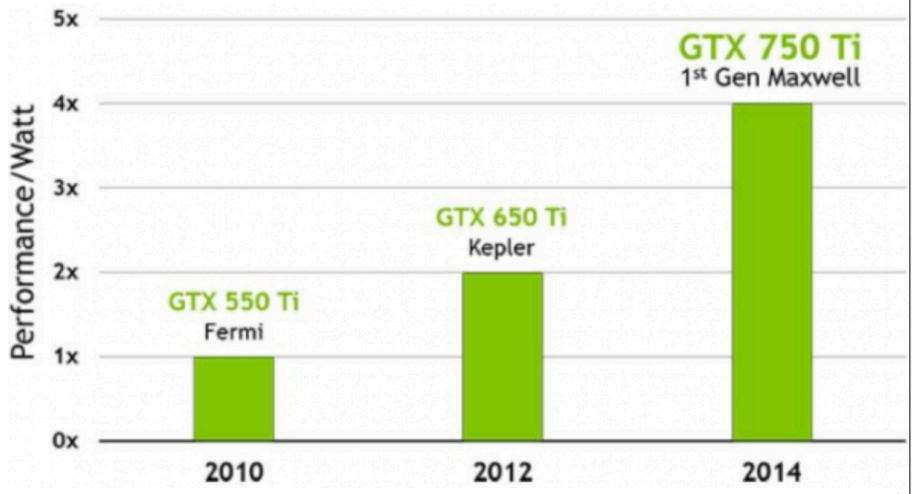
A comparison versus Kepler



Major enhancements



Power efficiency



II. 6. A summary of four generations

Scalability for the architecture: A summary of four generations

	Tesla		Fermi		Kepler				Maxwell	
Architecture	**G80**	**GT200**	**GF100**	**GF104**	**GK104 (K10)**	**GK110 (K20X)**	**GK110 (K40)**	**GK210 (K80)**	**GM107 (GTX750)**	**GM204 (GTX980)**
Time frame	**2006 /07**	**2008 /09**	**2010**	**2011**	**2012**	**2013**	**2013 /14**	**2014**	**2014 /15**	**2014 /15**
CUDA Compute Capability	**1.0**	**1.3**	**2.0**	**2.1**	**3.0**	**3.5**	**3.5**	**3.7**	**5.0**	**5.2**
N (multiprocs.)	16	30	16	7	8	14	15	30	5	16
M (cores/multip.)	8	8	32	48	192	192	192	192	128	128
Number of cores	128	240	512	336	1536	2688	2880	5760	640	2048