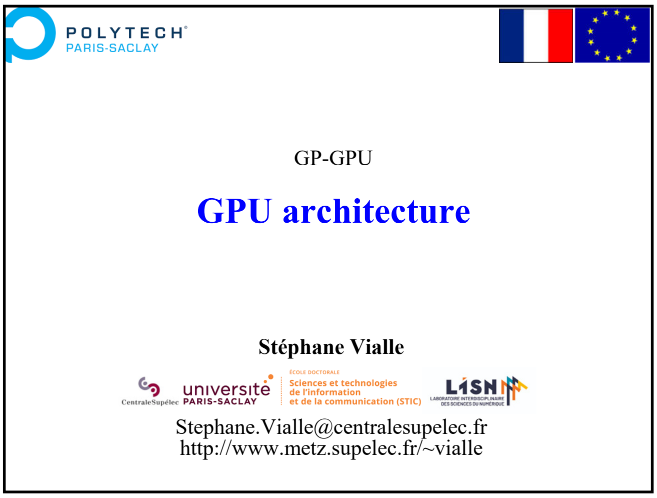


POLYTECH PARIS-SACLAY

GP-GPU

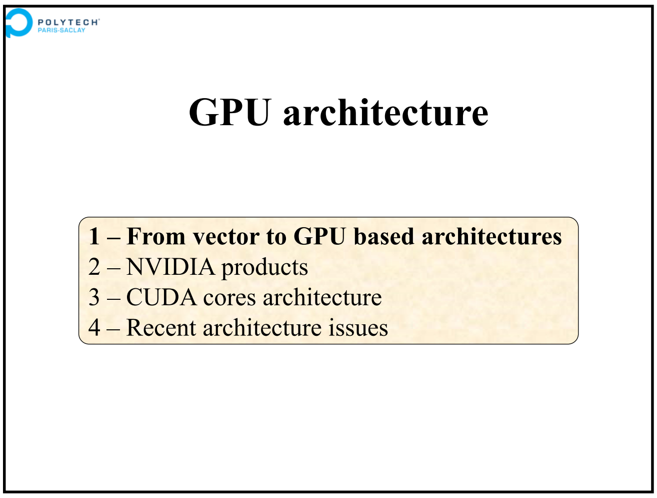
GPU architecture

Stéphane Vialle

CentraleSupélec · université PARIS-SACLAY · École doctorale Sciences et technologies de l'information et de la communication (STIC) · LISN Laboratoire interdisciplinaire des sciences du numérique

Stephane.Vialle@centralesupelec.fr
http://www.metz.supelec.fr/~vialle

1

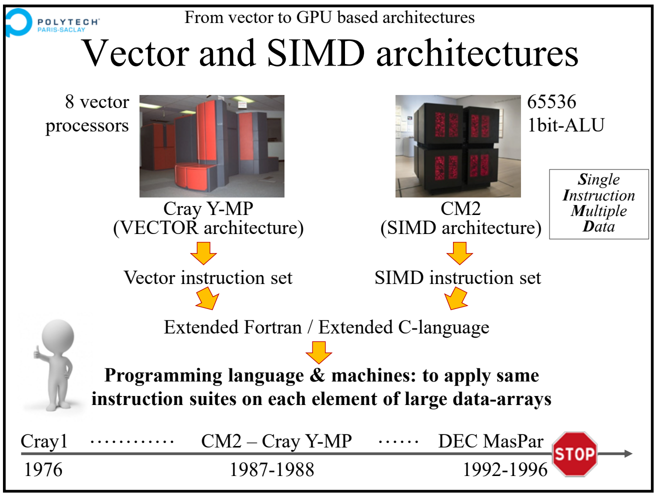


POLYTECH PARIS-SACLAY

GPU architecture

1 – From vector to GPU based architectures
2 – NVIDIA products
3 – CUDA cores architecture
4 – Recent architecture issues

2



From vector to GPU based architectures

Vector and SIMD architectures

8 vector processors

Cray Y-MP
(VECTOR architecture)

Vector instruction set

65536 1bit-ALU

CM2
(SIMD architecture)

Single Instruction Multiple Data

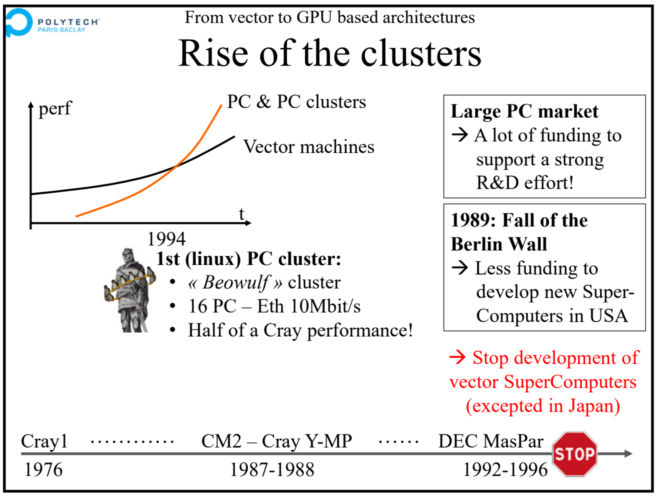
SIMD instruction set

Extended Fortran / Extended C-language

Programming language & machines: to apply same instruction suites on each element of large data-arrays

Cray1 ………… CM2 – Cray Y-MP …… DEC MasPar STOP
1976 1987-1988 1992-1996

3



From vector to GPU based architectures

Rise of the clusters

1st (linux) PC cluster:
- *« Beowulf »* cluster
- 16 PC – Eth 10Mbit/s
- Half of a Cray performance!

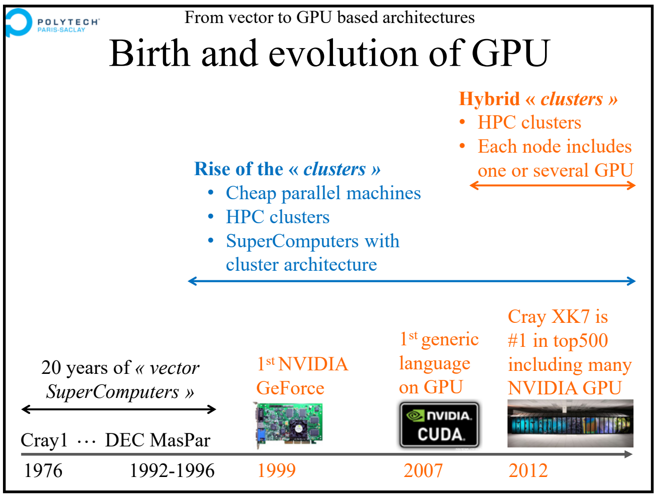
Large PC market
→ A lot of funding to support a strong R&D effort!

1989: Fall of the Berlin Wall
→ Less funding to develop new Super-Computers in USA

→ Stop development of vector SuperComputers (excepted in Japan)

Cray1 ………… CM2 – Cray Y-MP …… DEC MasPar STOP
1976 1987-1988 1992-1996

4



From vector to GPU based architectures

Birth and evolution of GPU

Hybrid *« clusters »*
- HPC clusters
- Each node includes one or several GPU

Rise of the *« clusters »*
- Cheap parallel machines
- HPC clusters
- SuperComputers with cluster architecture

20 years of *« vector SuperComputers »*

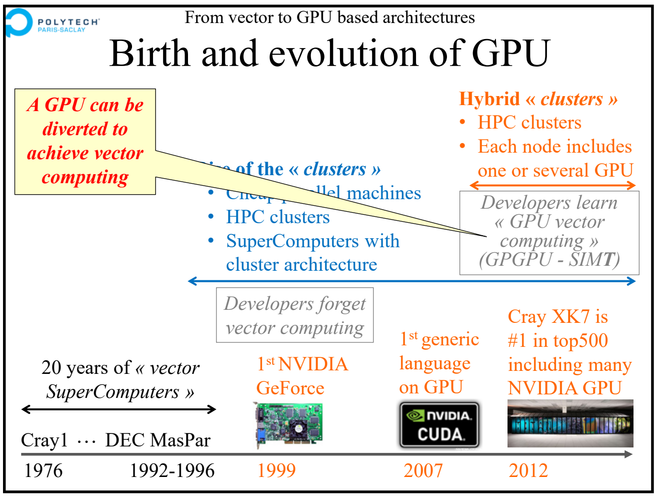
1st NVIDIA GeForce

1st generic language on GPU

Cray XK7 is #1 in top500 including many NVIDIA GPU

Cray1 ··· DEC MasPar
1976 1992-1996 1999 2007 2012

5



From vector to GPU based architectures

Birth and evolution of GPU

A GPU can be diverted to achieve vector computing

Hybrid *« clusters »*
- HPC clusters
- Each node includes one or several GPU

Developers learn « GPU vector computing » (GPGPU - SIMT)

…e of the *« clusters »*
- Cheap …llel machines
- HPC clusters
- SuperComputers with cluster architecture

Developers forget vector computing

20 years of *« vector SuperComputers »*

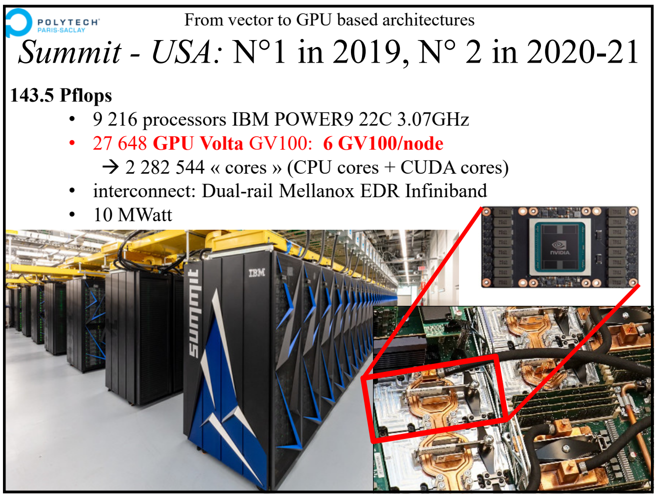
1st NVIDIA GeForce

1st generic language on GPU

Cray XK7 is #1 in top500 including many NVIDIA GPU

Cray1 ··· DEC MasPar
1976 1992-1996 1999 2007 2012

6



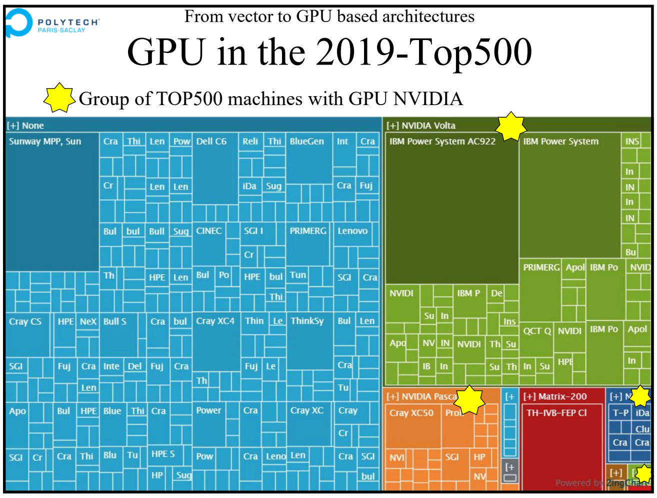
From vector to GPU based architectures

Summit - USA: N°1 in 2019, N° 2 in 2020-21

143.5 Pflops

- 9 216 processors IBM POWER9 22C 3.07GHz
- 27 648 **GPU Volta** GV100: **6 GV100/node**
  - → 2 282 544 « cores » (CPU cores + CUDA cores)
- interconnect: Dual-rail Mellanox EDR Infiniband
- 10 MWatt

7

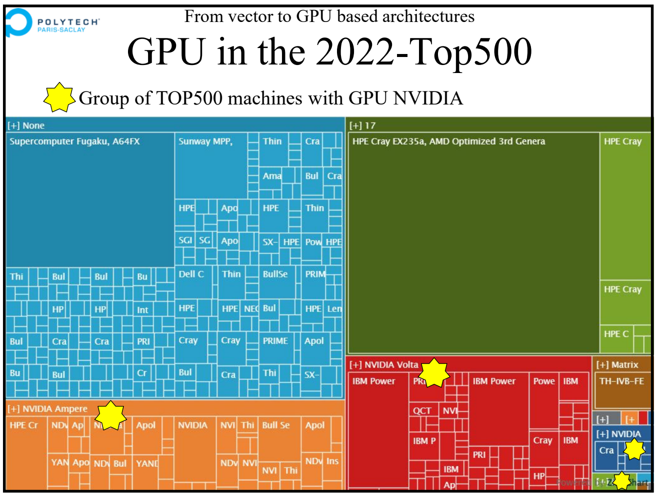


From vector to GPU based architectures

GPU in the 2019-Top500

Group of TOP500 machines with GPU NVIDIA

8

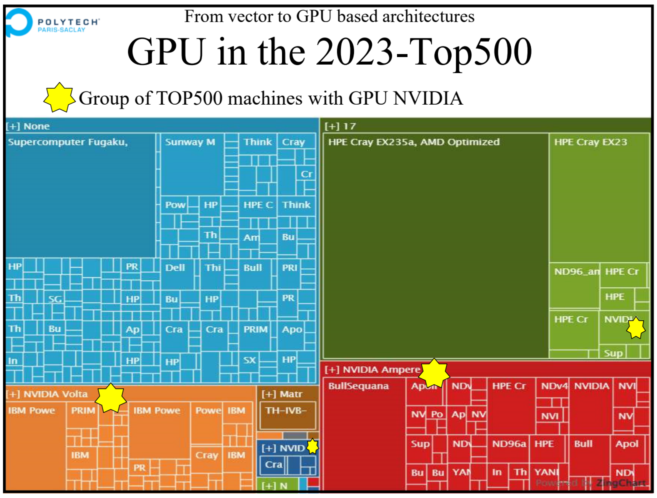


From vector to GPU based architectures

GPU in the 2022-Top500

Group of TOP500 machines with GPU NVIDIA

9



From vector to GPU based architectures

GPU in the 2023-Top500

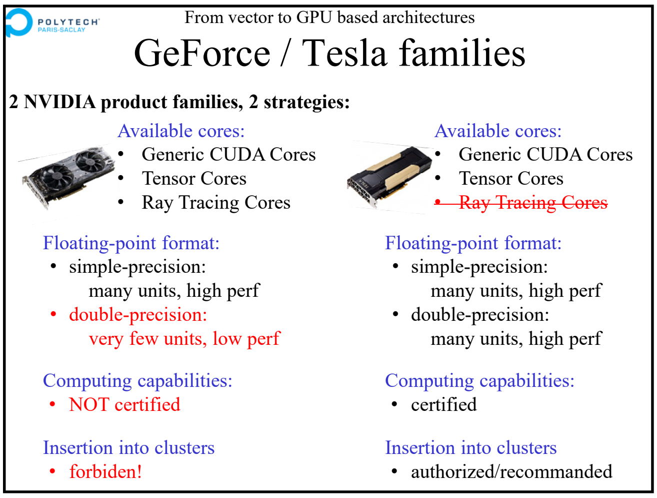
Group of TOP500 machines with GPU NVIDIA

10

GPU architecture

1 – From vector to GPU based architectures
2 – NVIDIA products
3 – CUDA cores architecture
4 – Recent architecture issues

11



From vector to GPU based architectures

GeForce / Tesla families

2 NVIDIA product families, 2 strategies:

Available cores:
- Generic CUDA Cores
- Tensor Cores
- Ray Tracing Cores

Floating-point format:
- simple-precision:
  many units, high perf
- double-precision:
  very few units, low perf

Computing capabilities:
- NOT certified

Insertion into clusters
- forbiden!

Available cores:
- Generic CUDA Cores
- Tensor Cores
- ~~Ray Tracing Cores~~

Floating-point format:
- simple-precision:
  many units, high perf
- double-precision:
  many units, high perf

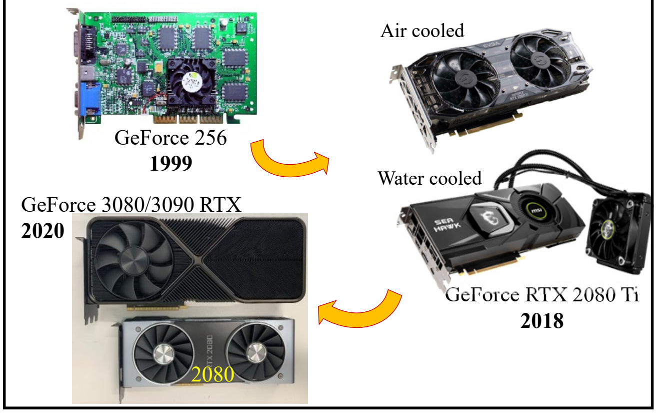
Computing capabilities:
- certified

Insertion into clusters
- authorized/recommanded

12

From vector to GPU based architectures

GeForce packaging



GeForce 256
1999

Air cooled

Water cooled

GeForce RTX 2080 Ti
2018

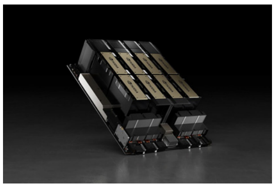
GeForce 3080/3090 RTX
2020

2080

13

From vector to GPU based architectures

Tesla H100 packaging (V100 → A100 → H100)



V100 has been a 3 billion dollar R&D project ….
A100 ? … H100 ?

14

GPU architecture

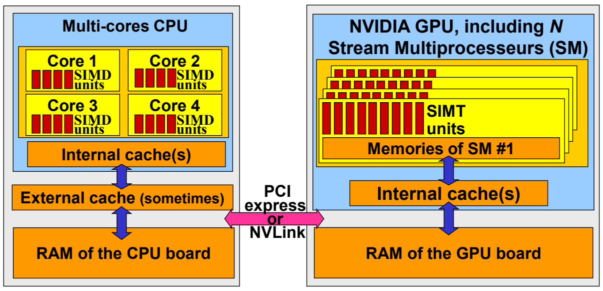
1 – From vector to GPU based architectures
2 – NVIDIA products
3 – CUDA cores architecture
4 – Recent architecture issues

15

CUDA cores architecture

CPU-GPU overview

- Both CPU and GPU are « multi-cores » with « hierarchical memories »



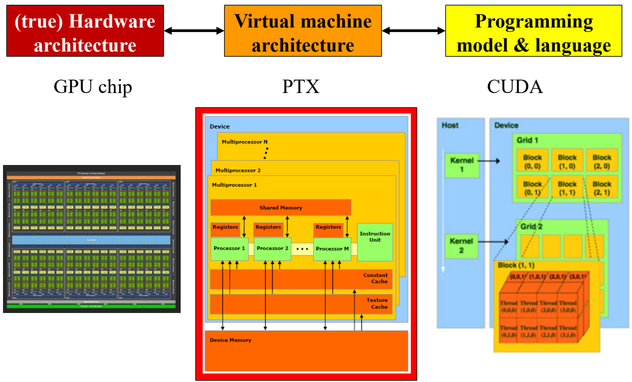
- CPU and GPU boards communicate across a fast NVLink, or a std (and slow) PCI-eXpress

→ CPU uses GPU as a *scientific coprocessor* for *vector/SIMD computing*

16

CUDA cores architecture

PTX virtual machine



In this chapter

17

CUDA cores architecture

Stream Multiprocessors architecture

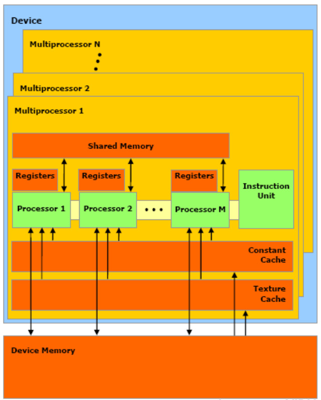
A GPU is a set of *N* Stream multiprocessors (SM):

- *N* independent « SIMT » machines
- Sharing the GPU board memory

Before Volta & Turing architectures one SM included:

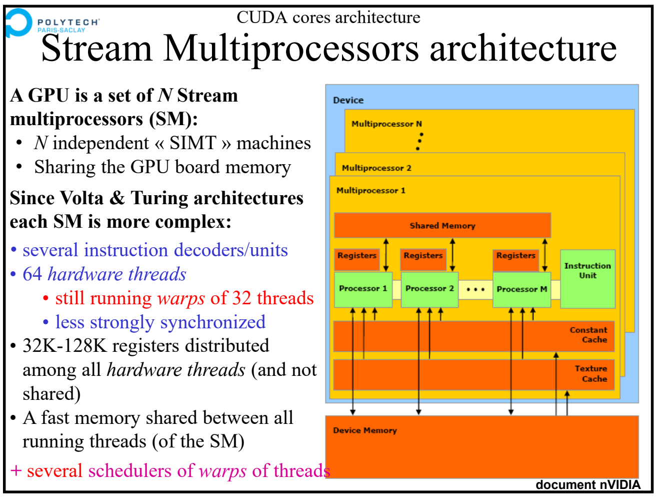
- 1 instruction decoder/unit
- 32 strongly synchronized *hardware threads*, running *warps* of 32 threads
- 32K-128K registers distributed among all *hardware threads* (and not shared)
- A fast memory shared between all running threads (of the SM)

+ A scheduler of warps of threads, among a larger block of threads



document nVIDIA

18



CUDA cores architecture

Stream Multiprocessors architecture

A GPU is a set of *N* Stream multiprocessors (SM):

- *N* independent « SIMT » machines
- Sharing the GPU board memory

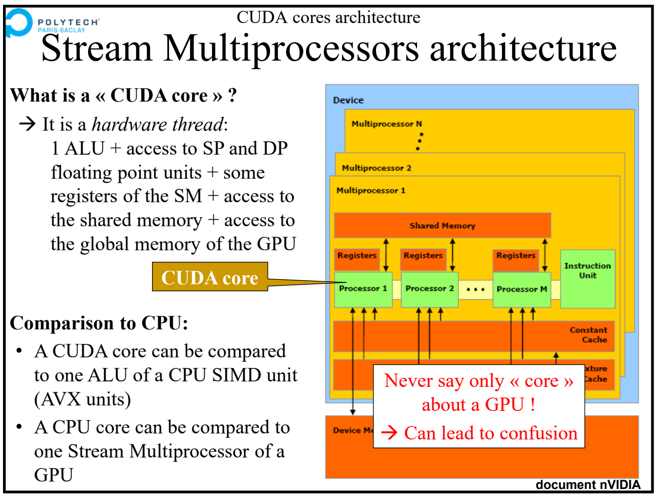
Since Volta & Turing architectures each SM is more complex:

- several instruction decoders/units
- 64 *hardware threads*
  - still running *warps* of 32 threads
  - less strongly synchronized
- 32K-128K registers distributed among all *hardware threads* (and not shared)
- A fast memory shared between all running threads (of the SM)

+ several schedulers of *warps* of threads

document nVIDIA

19



CUDA cores architecture

Stream Multiprocessors architecture

What is a « CUDA core » ?

→ It is a *hardware thread*:
1 ALU + access to SP and DP floating point units + some registers of the SM + access to the shared memory + access to the global memory of the GPU

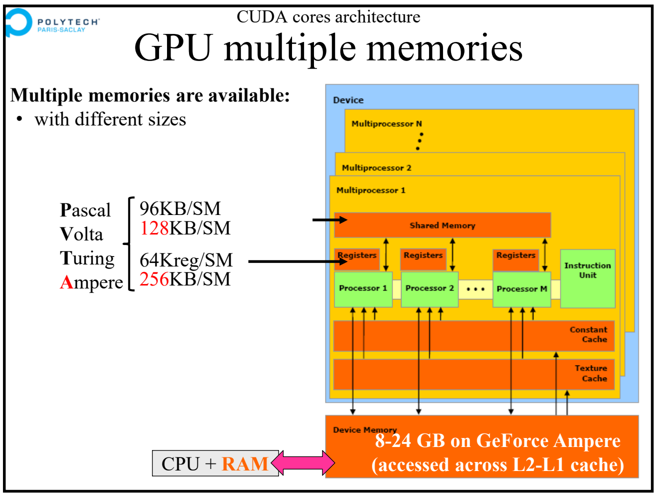
Comparison to CPU:

- A CUDA core can be compared to one ALU of a CPU SIMD unit (AVX units)
- A CPU core can be compared to one Stream Multiprocessor of a GPU

Never say only « core » about a GPU !
→ Can lead to confusion

document nVIDIA

20



CUDA cores architecture

GPU multiple memories

Multiple memories are available:

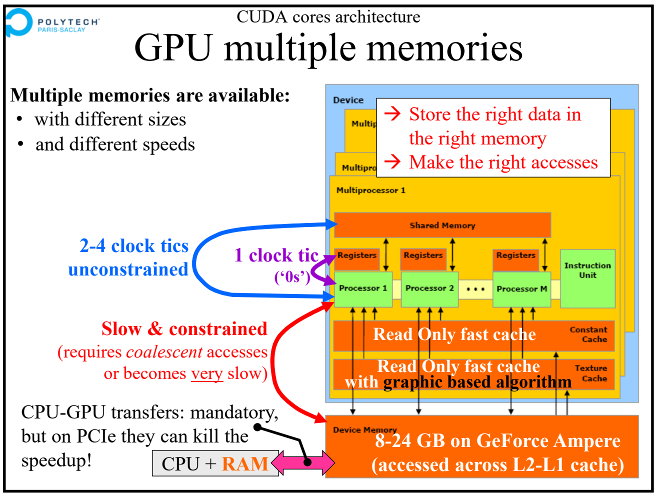
- with different sizes

Pascal	96KB/SM
Volta	128KB/SM
Turing	64Kreg/SM
Ampere	256KB/SM

CPU + RAM

8-24 GB on GeForce Ampere (accessed across L2-L1 cache)

21



CUDA cores architecture

GPU multiple memories

Multiple memories are available:

- with different sizes
- and different speeds

→ Store the right data in the right memory
→ Make the right accesses

Slow & constrained (requires *coalescent* accesses or becomes very slow)

Read Only fast cache

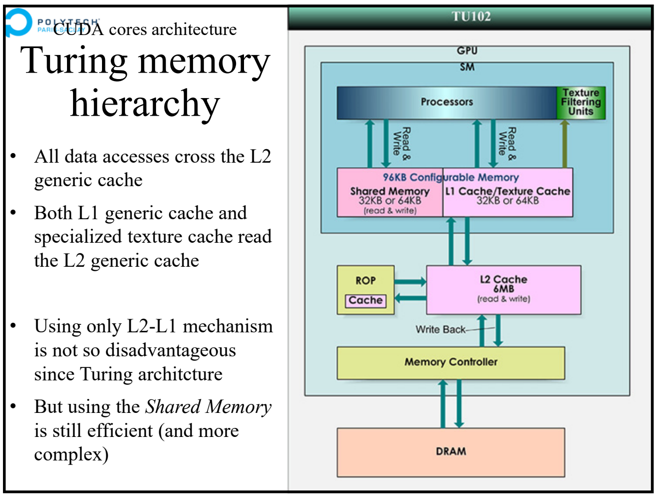
Read Only fast cache with graphic based algorithm

CPU-GPU transfers: mandatory, but on PCIe they can kill the speedup!

CPU + RAM

8-24 GB on GeForce Ampere (accessed across L2-L1 cache)

22



CUDA cores architecture

Turing memory hierarchy

- All data accesses cross the L2 generic cache
- Both L1 generic cache and specialized texture cache read the L2 generic cache
- Using only L2-L1 mechanism is not so disadvantageous since Turing architcture
- But using the *Shared Memory* is still efficient (and more complex)

23

GPU architecture

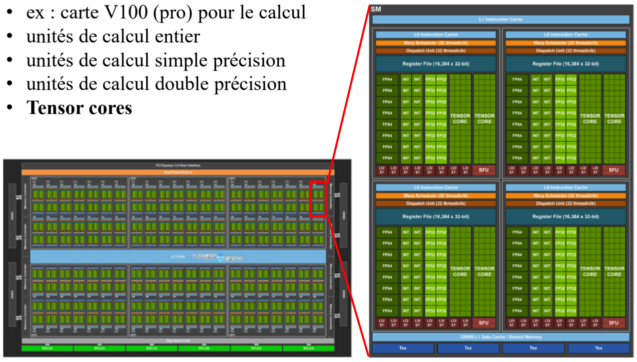
1 – From vector to GPU based architectures
2 – NVIDIA products
3 – CUDA cores architecture
4 – Recent architecture issues

24

Recent architecture issues

New Tesla GPU

Volta architecture:
- ex : carte V100 (pro) pour le calcul
- unités de calcul entier
- unités de calcul simple précision
- unités de calcul double précision
- **Tensor cores**


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
SFU
128KB L1 Data Cache / Shared Memory
Tex

1 Stream Multiprocessor

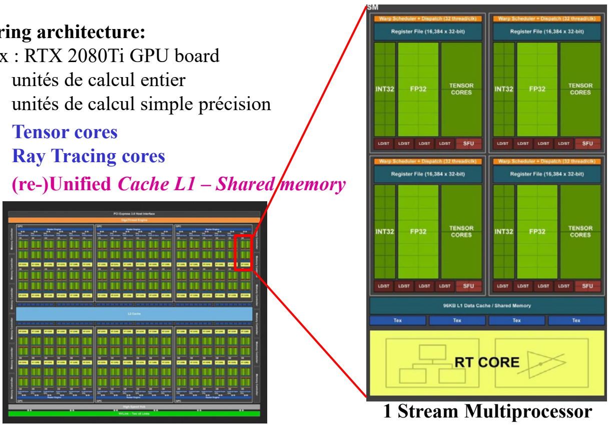
25

Recent architecture issues

New GeForce GPU

Turing architecture:

Ex : RTX 2080Ti GPU board
- unités de calcul entier
- unités de calcul simple précision
- **Tensor cores**
- **Ray Tracing cores**
- **(re-)Unified *Cache L1 – Shared memory***


SM
Warp Scheduler + Dispatch (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
TENSOR CORES
LD/ST
SFU
96KB L1 Data Cache / Shared Memory
Tex
RT CORE

1 Stream Multiprocessor

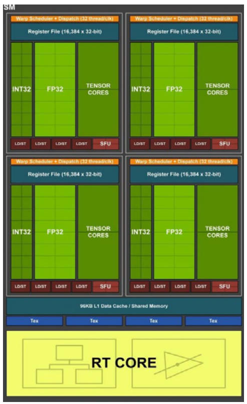
26

Recent architecture issues

New GeForce GPU

Turing architecture:

Ex : RTX 2080Ti GPU board
- 72 Stream Multiprocessor (SM)
- 64 CUDA cores / SM
→ 4608 CUDA cores
- Double-precision computing is possible but slow
- 8 Tensor cores / SM
→ 576 Tensor cores
- 1 Ray Tracing core / SM
→ 72 RT cores
- More efficient cache memory


SM
Warp Scheduler + Dispatch (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
TENSOR CORES
LD/ST
SFU
96KB L1 Data Cache / Shared Memory
Tex
RT CORE

1 Stream Multiprocessor

27

Recent architecture issues

Motivation to design RT cores

Ray Tracing cores:
- Final objective: « real time ray tracing for video »
- Currently: GPU not powerful enough
→ Real Time RT on a subset of rays
+ interpolation with Tensor Cores

Video game remains the main market for NVIDIA

→ GPU architecture evolutions must be useful for the video game market

SOL MAN from NVIDIA SOL ray tracing demo running on a Turing TU102 GPU with NVIDIA RTX technology in real-time

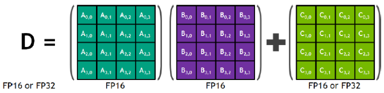
28

Recent architecture issues

Tensor Core features

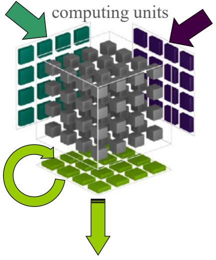
Tensor cores:

1 TC achieves a flow of product-add on a flow of 4x4 matrixes
- D = A.B : produces a flow of D output matrixes
- D = A.B + C, with accumulation of A.B product flow into C matrix



$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

64 specialized computing units



Possible mixed-precision:
- input matrixes encoded on 16 bits
- internal computing on 32 bits
- output matrix flow on 16 or 32 bits

Useful for many kinds of applications: image processing, Machine Learning, Linear Algebra…

29

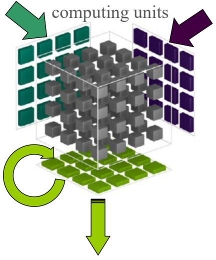
Recent architecture issues

Tensor Core features

Tensor cores:

1 TC achieves a flow of product-add on a flow of 4x4 matrixes
- D = A.B : produces a flow of D output matrixes
- D = A.B + C, with accumulation of A.B product flow into C matrix

64 specialized computing units



A Tensor core:
- is a hardware implementation of a matrix operator,
- is a very useful operator for modern applications,
- including graphic applications (main GPU market).

→ A mathematical operator whose genericity justifies that it occupies a part of the chip!

30

Recent architecture issues

Cache memory improvement

New fast memory:

- 96 or 128 KBytes per SM
- Used for both: L1 cache, shared memory, texture cache

Rmk: The « shared memory » is an unmanaged L1 cache memory. The application developer has to design and implement a strategy adapted to its computations!

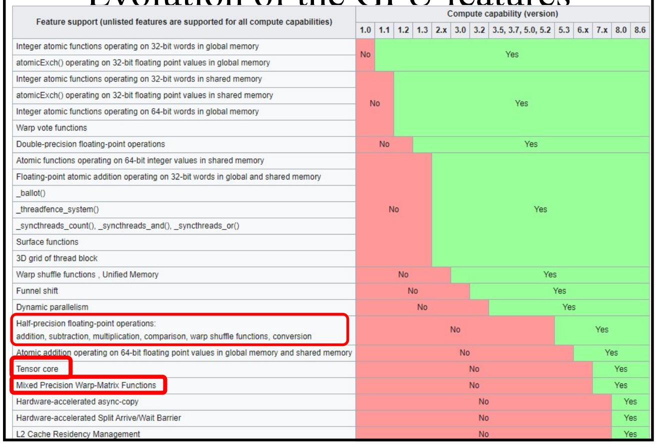
- If the shared memory is unused, the 96 KBytes will be automatically used for L1 cache
- A new and more efficient cache management strategy has been implemented

Objectives of this new fast memory architecture and management:

- To decrease the performance loss when not using shared memory…
- … many users have refused to design and implement a new cache management strategy (too difficult).

Recent architecture issues

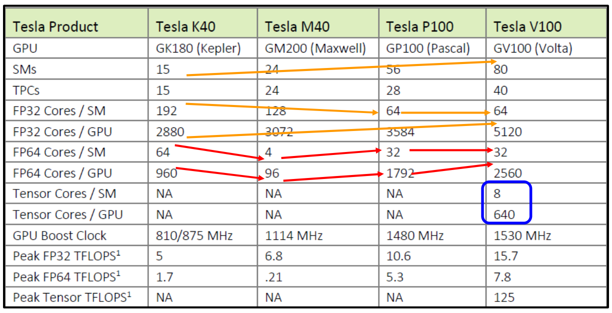
Evolution of the GPU features



<table>
<tr><th rowspan="2">Feature support (unlisted features are supported for all compute capabilities)</th><th colspan="13">Compute capability (version)</th></tr>
<tr><th>1.0</th><th>1.1</th><th>1.2</th><th>1.3</th><th>2.x</th><th>3.0</th><th>3.2</th><th>3.5, 3.7, 5.0, 5.2</th><th>5.3</th><th>6.x</th><th>7.x</th><th>8.0</th><th>8.6</th></tr>
<tr><td>Integer atomic functions operating on 32-bit words in global memory</td><td rowspan="2">No</td><td rowspan="2" colspan="12">Yes</td></tr>
<tr><td>atomicExch() operating on 32-bit floating point values in global memory</td></tr>
<tr><td>Integer atomic functions operating on 32-bit words in shared memory</td><td rowspan="4" colspan="2">No</td><td rowspan="4" colspan="11">Yes</td></tr>
<tr><td>atomicExch() operating on 32-bit floating point values in shared memory</td></tr>
<tr><td>Integer atomic functions operating on 64-bit words in global memory</td></tr>
<tr><td>Warp vote functions</td></tr>
<tr><td>Double-precision floating-point operations</td><td colspan="3">No</td><td colspan="10">Yes</td></tr>
<tr><td>Atomic functions operating on 64-bit integer values in shared memory</td><td rowspan="7" colspan="4">No</td><td rowspan="7" colspan="9">Yes</td></tr>
<tr><td>Floating-point atomic addition operating on 32-bit words in global and shared memory</td></tr>
<tr><td>_ballot()</td></tr>
<tr><td>_threadfence_system()</td></tr>
<tr><td>_syncthreads_count(), _syncthreads_and(), _syncthreads_or()</td></tr>
<tr><td>Surface functions</td></tr>
<tr><td>3D grid of thread block</td></tr>
<tr><td>Warp shuffle functions , Unified Memory</td><td colspan="5">No</td><td colspan="8">Yes</td></tr>
<tr><td>Funnel shift</td><td colspan="6">No</td><td colspan="7">Yes</td></tr>
<tr><td>Dynamic parallelism</td><td colspan="7">No</td><td colspan="6">Yes</td></tr>
<tr><td>Half-precision floating-point operations:
addition, subtraction, multiplication, comparison, warp shuffle functions, conversion</td><td colspan="8">No</td><td colspan="5">Yes</td></tr>
<tr><td>Atomic addition operating on 64-bit floating point values in global memory and shared memory</td><td colspan="9">No</td><td colspan="4">Yes</td></tr>
<tr><td>Tensor core</td><td colspan="10">No</td><td colspan="3">Yes</td></tr>
<tr><td>Mixed Precision Warp-Matrix Functions</td><td colspan="10">No</td><td colspan="3">Yes</td></tr>
<tr><td>Hardware-accelerated async-copy</td><td colspan="11">No</td><td colspan="2">Yes</td></tr>
<tr><td>Hardware-accelerated Split Arrive/Wait Barrier</td><td colspan="11">No</td><td colspan="2">Yes</td></tr>
<tr><td>L2 Cache Residency Management</td><td colspan="11">No</td><td colspan="2">Yes</td></tr>
</table>

Recent architecture issues

Evolution of the GPU Tesla



Tesla Product	Tesla K40	Tesla M40	Tesla P100	Tesla V100
GPU	GK180 (Kepler)	GM200 (Maxwell)	GP100 (Pascal)	GV100 (Volta)
SMs	15	24	56	80
TPCs	15	24	28	40
FP32 Cores / SM	192	128	64	64
FP32 Cores / GPU	2880	3072	3584	5120
FP64 Cores / SM	64	4	32	32
FP64 Cores / GPU	960	96	1792	2560
Tensor Cores / SM	NA	NA	NA	8
Tensor Cores / GPU	NA	NA	NA	640
GPU Boost Clock	810/875 MHz	1114 MHz	1480 MHz	1530 MHz
Peak FP32 TFLOPS[1]	5	6.8	10.6	15.7
Peak FP64 TFLOPS[1]	1.7	.21	5.3	7.8
Peak Tensor TFLOPS[1]	NA	NA	NA	125

Some architecture features do not evolve monotonously

GPU architecture

End