Quels facteurs de rapidité de calcul ?

- Computing unit : nombre d’opérations par seconde
- Memory unit : vitesse d’accès mémoire
- Communication : vitesse de transfert des données

Coté processeur (computing unit)

- Ce qu’on veut améliorer:
  - Le nombre d’opérations par secondes
- Deux principaux paramètres de contrôle :
  - nombre d’opérations (instructions) par cycle (IPC)
  - nombre de cycles par seconde (clock speed)

- Limitations :
  - physiques, transistors trop petits, chauffage trop important
    - Consommation électrique : fréquence^3

	mono-cœur	bi-cœurs
Fréquence	F	0.75F
Consommation par processeur	W	0.84 W
Performance par processeur	P	1.5P

Coté mémoire (memory units)

- Ce qu’on veut améliorer :
  - Diminuer la *latency* (temps d’accès à l’information, exprimée en temps ou nombre de cycles)

- Exemples :
    - Disque dur à tête de lecture
    - Solid state hard drive
    - RAM
    - L1/L2 cache
    - Registres (~1 cycle)

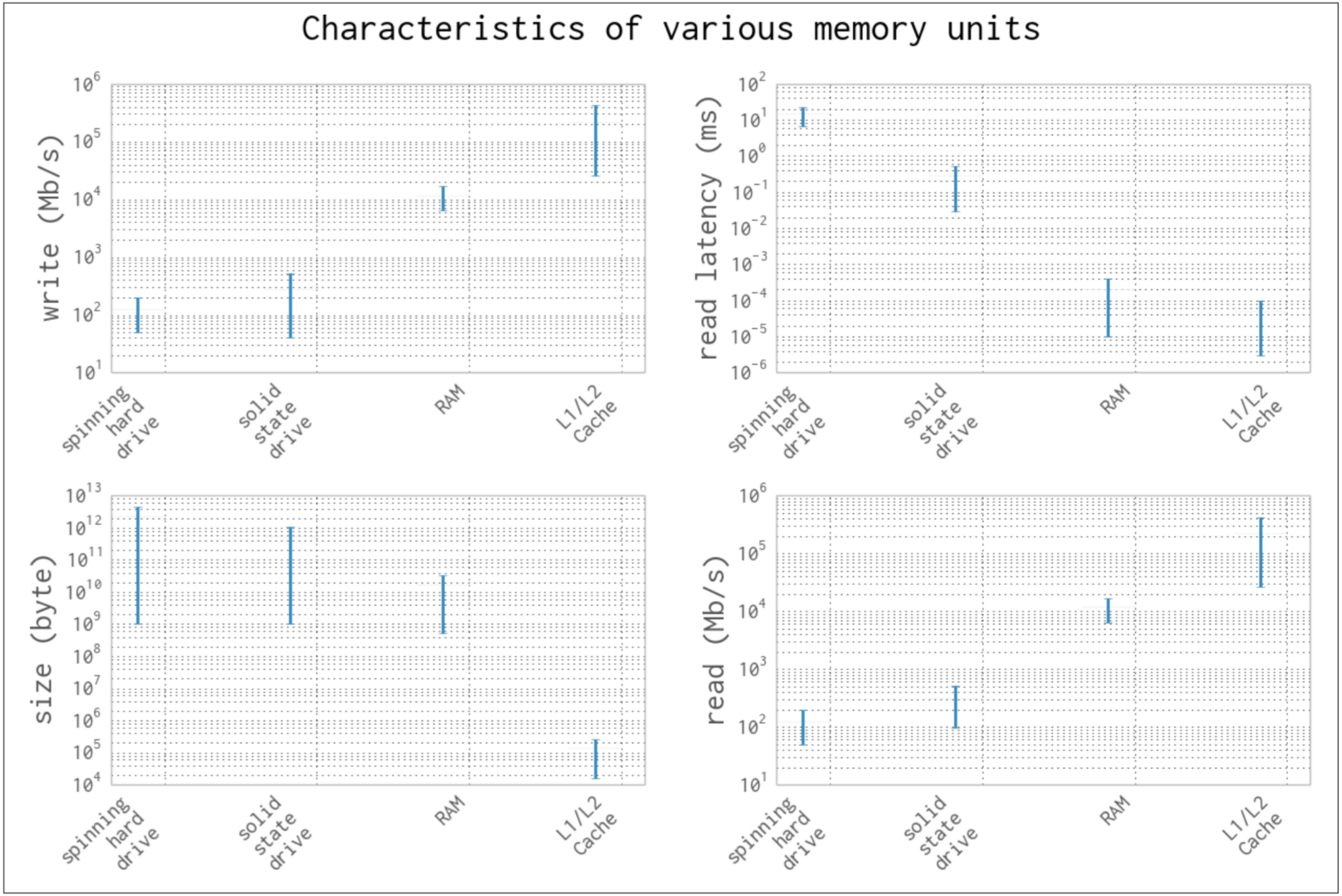
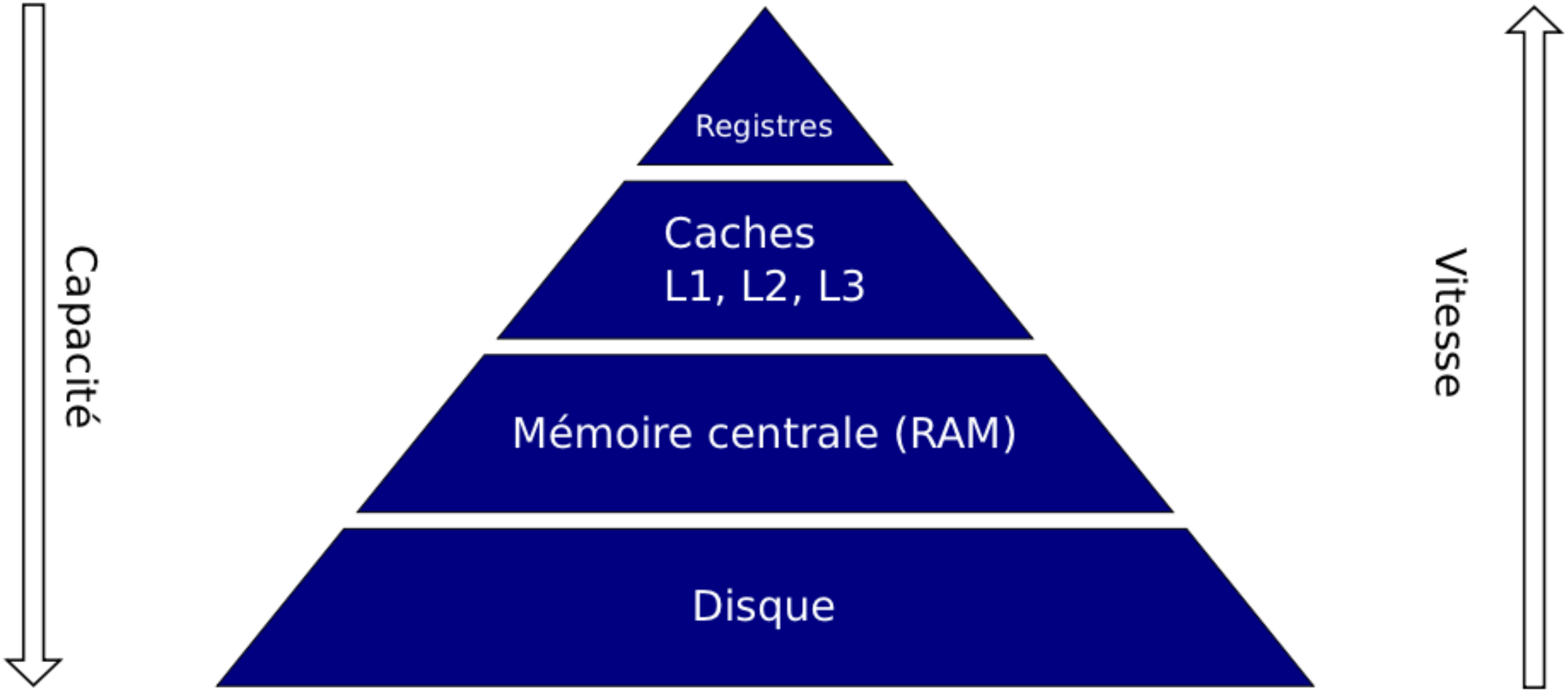


Figure 1-2. Characteristic values for different types of memory units (values from February 2014)

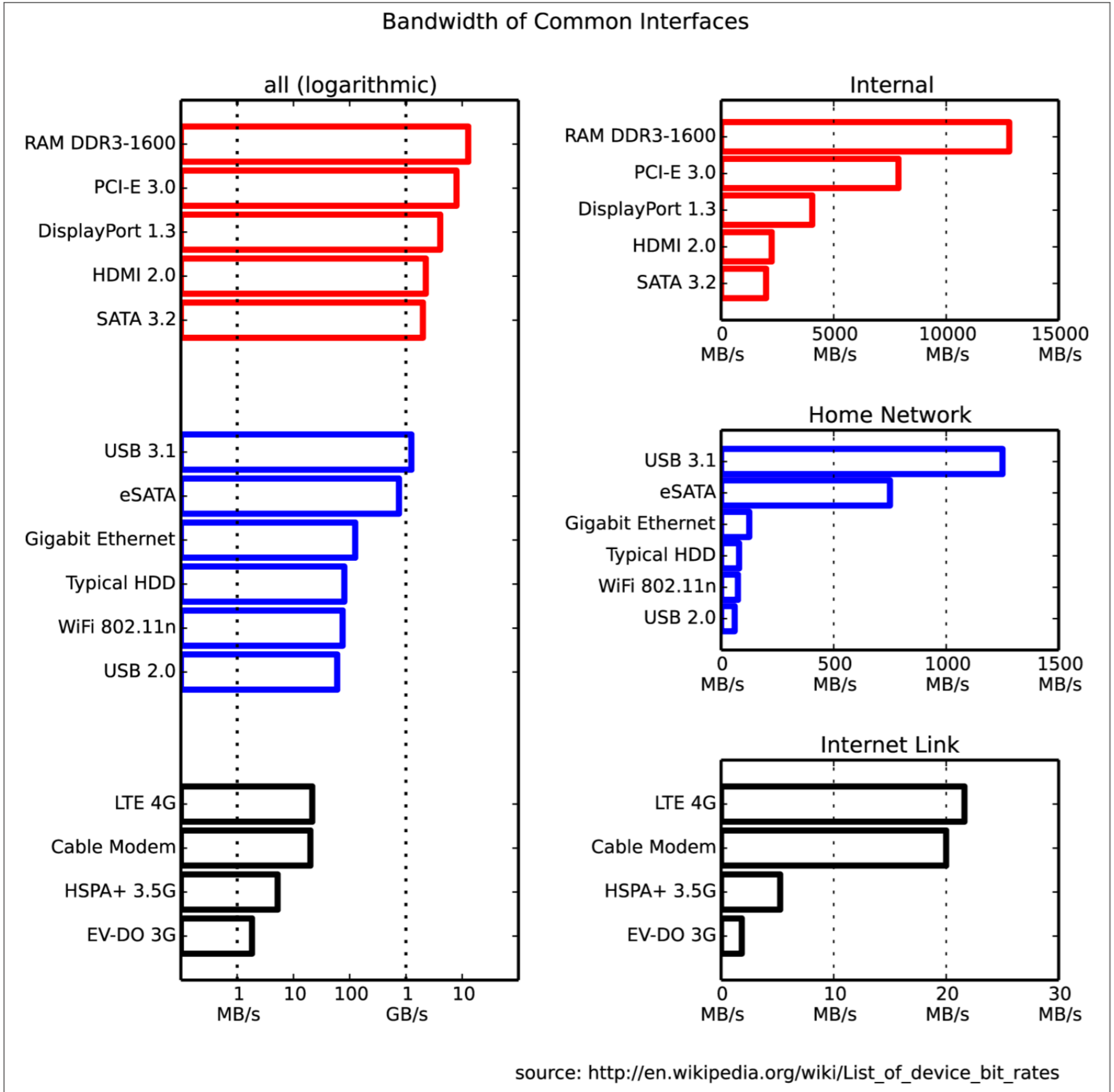
In High Performance Python


Registres
Caches
L1, L2, L3
Mémoire centrale (RAM)
Disque
Capacité
Vitesse

Communication

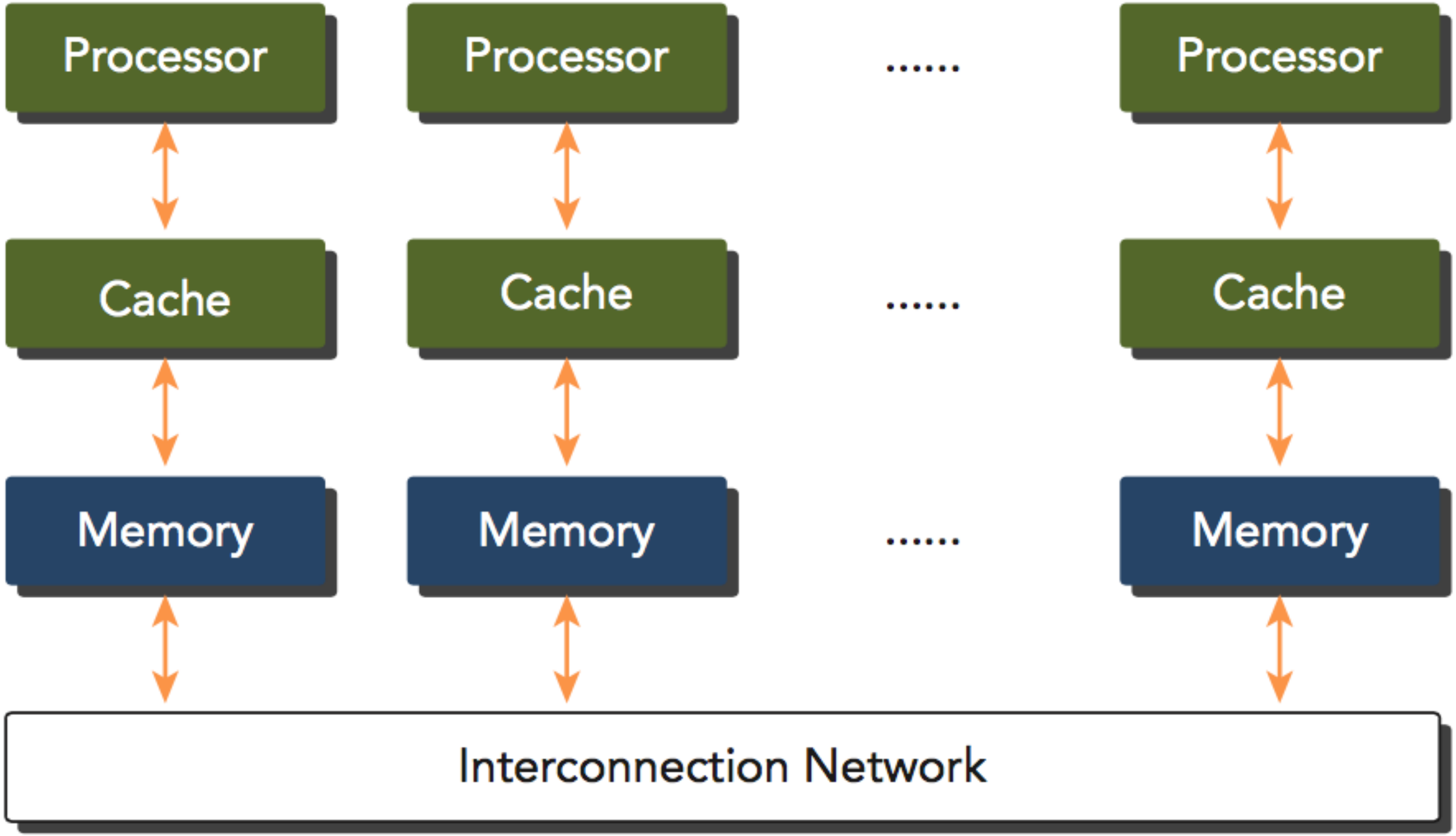
- Ce qu’on veut améliorer :
  - Data transfer rate (*bits*) from memory components to computing units
- Two main parameters :
  - *Bus width* : amount of bit that can be moved at once
  - *Bus frequency* : how many times per second

- Exemples :
  - frontside bus (between RAM et caches)
  - backside bus (between caches et CPU)
  - external bus (autres périphériques, carte réseau…)


Bandwidth of Common Interfaces
all (logarithmic)
RAM DDR3-1600
PCI-E 3.0
DisplayPort 1.3
HDMI 2.0
SATA 3.2
USB 3.1
eSATA
Gigabit Ethernet
Typical HDD
WiFi 802.11n
USB 2.0
LTE 4G
Cable Modem
HSPA+ 3.5G
EV-DO 3G
1 MB/s
10
100
1 GB/s
10
Internal
0 MB/s
5000 MB/s
10000 MB/s
15000 MB/s
Home Network
0 MB/s
500 MB/s
1000 MB/s
1500 MB/s
Internet Link
0 MB/s
10 MB/s
20 MB/s
30 MB/s
source: http://en.wikipedia.org/wiki/List_of_device_bit_rates

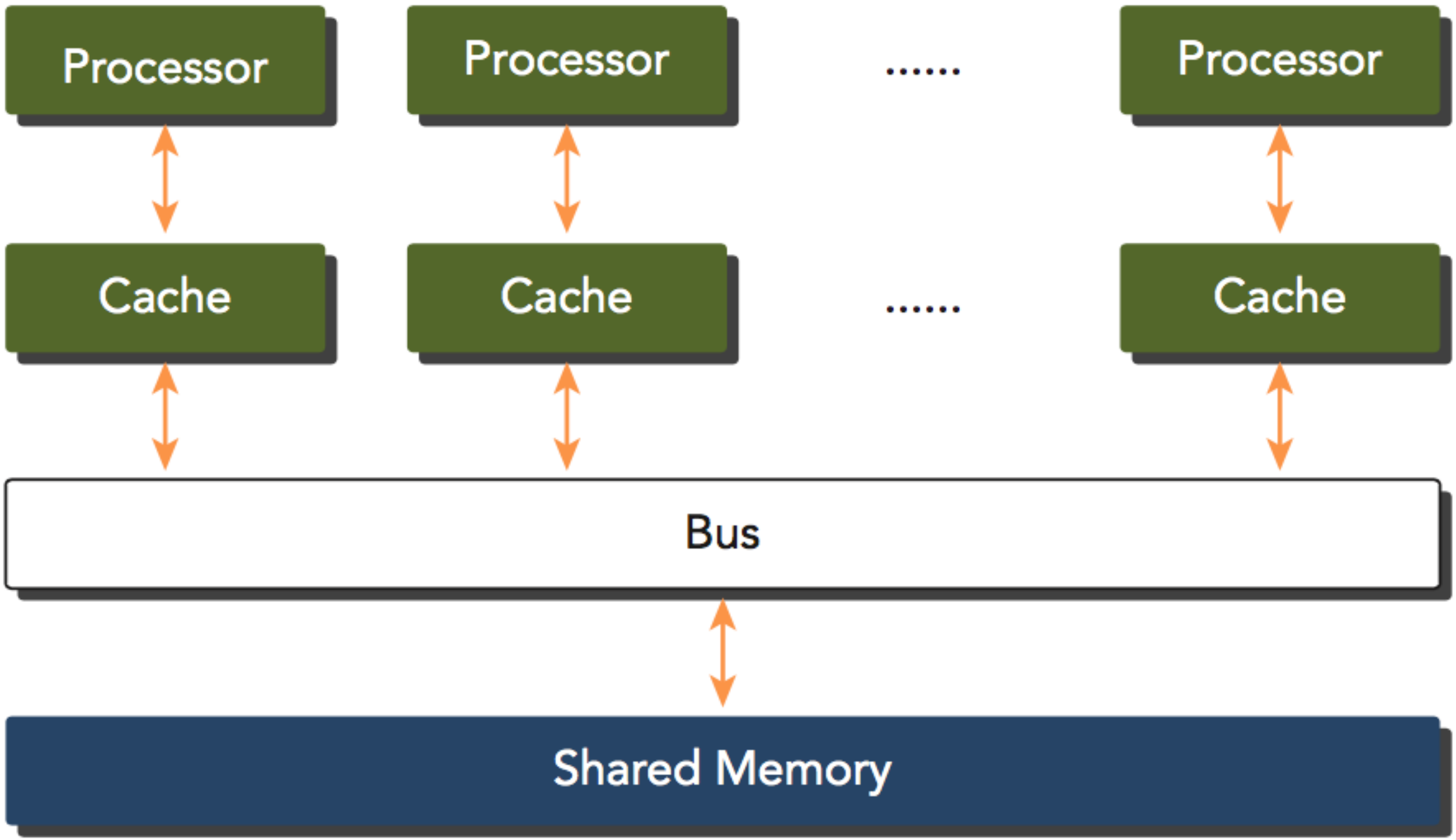
Architectures parallèles

Clusters



Architectures parallèles

Shared memory

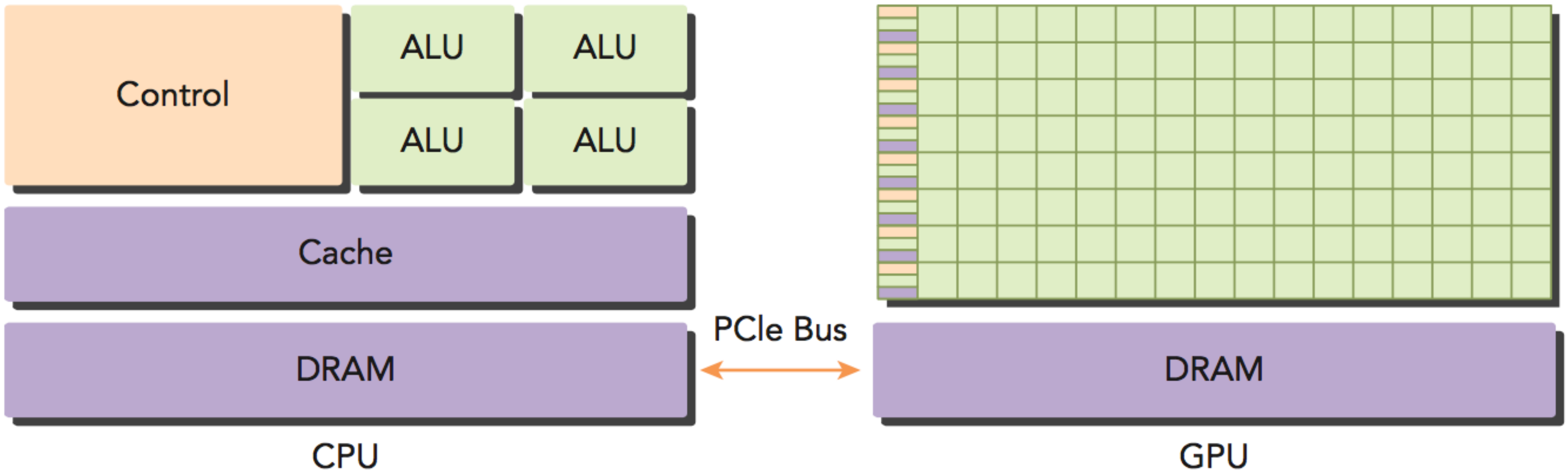


Architectures parallèles

Multiprocessors

- Processors can consist of several cores placed on one or several chips
- Many-core : lots of cores (tens or hundreds)
- GPU : many-core. More than SIMD. Called *SIMT*.
  - Threads can have different paths.
  - Threads have their own register state.

Heterogeneous architecture : master-slave control



Concepts

- Latency = time to perform an operation (data fetch or computation)
- Throughput = number of operations per sec : Gflops, Tflops…
- Bandwidth =data processed per time: Mb/s, Gb/s…

Parallel computation : software side

- Assign tasks to different threads.

- Two main strategies for handling data : block & cyclic partition.

Block partition: each thread takes one data block

Cyclic partition: each thread takes two data blocks

Pros and cons of GPUs

Pros

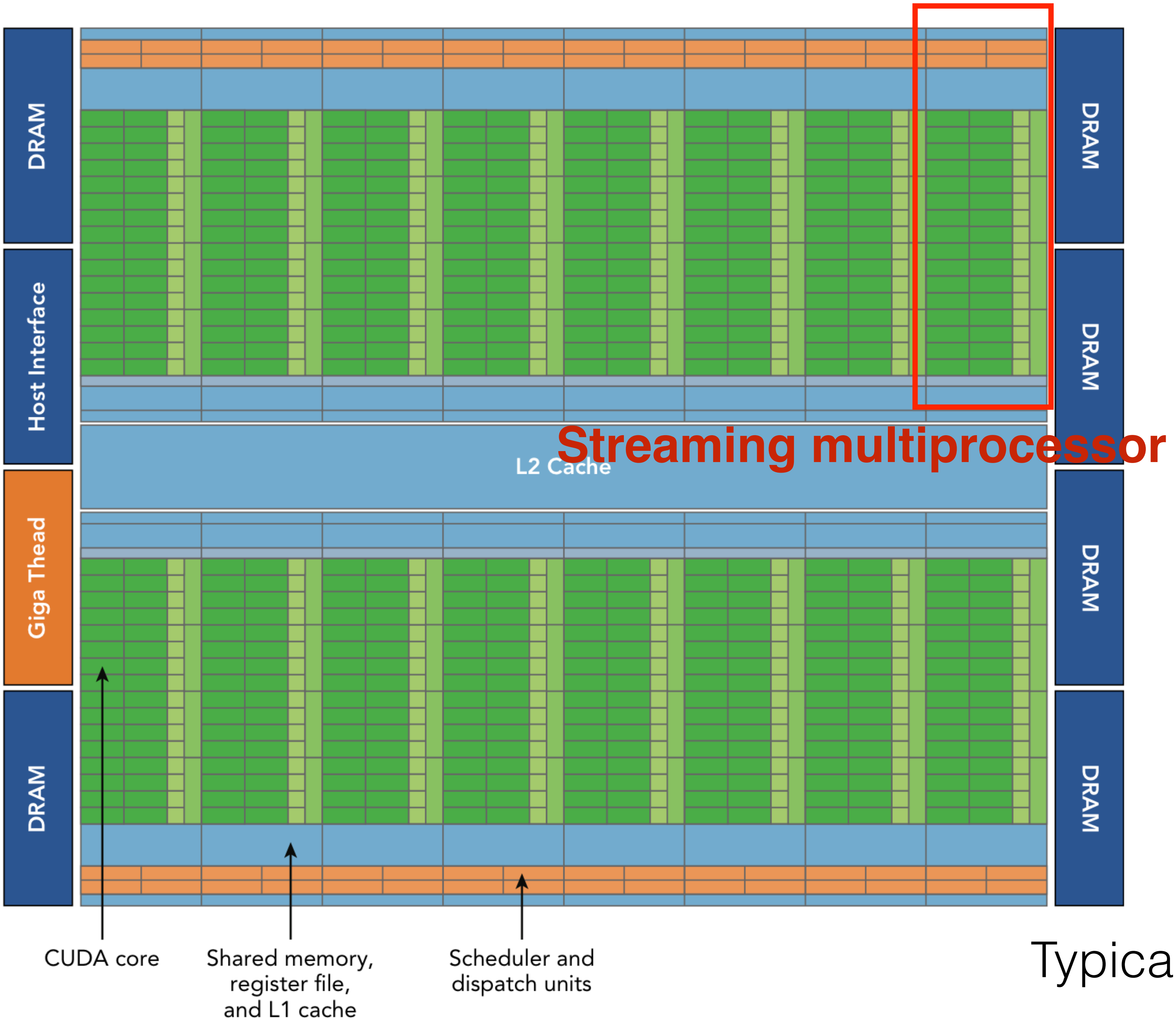
- 9 x less space
- 7 x less electrical consumption
- 6 x less expensive

High Performance Computing : are GPU going to take over ?", A. Nedelcoux, Université du SI, 2011.

Cons

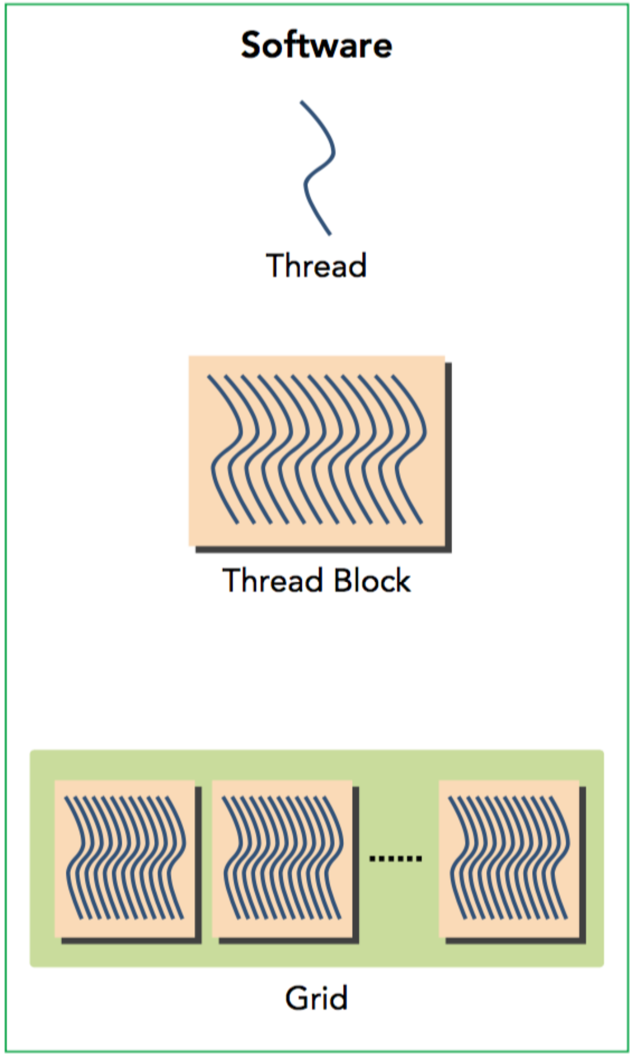
- LATENCY !
  - PCI bus slower than frontside bus (RAM)
    - Solutions : Heterogeneous architectures (Fusion chez AMD, Sandy-Bridge chez Intel…)

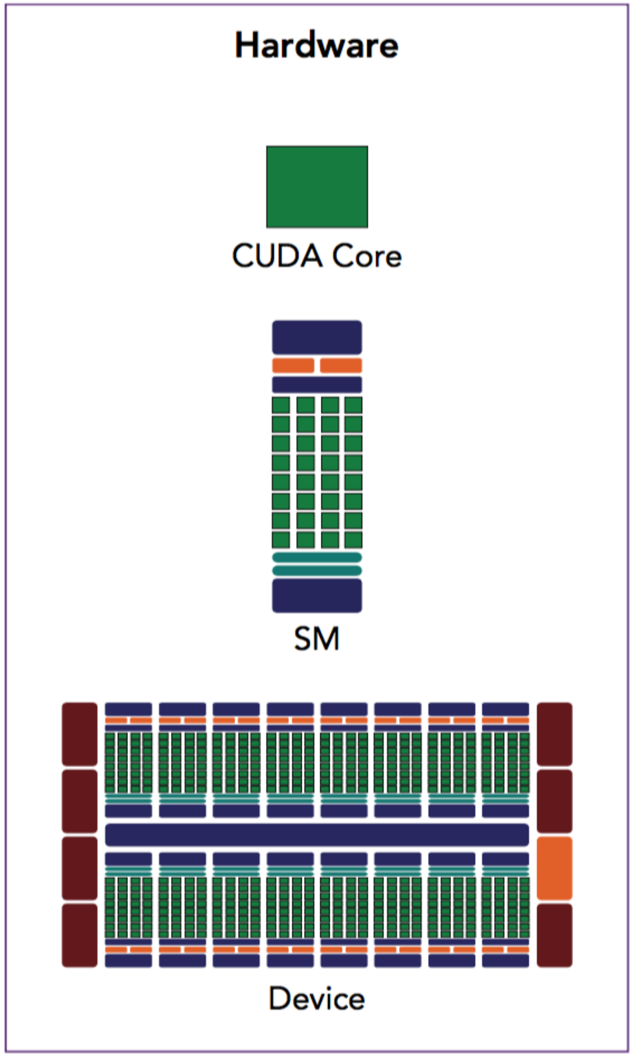
GPU Architecture


DRAM
Host Interface
Giga Thead
DRAM
DRAM
DRAM
DRAM
DRAM
L2 Cache
Streaming multiprocessor
CUDA core
Shared memory, register file, and L1 cache
Scheduler and dispatch units

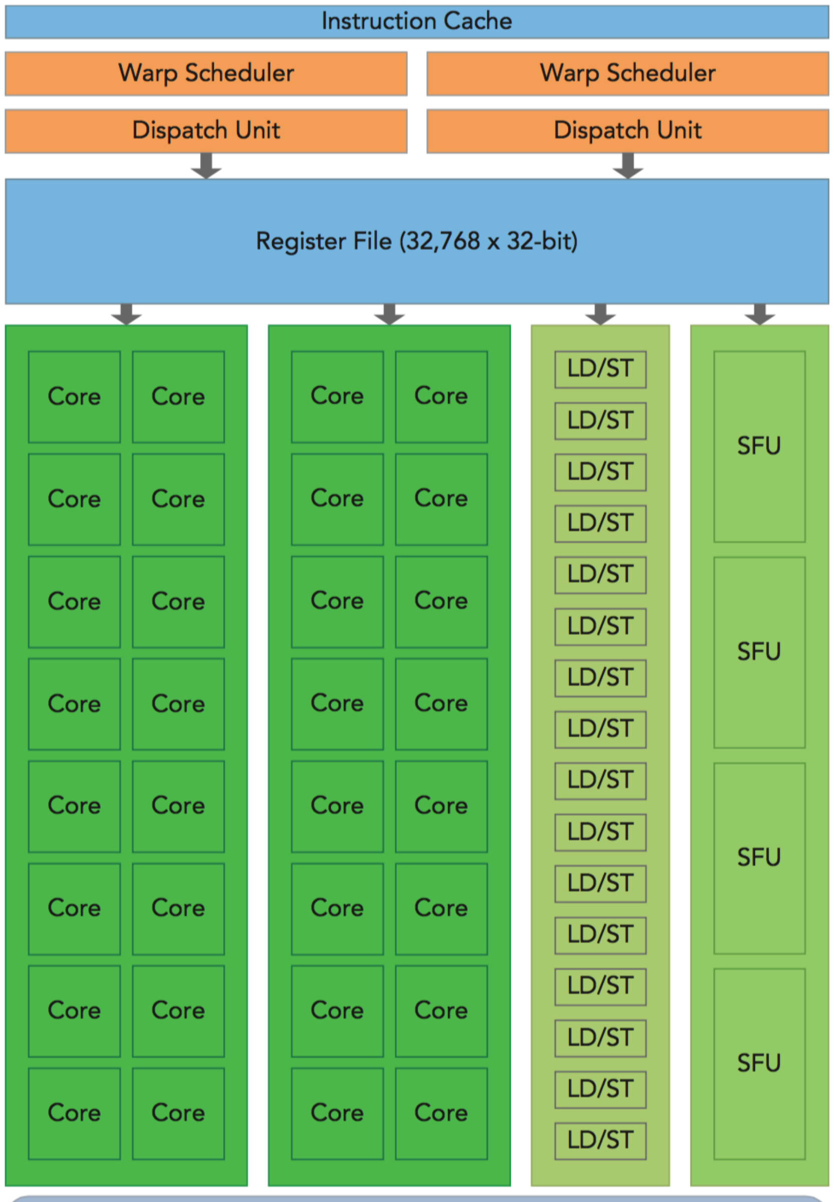
Typical GPU

Streaming multiprocessors





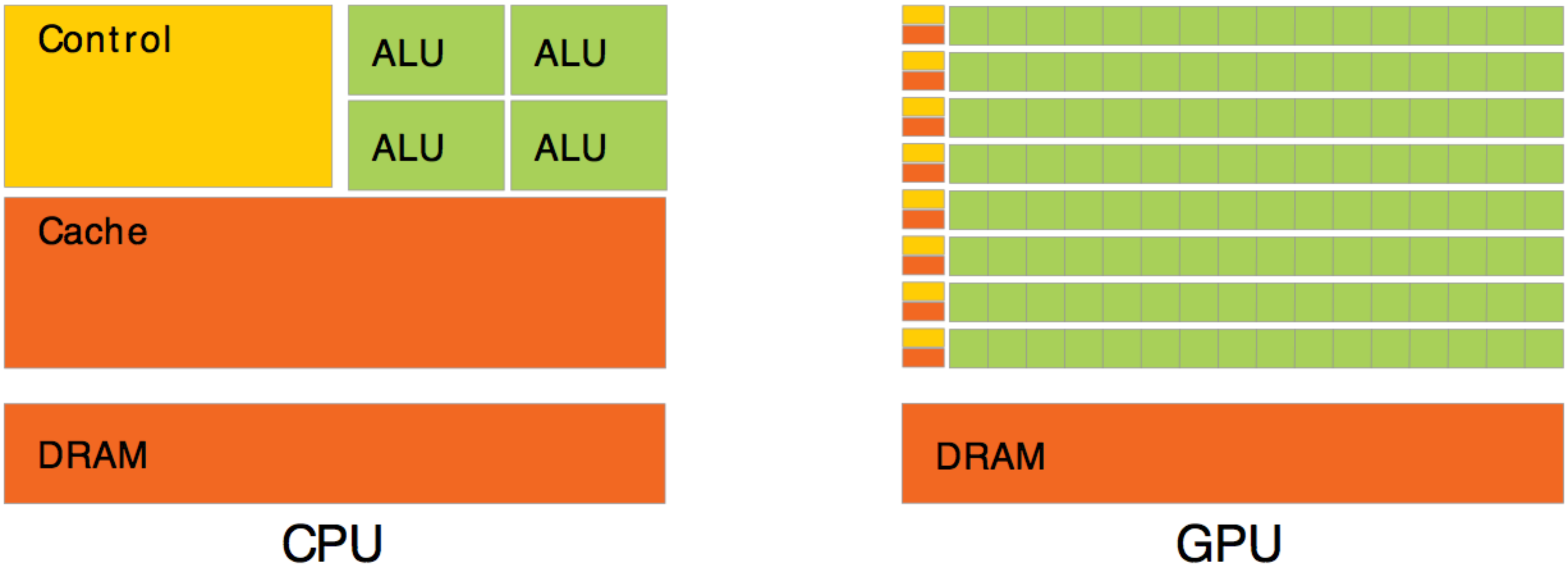
Streaming multiprocessors



Blocks of threads scheduled to one SM

Streaming multiprocessors

- Minimum « packet » of threads : 32. Called a *wrap*.
- One block threads assigned to one SM -> Memory sharing, threads collaborations…
- You *can* synchronize threads within on block
- You *cannot do it* among blocks



More ALUs devoted to computation

Difference between CPU and GPU threads

- « Heavyweight » vs « lightweight »
- Context switching costless

Nvidia Families

- Tegra (mobiles and tablets)
- GeForce (multipurpose)
- Quadro (visualisation)
- Tesla (computation)

Tesla family

- Tesla architecture (attention to confusion !) : 1.0
- Fermi architecture (2.0) : 2010
- Kepler (3.0) : 2012

	FERMI (TESLA C2050)	**KEPLER (TESLA K10)**
CUDA Cores	448	2 x 1536
Memory	6 GB	8 GB
Peak Performance*	1.03 Tflops	4.58 Tflops
Memory Bandwidth	144 GB/s	320 GB/s

CUDA API

CUDA API

1. Low-Low level API : driver API (not covered here)

2. Low level : device API

- Mutually exclusive

- No performance difference

API overview : libraries and packages

https://developer.nvidia.com/gpu-accelerated-libraries

- cublas (BLAS)
- cublas_device (BLAS Kernel Interface)
- cuda_occupancy (Kernel Occupancy Calculation [header file implementation])
- cudadevrt (CUDA Device Runtime)
- cudart (CUDA Runtime)
- cufft (Fast Fourier Transform [FFT])
- cupti (Profiling Tools Interface)
- curand (Random Number Generation)
- cusparse (Sparse Matrix)
- cusolver : combines cuBlas and cuSparse
- npp (NVIDIA Performance Primitives [image and signal processing])
- nvblas ("Drop-in" BLAS)
- nvcuvid (CUDA Video Decoder [Windows, Linux])
- thrust (Parallel Algorithm Library [header file implementation])
- Arrayfire : higher level functions

Why use libraries ?

- Highly optimized
- Portable and maintained
- Thread safety

Verification !

Cuda toolkit v 6.0 installers (5.0 is ok also) :
https://developer.nvidia.com/cuda-toolkit-60

- which nvcc (should be /usr/local/cuda/bin on Linux/OS X)
- nvcc -V

MAC

http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-mac-os-x/index.html#axzz3YJSArOre

- Répertoire d'installation :
  - /Developer/NVIDIA/CUDA-xx
  - Symlinks créés dans /usr/local/cuda

Linux

http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/index.html#axzz3YJSArOre

- Répertoire d’installation :
  - /usr/local/cuda

Examples coming with cuda toolkit

- Go to /usr/local/cuda/samples
- Make for all samples (can take some time)
- Or make inside a specific application

Hardware exploration

deviceQuery

- Go in /usr/local/cuda/samples/1_Utilities/deviceQuery
- Run make and execute
- Sample outputs next slides

```
Device 0: "GeForce GT 650M"
  CUDA Driver Version / Runtime Version          5.5 / 5.5
  CUDA Capability Major/Minor version number:    3.0
  Total amount of global memory:                 1024 MBytes (1073414144 bytes)
  ( 2) Multiprocessors, (192) CUDA Cores/MP:     384 CUDA Cores
  GPU Clock rate:                                900 MHz (0.90 GHz)
  Memory Clock rate:                             2508 Mhz
  Memory Bus Width:                              128-bit
  L2 Cache Size:                                 262144 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Bus ID / PCI location ID:           1 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 5.5, CUDA Runtime Version = 5.5, NumDevs = 1, Device0 = GeForce GT 650M
Result = PASS
```

```
Device 1: "Tesla M2050"
  CUDA Driver Version / Runtime Version          5.0 / 5.0
  CUDA Capability Major/Minor version number:    2.0
  Total amount of global memory:                 3072 MBytes (3220897792 bytes)
  (14) Multiprocessors, ( 32) CUDA Cores/MP:     448 CUDA Cores
  GPU Clock rate:                                1147 MHz (1.15 GHz)
  Memory Clock rate:                             1546 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 786432 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65535), 3D=(2048, 2048, 2048)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 32768
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1536
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (65535, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Bus ID / PCI location ID:           131 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
> Peer access from Tesla M2050 (GPU0) -> Tesla M2050 (GPU1) : No
> Peer access from Tesla M2050 (GPU1) -> Tesla M2050 (GPU0) : No

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 5.0, CUDA Runtime Version = 5.0, NumDevs = 2, Device0 = Tesla M2050, Device1 = Tesla M
```

Bandwidth test

- Run the example in /usr/local/cuda/samples/1_Utilities/bandwidthTest

NVIDIA Management Library (NVML)

https://developer.nvidia.com/nvidia-management-library-nvml
http://developer.download.nvidia.com/compute/cuda/6_0/rel/gdk/nvidia-smi.331.38.pdf
https://www.microway.com/hpc-tech-tips/nvidia-smi_control-your-gpus/

- C-based API useful for :
- Monitoring GPU use
- Modifying some settings
- Available on Linux and Windows

- Command line tool : nvidia-smi
- In c code : #include <nvml.h>