Programming in CUDA

Part I. General presentation

Frédéric S. Masset

Instituto de Ciencias Físicas, UNAM

Lecture on CUDA

Outline

Outline

GPU : Graphics Processing Unit

- Traditionally, a GPU is the sub-system of a computer that takes care of graphics
- Realistic graphics (e.g, in games) can be very computationally expensive
- Computational throughput of GPUs can be (much) larger than that of CPUs

Who makes GPUs ?

Two big companies share almost all the market of GPUs

- NVIDIA
- AMD, which makes ATI Graphics cards.

There also exist low-end GPUs directly embedded within the CPUs (e.g. on the Intel Chips that are present on some Mac Books)

The specific case of NVIDIA's GPUs

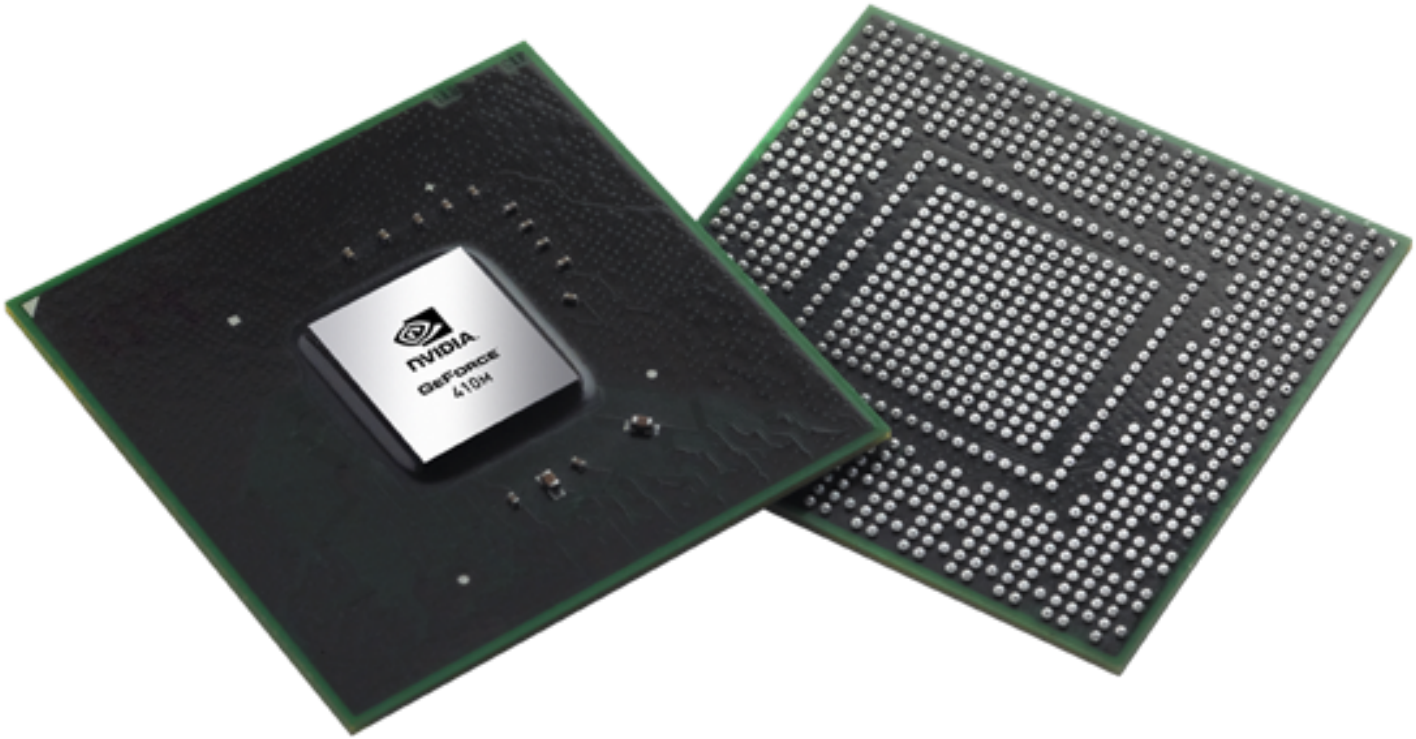
NVIDIA has developed a new framework / language, called CUDA (Compute Unified Device Architecture), that enables one to take advantage of the computational power of GPUS, *not for graphics, but for general purpose programming.*
This lecture is aimed at getting some working knowledge of CUDA.

- We therefore exclusively focus on NVIDIA's graphics chips
- This is a proprietary solution : CUDA does *not* work on ATI Graphics cards
- There exists an open standard for GPUs programming : OpenCL, but its use is more awkward than CUDA (see last lecture).

Example of NVIDIA's graphics chips

Graphics card GeForce GTX580 : 512 cores !

Example of NVIDIA's graphics chips



Graphics card GeForce410M : used on laptop. The GeForce series is the low-end series of NVIDIA (cheapest GPUs).

Example of NVIDIA's graphics chips

Graphics card Quadro 5000 : 352 cores. High end card, for very demanding graphical tasks, but also for GPGPU programming

Example of NVIDIA's graphics chips

Graphics card Tesla C2050 : 448 cores. The most famous GPGPU card of NVIDIA. The Tesla series is *dedicated to HPC*.

Kepler series

More recently, NVIDIA has issued the new Kepler series (compute capabilities 3.0 and 3.5). The Kepler K20 that are on board the cluster “kepler” have 2496 cores each. There are four K20 cards on each node, and 6 nodes in total, hence about 60,000 cores in total.

Connection of the cards

All cards (except laptop ones) have a connector that is plugged into the PCI bus of the computer.
They also have one or various video outputs for monitors.
Interestingly, the TESLA card shown before only has one video output. It is not primarily intended for graphics.

A perhaps not so obvious fact...

If one develops a CUDA program that runs on one of these GPUs, it will basically run on any NVIDIA GPU.

- You can develop a whole code on your laptop (provided it has an NVIDIA card inside) and it will run on a cluster of Tesla or Kepler GPUs.
- Some differences may arise between different GPUs (handling of double precision floating ops, *atomic* operations, etc.) but the broad idea is that CUDA is universal for NVIDIA chips.

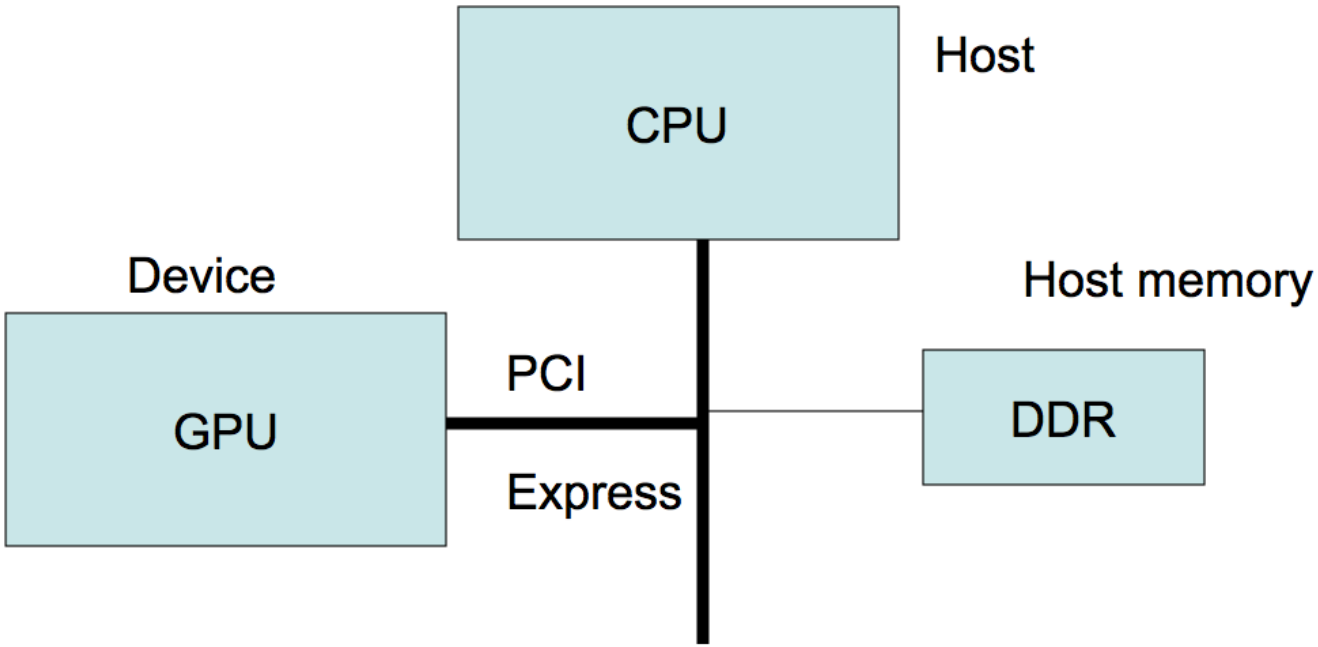
A perhaps not so obvious fact...

If one develops a CUDA program that runs on one of these GPUs, it will basically run on any NVIDIA GPU.

- You can develop a whole code on your laptop (provided it has an NVIDIA card inside) and it will run on a cluster of Tesla or Kepler GPUs.
- Some differences may arise between different GPUs (handling of double precision floating ops, *atomic* operations, etc.) but the broad idea is that CUDA is universal for NVIDIA chips.

Outline

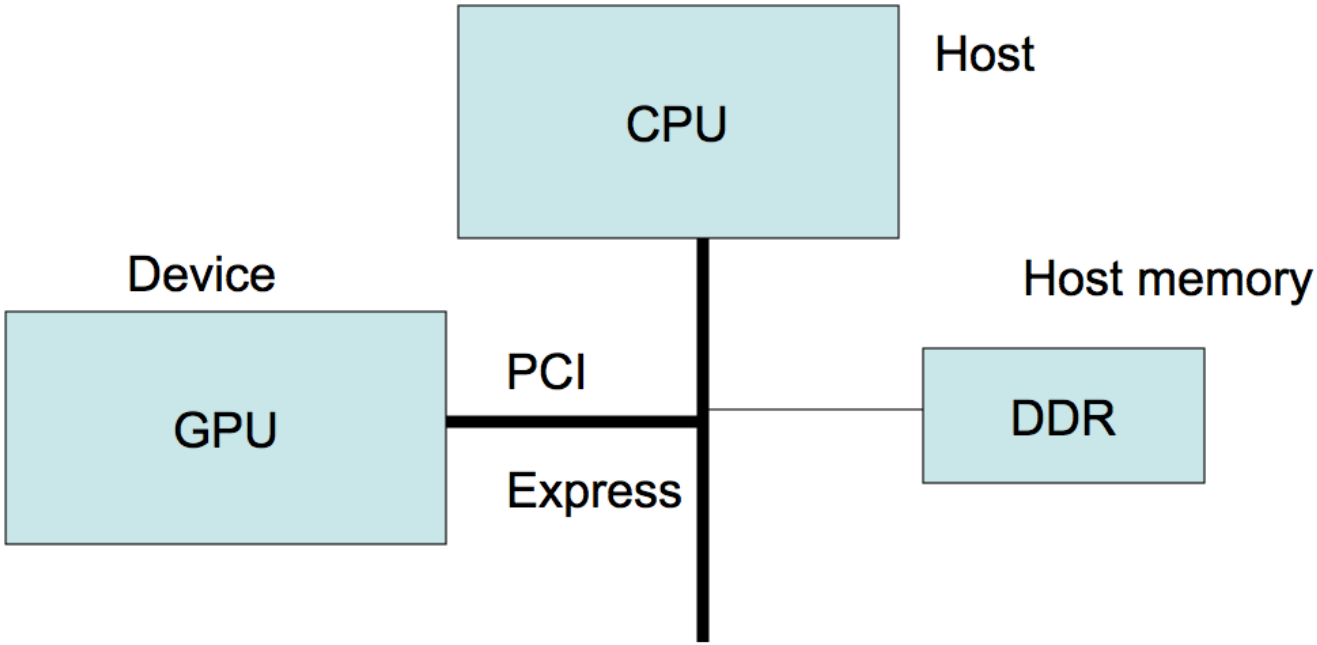
Computer architecture



- CPU : Central Processing Unit
- The GPU has also its own on-board memory (video RAM)

The GPU memory ranges from typically 64 or 128 Mbytes, for low-end chips, to 12 Gb (GeForce GTX TITAN Z).

Computer architecture



Terminology

From now on, everything that is related to the CPU and its memory is called the *host*.
The GPU is referred to as the *device*

Example of a CUDA command

For instance, let us have a look at a very useful and widely used CUDA command : `cudaMemcpy`

```
cudaError_t cudaMemcpy ( void *                  dst,
                         const void *            src,
                         size_t                  count,
                         enum cudaMemcpyKind     kind
                       )
```

Copies `count` bytes from the memory area pointed to by `src` to the memory area pointed to by `dst`, where `kind` is one of **cudaMemcpyHostToHost**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, or **cudaMemcpyDeviceToDevice**, and specifies the direction of the copy. The memory areas may not overlap. Calling **cudaMemcpy()** with `dst` and `src` pointers that do not match the direction of the copy results in an undefined behavior.

The final keyword specifies whether the copy of a memory chunks occurs from the normal RAM to the video RAM (`cudaMemcpyHostToDevice`) or vice-versa (`cudaMemcpyDeviceToHost`), or from a given location to another one within the video RAM (`cudaMemcpyDeviceToDevice`).

Example of a CUDA command

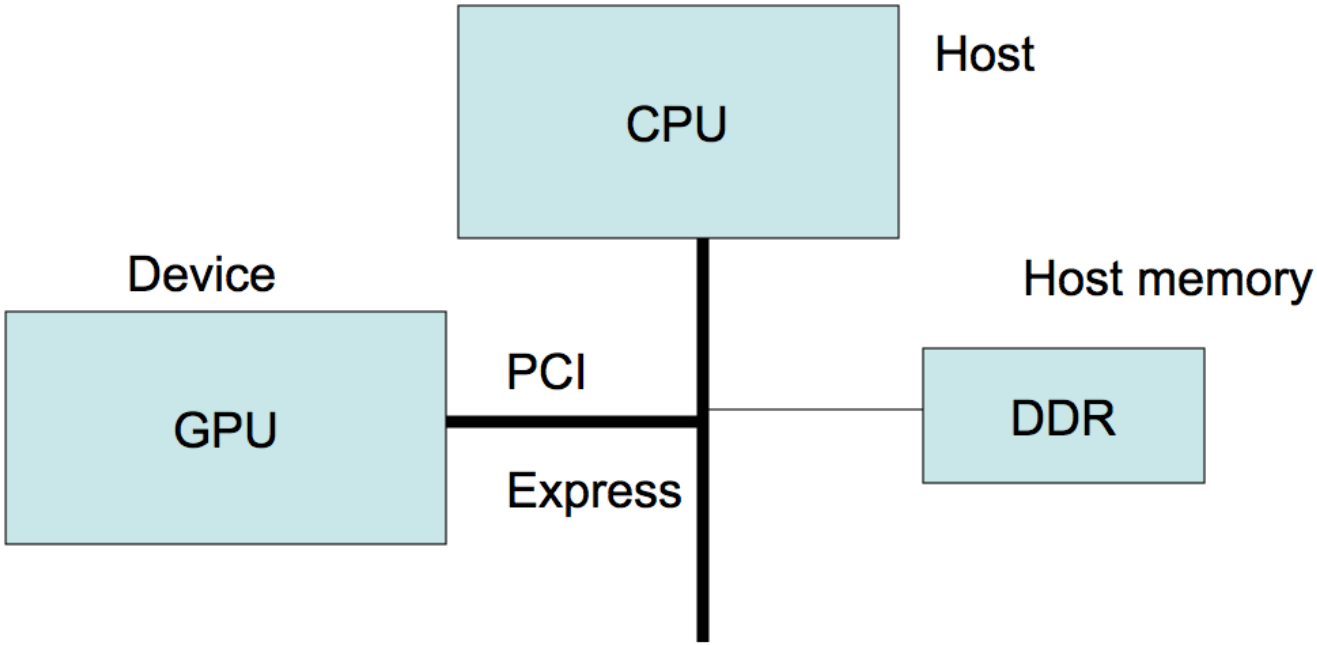
For instance, let us have a look at a very useful and widely used CUDA command : `cudaMemcpy`

```
cudaError_t cudaMemcpy ( void *                  dst,
                         const void *            src,
                         size_t                  count,
                         enum cudaMemcpyKind     kind
                       )
```

Copies `count` bytes from the memory area pointed to by `src` to the memory area pointed to by `dst`, where `kind` is one of **cudaMemcpyHostToHost**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, or **cudaMemcpyDeviceToDevice**, and specifies the direction of the copy. The memory areas may not overlap. Calling **cudaMemcpy()** with `dst` and `src` pointers that do not match the direction of the copy results in an undefined behavior.

The final keyword specifies whether the copy of a memory chunks occurs from the normal RAM to the video RAM (`cudaMemcpyHostToDevice`) or vice-versa (`cudaMemcpyDeviceToHost`), or from a given location to another one within the video RAM (`cudaMemcpyDeviceToDevice`). There is even a host-to-host directive, for completeness...

Computer architecture



Communications between the host and device are made through the PCI bus. There are therefore limited by its bandwidth.

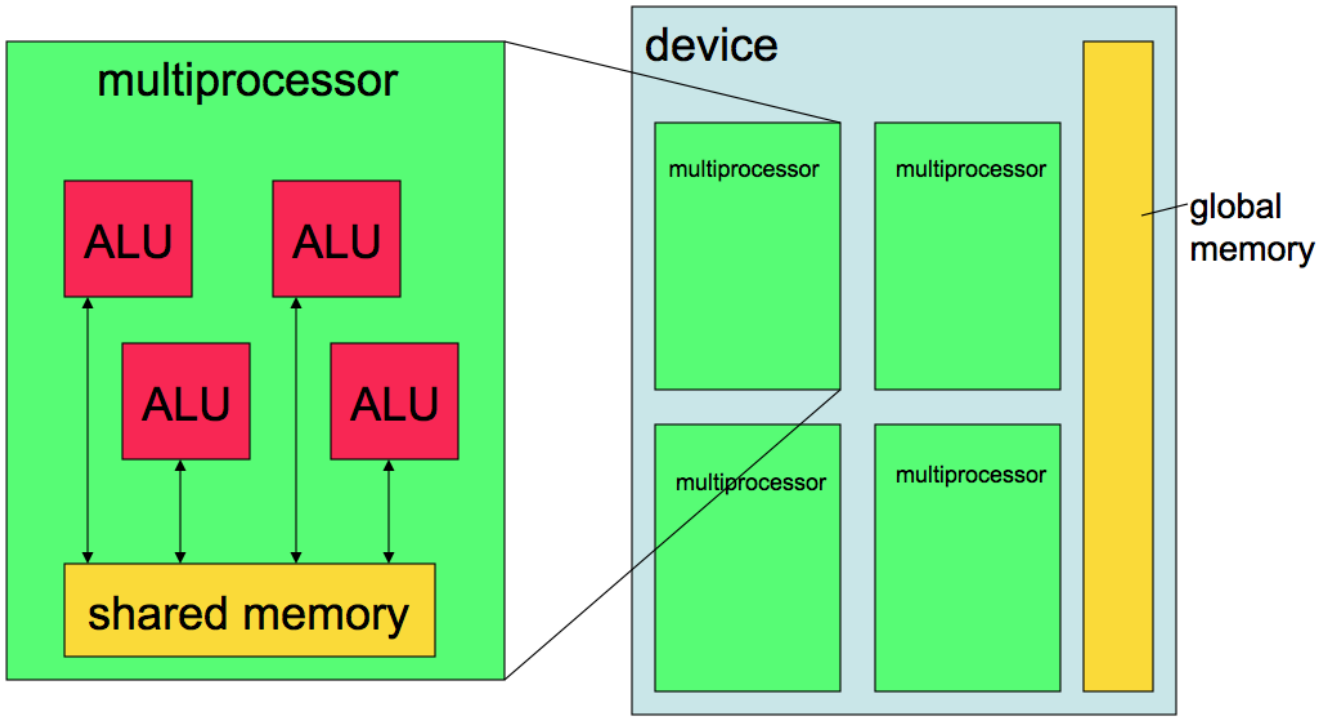
GPUs : many core computing

- A modern CPU has 2 to 8 (or even more) cores.
- Each CPU core is quite sophisticated (*e.g.* several levels of cache memory)
- A GPU has many more cores, but there are much less sophisticated

What is different on a GPU core ?

Not always automatic data caching ! cache (*shared memory* as we will see) can be manual (this is true only of old, 1.x devices)...
"cache" size is small (16k to 48k !!) (can be more than 1 Mb on a modern CPU) !

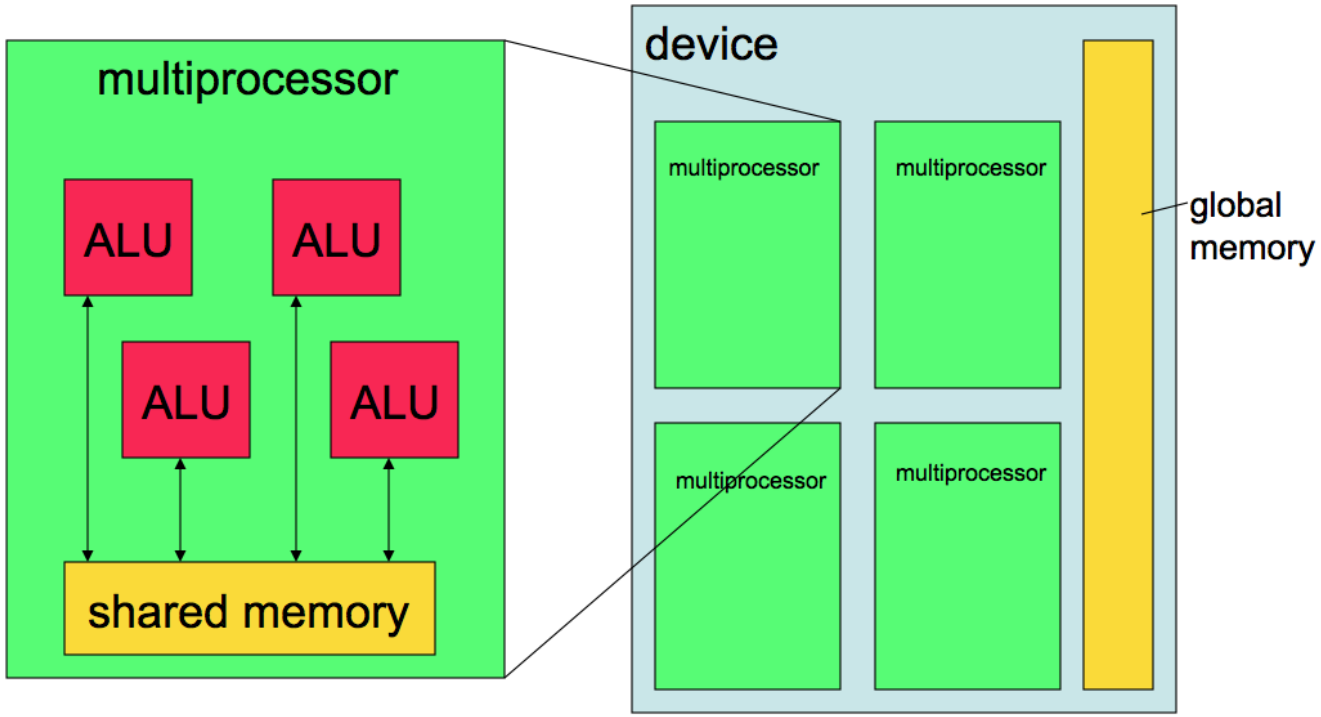
Hardware overview



Terminology

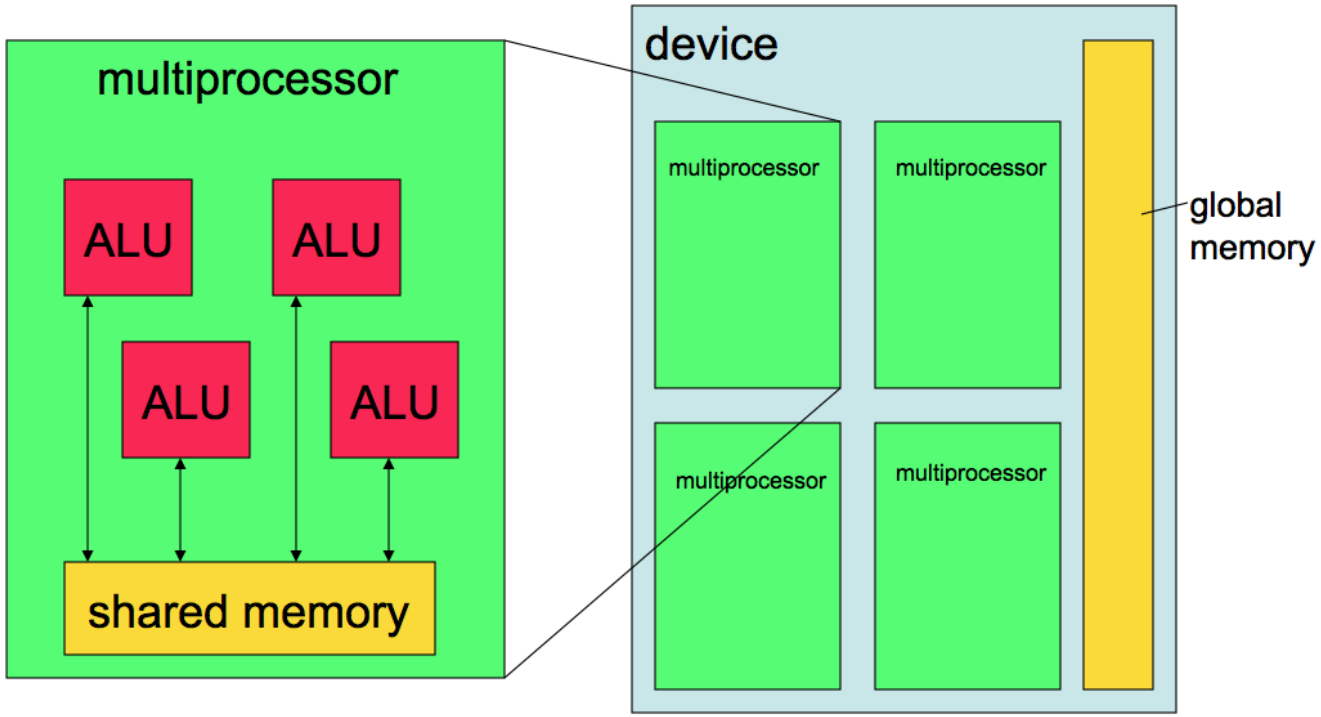
The video RAM is called the *global memory*. This corresponds to the 64 Mb to 12 Gb depending on the card...

Hardware overview



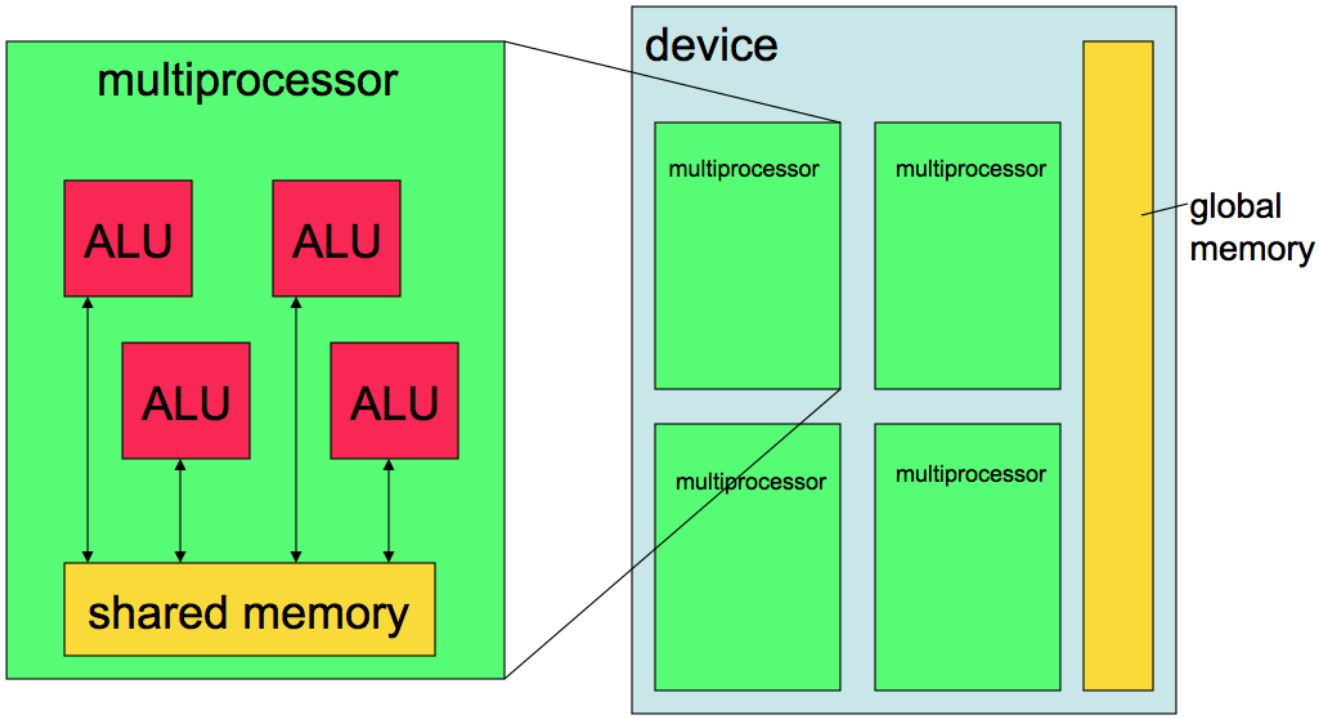
The device is split into several *multiprocessors*.
Each multiprocessor is divided into ALU (*Arithmetic Logical Units*). We can have from 8 to 192 ALUs / multiproc.

Hardware overview



The device is split into several *multiprocessors*.
Each multiprocessor is divided into ALU (*Arithmetic Logical Units*). We can have from 8 to 192 ALUs / multiproc.
One core = One ALU

Hardware overview



Each multiprocessor has its own memory, called the *shared memory* (16k to 48k). All ALUs within a multiprocessor can access this memory. They can also access the global memory at any time, but this is much slower.

Memory performance

- One access to the global memory costs about 400-600 clock cycles
- One access to the shared memory costs 4 clock cycles

Other flavors of device memory

- Constant memory (read only, loaded from the host) : 64k, cached (automatically !)
- Texture memory : 64k, cached !

Memory performance

- One access to the global memory costs about 400-600 clock cycles
- One access to the shared memory costs 4 clock cycles

Last kind : *local* memory

ill-named... Used when shared memory is not sufficient... a small amount of global memory is then reserved for a given multiprocessor, and regarded as *local* to it... But it is *slow* ! (400 - 600 clock cycles).

Memory performance

- One access to the global memory costs about 400-600 clock cycles
- One access to the shared memory costs 4 clock cycles

Last kind : *local* memory

ill-named... Used when shared memory is not sufficient... a small amount of global memory is then reserved for a given multiprocessor, and regarded as *local* to it... But it is *slow* ! (400 - 600 clock cycles). This is called *memory spill*

Summary of different kinds of memories

Kind	Size	Speed
Global memory	Large (up to 6 Gb)	Slow
Shared memory	tiny (16k to 48k)	Fast
Constant memory	small (64k)	Fast
Texture memory	small (64k)	Fast
Local memory	arbitrary	Slow

Summary of different kinds of memories

Kind	Size	Speed
Global memory	Large (up to 6 Gb)	Slow
Shared memory	tiny (16k to 48k)	Fast
Constant memory	small (64k)	Fast
Texture memory	small (64k)	Fast
Local memory	arbitrary	Slow

Summary of different kinds of memories

Kind	Size	Speed
Global memory	Large (up to 6 Gb)	Slow
Shared memory	tiny (16k to 48k)	Fast
Constant memory	small (64k)	Fast
Texture memory	small (64k)	Fast
Local memory	arbitrary	Slow

Summary of different kinds of memories

Kind	Size	Speed
Global memory	Large (up to 6 Gb)	Slow
Shared memory	tiny (16k to 48k)	Fast
Constant memory	small (64k)	Fast
Texture memory	small (64k)	Fast
Local memory	arbitrary	Slow

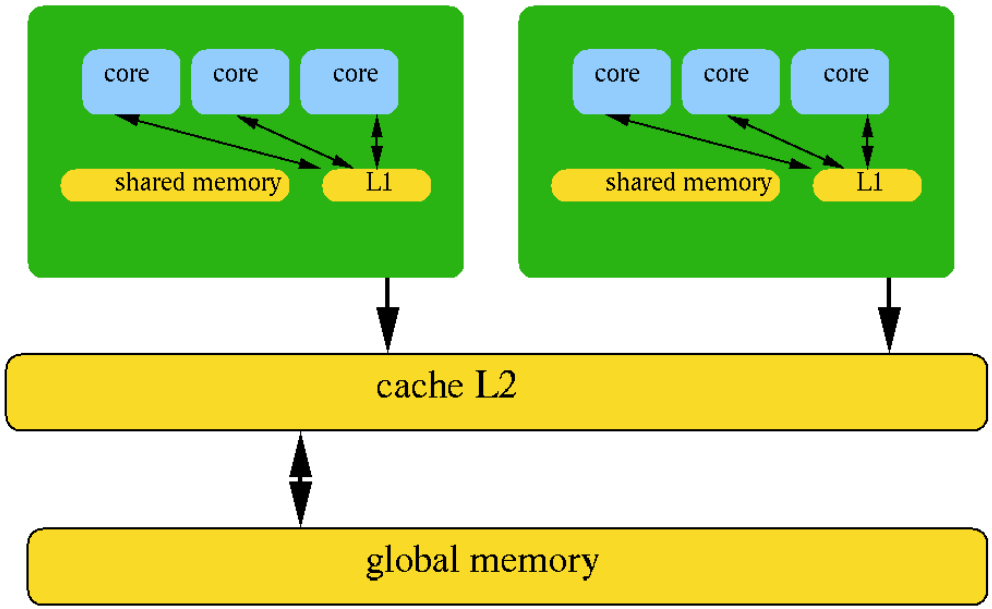
Summary of different kinds of memories

Kind	Size	Speed
Global memory	Large (up to 6 Gb)	Slow
Shared memory	tiny (16k to 48k)	Fast
Constant memory	small (64k)	Fast
Texture memory	small (64k)	Fast
Local memory	arbitrary	Slow

Different memory model for Fermi's

Newer GPUs (Fermi's and beyond) have *real* caches : L1 and L2

- L1 is local to each multiprocessor. Local memory is cached in L1.
- L2 is a cache for interaction with global memory. Even communications with the host go through the L2 cache.



shared memory vs. L1 cache

By default, on a Fermi architecture, on has:

- 48 Kb of shared memory per multiprocessor
- 16 Kb of L1 cache

It is possible to change this distribution as:

- 16 Kb of shared memory per MP (like in 1.x architectures)
- 48 Kb of L1 cache

How to take care of the L1 / L2 caches ?

It depends on your purpose

- Act as if there was no cache : your code will also be optimized on 1.x architectures
- Also, GPU caches are much smaller then CPU caches, and they are hit by many more threads than CPU caches : difficult to block a cache line.
- GPU caches are intended to smooth out some pattern accesses, or to help in case of memory spill.
- Use the shared memory as a manual cache. You keep the control.
- Nonetheless simple applications may turn out to be as fast as, or even faster, when one relies on automatic caching rather than dealing explicitly with the shared memory.

How do we use all these cores ?

Answer:

We spawn many threads (replica of a given “program” or “execution flow”). A given thread is executed on a core.

Some more terminology

On the device, we do not speak of functions or processes (like we would on the CPU).

- What would be a process on the CPU is called a context on the GPU.
- What would be a function in a C program on the CPU is rather called a kernel on the GPU.

The task of a CUDA programmer is to write kernels, which will execute on the GPUs. This should not be confused with the kernel of the operating system running on the host (*e.g.* a Linux kernel) which is obviously quite a different thing.

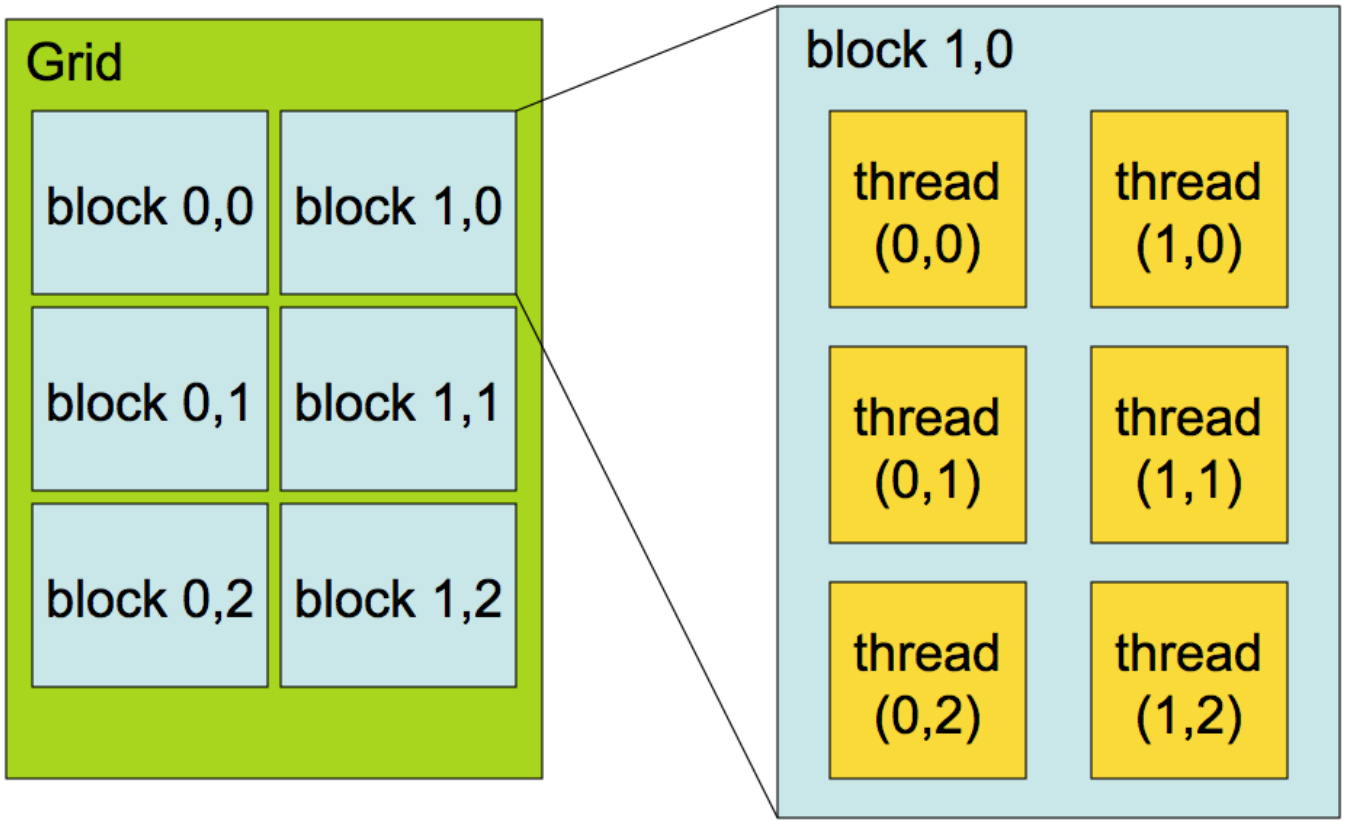
Many core computing

A kernel spawns a large number of copies of itself, the threads, on the GPU cores.
This number must be large to ensure good performance.
Context switches yield some overhead $\Rightarrow$ mask it by feeding many threads.

Analogy

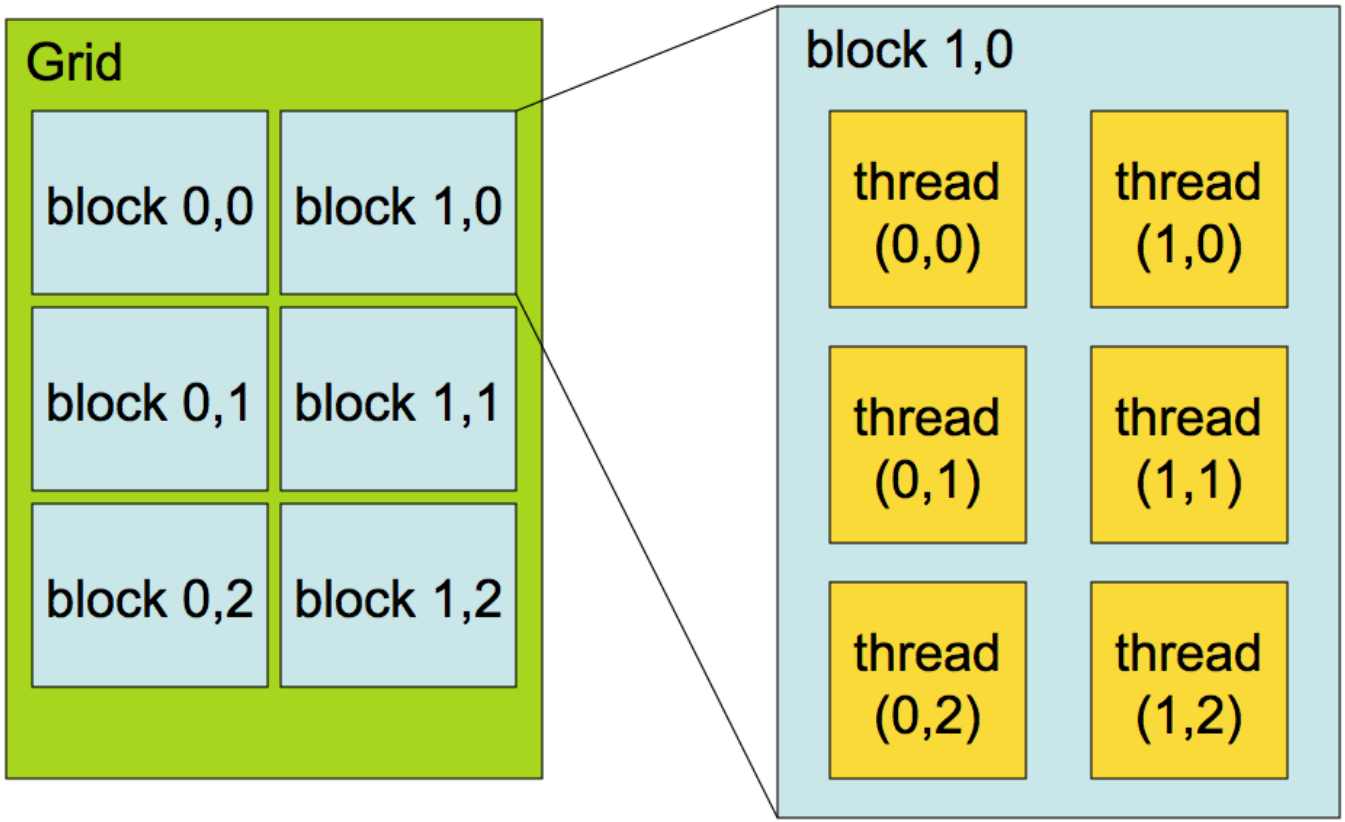
Moulding craft

Logical view of threads spawning



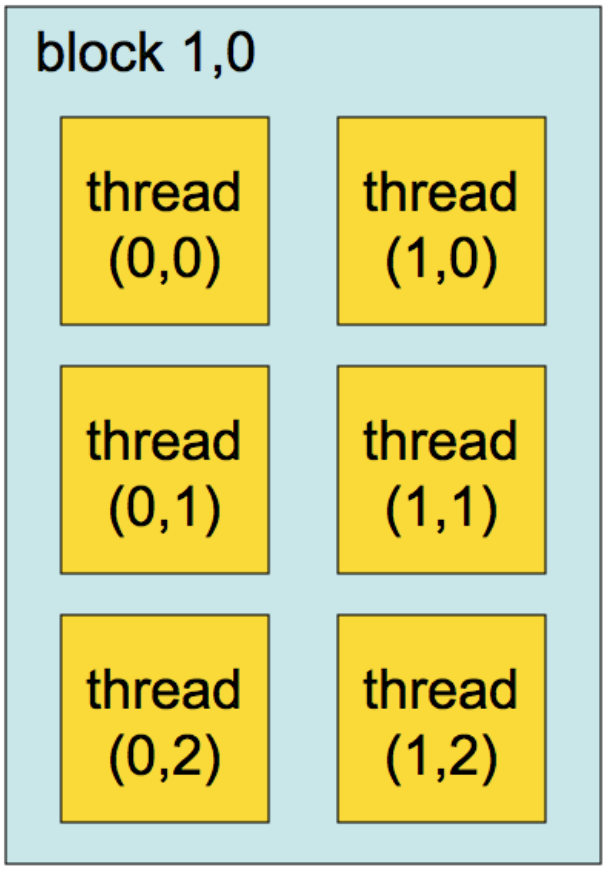
The Grid array of block is 1D or 2D. Its size has nothing to do with the number of multiproc's on the GPU. It is the programmer's choice.

Logical view of threads spawning



A given block is treated by one multiprocessor. A multiprocessor can run concurrently several blocks.

Block splitting into threads



The array of threads in a block is 1D, 2D or 3D. One can (and must) have many more threads in a block than cores in a multiproc.

Threads IN A SAME BLOCK ONLY can cooperate : share data through shared memory, synchronize execution.

Warps of threads

A warp is a subset of 32 threads within a given block (hence on a given multiprocessor). They can be thought of as executing simultaneously and therefore have to do the very same thing (beware of divergent tests...)
Technically, the smallest sub-unit is the half-warp. We will reexamine this concept when dealing with coalesced memory accesses.

Unveiling what's next

Assume that we deal with, say, a matrix addition:

$$C_{ij} = A_{ij} + B_{ij}$$

We will manage so that one given thread treat one given "pixel" (i, j), perform its own scalar sum, and return. The work performed collectively by all threads will amount to the full matrix addition.

But before... time to hands on !

We need to split into work groups.
Please identify your group. Be sure to know your login and password.
We must set up a full CUDA working environment.
If you have not done so already, go to

`http://developer.nvidia.com/cuda-toolkit-40`

and select your OS, then install the three different packages:

- The driver
- The toolkit (compiler, libraries, doc, etc.)
- The SDK (software development kit)

What GPU do I have in my machine ?

Once this is done, you need to identify where are the source files of the program called `deviceQuery`. They should be in `$PATH_SDK/C/src/deviceQuery`.
There, issue a `make` command. The executable location will depend on your OS (look for `$PATH_SDK/C/bin/` etc.). Once you have located it, execute it and examine the output.

Typical output of `deviceQuery`

```
  masset:release> ./deviceQuery
./deviceQuery Starting...
CUDA Device Query (Runtime API) version (CUDART static linking)

There is 1 device supporting CUDA

Device 0: "GeForce 8600M GT"
  CUDA Driver Version:                           3.20
  CUDA Runtime Version:                          3.20
  CUDA Capability Major/Minor version number:    1.1
  Total amount of global memory:                 268238848 bytes
  Multiprocessors x Cores/MP = Cores:            4 (MP) x 8 (Cores/MP)
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
```

Typical output of `deviceQuery`

```
  masset:release> ./deviceQuery
./deviceQuery Starting...
CUDA Device Query (Runtime API) version (CUDART static linking)

There is 1 device supporting CUDA

Device 0: "GeForce 8600M GT"
  CUDA Driver Version:                           3.20
  CUDA Runtime Version:                          3.20
  CUDA Capability Major/Minor version number:    1.1
  Total amount of global memory:                 268238848 bytes
  Multiprocessors x Cores/MP = Cores:            4 (MP) x 8 (Cores/MP)
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
```

Software version. May be updated by NVIDIA

Typical output of `deviceQuery`

```
  masset:release> ./deviceQuery
./deviceQuery Starting...
CUDA Device Query (Runtime API) version (CUDART static linking)

There is 1 device supporting CUDA

Device 0: "GeForce 8600M GT"
  CUDA Driver Version:                           3.20
  CUDA Runtime Version:                          3.20
  CUDA Capability Major/Minor version number:    1.1
  Total amount of global memory:                 268238848 bytes
  Multiprocessors x Cores/MP = Cores:            4 (MP) x 8 (Cores/MP)
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
```

Hardware version. Cannot change. 2.0 and above $\Rightarrow$ Fermi's

Typical output of `deviceQuery`

```
  masset:release> ./deviceQuery
./deviceQuery Starting...
CUDA Device Query (Runtime API) version (CUDART static linking)

There is 1 device supporting CUDA

Device 0: "GeForce 8600M GT"
  CUDA Driver Version:                           3.20
  CUDA Runtime Version:                          3.20
  CUDA Capability Major/Minor version number:    1.1
  Total amount of global memory:                 268238848 bytes
  Multiprocessors x Cores/MP = Cores:            4 (MP) x 8 (Cores/MP)
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
```

256 Mb of RAM

Typical output of `deviceQuery`

```
  masset:release> ./deviceQuery
./deviceQuery Starting...
CUDA Device Query (Runtime API) version (CUDART static linking)

There is 1 device supporting CUDA

Device 0: "GeForce 8600M GT"
  CUDA Driver Version:                           3.20
  CUDA Runtime Version:                          3.20
  CUDA Capability Major/Minor version number:    1.1
  Total amount of global memory:                 268238848 bytes
  Multiprocessors x Cores/MP = Cores:            4 (MP) x 8 (Cores/MP)
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
```

Here 8 cores/MP. This figure can be larger for Fermi's

Typical output of `deviceQuery`

```
  masset:release> ./deviceQuery
./deviceQuery Starting...
CUDA Device Query (Runtime API) version (CUDART static linking)

There is 1 device supporting CUDA

Device 0: "GeForce 8600M GT"
  CUDA Driver Version:                           3.20
  CUDA Runtime Version:                          3.20
  CUDA Capability Major/Minor version number:    1.1
  Total amount of global memory:                 268238848 bytes
  Multiprocessors x Cores/MP = Cores:            4 (MP) x 8 (Cores/MP)
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
```

Typical output of `deviceQuery`

```
  masset:release> ./deviceQuery
./deviceQuery Starting...
CUDA Device Query (Runtime API) version (CUDART static linking)

There is 1 device supporting CUDA

Device 0: "GeForce 8600M GT"
  CUDA Driver Version:                           3.20
  CUDA Runtime Version:                          3.20
  CUDA Capability Major/Minor version number:    1.1
  Total amount of global memory:                 268238848 bytes
  Multiprocessors x Cores/MP = Cores:            4 (MP) x 8 (Cores/MP)
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
```

Typical output of `deviceQuery`

```
  masset:release> ./deviceQuery
./deviceQuery Starting...
CUDA Device Query (Runtime API) version (CUDART static linking)

There is 1 device supporting CUDA

Device 0: "GeForce 8600M GT"
  CUDA Driver Version:                           3.20
  CUDA Runtime Version:                          3.20
  CUDA Capability Major/Minor version number:    1.1
  Total amount of global memory:                 268238848 bytes
  Multiprocessors x Cores/MP = Cores:            4 (MP) x 8 (Cores/MP)
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
```

Typical output of `deviceQuery`

```
  masset:release> ./deviceQuery
./deviceQuery Starting...
CUDA Device Query (Runtime API) version (CUDART static linking)

There is 1 device supporting CUDA

Device 0: "GeForce 8600M GT"
  CUDA Driver Version:                           3.20
  CUDA Runtime Version:                          3.20
  CUDA Capability Major/Minor version number:    1.1
  Total amount of global memory:                 268238848 bytes
  Multiprocessors x Cores/MP = Cores:            4 (MP) x 8 (Cores/MP)
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
```

Typical output of `deviceQuery`

```
  masset:release> ./deviceQuery
./deviceQuery Starting...
CUDA Device Query (Runtime API) version (CUDART static linking)

There is 1 device supporting CUDA

Device 0: "GeForce 8600M GT"
  CUDA Driver Version:                           3.20
  CUDA Runtime Version:                          3.20
  CUDA Capability Major/Minor version number:    1.1
  Total amount of global memory:                 268238848 bytes
  Multiprocessors x Cores/MP = Cores:            4 (MP) x 8 (Cores/MP)
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
```

Typical output of `deviceQuery`

```
Maximum sizes of each dimension of a block:    512 x 512 x 64
Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
Maximum memory pitch:                          2147483647 bytes
Texture alignment:                             256 bytes
Clock rate:                                    1.04 GHz
Concurrent copy and execution:                 Yes
Run time limit on kernels:                     Yes
Integrated:                                    No
Support host page-locked memory mapping:       Yes
Compute mode:                                  Default
   (multiple host threads can use this device simultaneously)
Concurrent kernel execution:                   No
Device has ECC support enabled:                No
Device is using TCC driver mode:               No
```

Decomposition of a block into threads can therefore be 3D

Typical output of `deviceQuery`

```
Maximum sizes of each dimension of a block:    512 x 512 x 64
Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
Maximum memory pitch:                          2147483647 bytes
Texture alignment:                             256 bytes
Clock rate:                                    1.04 GHz
Concurrent copy and execution:                 Yes
Run time limit on kernels:                     Yes
Integrated:                                    No
Support host page-locked memory mapping:       Yes
Compute mode:                                  Default
   (multiple host threads can use this device simultaneously)
Concurrent kernel execution:                   No
Device has ECC support enabled:                No
Device is using TCC driver mode:               No
```

Grid of blocks are at most 2D

Typical output of `deviceQuery`

```
Maximum sizes of each dimension of a block:    512 x 512 x 64
Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
Maximum memory pitch:                          2147483647 bytes
Texture alignment:                             256 bytes
Clock rate:                                    1.04 GHz
Concurrent copy and execution:                 Yes
Run time limit on kernels:                     Yes
Integrated:                                    No
Support host page-locked memory mapping:       Yes
Compute mode:                                  Default
   (multiple host threads can use this device simultaneously)
Concurrent kernel execution:                   No
Device has ECC support enabled:                No
Device is using TCC driver mode:               No
```

Typical output of `deviceQuery`

```
Maximum sizes of each dimension of a block:    512 x 512 x 64
Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
Maximum memory pitch:                          2147483647 bytes
Texture alignment:                             256 bytes
Clock rate:                                    1.04 GHz
Concurrent copy and execution:                 Yes
Run time limit on kernels:                     Yes
Integrated:                                    No
Support host page-locked memory mapping:       Yes
Compute mode:                                  Default
   (multiple host threads can use this device simultaneously)
Concurrent kernel execution:                   No
Device has ECC support enabled:                No
Device is using TCC driver mode:               No
```

Typical output of `deviceQuery`

```
Maximum sizes of each dimension of a block:    512 x 512 x 64
Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
Maximum memory pitch:                          2147483647 bytes
Texture alignment:                             256 bytes
Clock rate:                                    1.04 GHz
Concurrent copy and execution:                 Yes
Run time limit on kernels:                     Yes
Integrated:                                    No
Support host page-locked memory mapping:       Yes
Compute mode:                                  Default
   (multiple host threads can use this device simultaneously)
Concurrent kernel execution:                   No
Device has ECC support enabled:                No
Device is using TCC driver mode:               No
```

Expected performance gain wrt 1 CPU core :
$1.04 \times 32/2.4 \sim 13.8$

Typical output of `deviceQuery`

```
Maximum sizes of each dimension of a block:    512 x 512 x 64
Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
Maximum memory pitch:                          2147483647 bytes
Texture alignment:                             256 bytes
Clock rate:                                    1.04 GHz
Concurrent copy and execution:                 Yes
Run time limit on kernels:                     Yes
Integrated:                                    No
Support host page-locked memory mapping:       Yes
Compute mode:                                  Default
   (multiple host threads can use this device simultaneously)
Concurrent kernel execution:                   No
Device has ECC support enabled:                No
Device is using TCC driver mode:               No
```

Typical output of `deviceQuery`

```
Maximum sizes of each dimension of a block:    512 x 512 x 64
Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
Maximum memory pitch:                          2147483647 bytes
Texture alignment:                             256 bytes
Clock rate:                                    1.04 GHz
Concurrent copy and execution:                 Yes
Run time limit on kernels:                     Yes
Integrated:                                    No
Support host page-locked memory mapping:       Yes
Compute mode:                                  Default
   (multiple host threads can use this device simultaneously)
Concurrent kernel execution:                   No
Device has ECC support enabled:                No
Device is using TCC driver mode:               No
```

Typical output of `deviceQuery`

```
Maximum sizes of each dimension of a block:    512 x 512 x 64
Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
Maximum memory pitch:                          2147483647 bytes
Texture alignment:                             256 bytes
Clock rate:                                    1.04 GHz
Concurrent copy and execution:                 Yes
Run time limit on kernels:                     Yes
Integrated:                                    No
Support host page-locked memory mapping:       Yes
Compute mode:                                  Default
   (multiple host threads can use this device simultaneously)
Concurrent kernel execution:                   No
Device has ECC support enabled:                No
Device is using TCC driver mode:               No
```

Typical output of `deviceQuery`

```
Maximum sizes of each dimension of a block:    512 x 512 x 64
Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
Maximum memory pitch:                          2147483647 bytes
Texture alignment:                             256 bytes
Clock rate:                                    1.04 GHz
Concurrent copy and execution:                 Yes
Run time limit on kernels:                     Yes
Integrated:                                    No
Support host page-locked memory mapping:       Yes
Compute mode:                                  Default
   (multiple host threads can use this device simultaneously)
Concurrent kernel execution:                   No
Device has ECC support enabled:                No
Device is using TCC driver mode:               No
```

Typical output of `deviceQuery`

```
Maximum sizes of each dimension of a block:    512 x 512 x 64
Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
Maximum memory pitch:                          2147483647 bytes
Texture alignment:                             256 bytes
Clock rate:                                    1.04 GHz
Concurrent copy and execution:                 Yes
Run time limit on kernels:                     Yes
Integrated:                                    No
Support host page-locked memory mapping:       Yes
Compute mode:                                  Default
   (multiple host threads can use this device simultaneously)
Concurrent kernel execution:                   No
Device has ECC support enabled:                No
Device is using TCC driver mode:               No
```

Typical output of `deviceQuery`

```
Maximum sizes of each dimension of a block:    512 x 512 x 64
Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
Maximum memory pitch:                          2147483647 bytes
Texture alignment:                             256 bytes
Clock rate:                                    1.04 GHz
Concurrent copy and execution:                 Yes
Run time limit on kernels:                     Yes
Integrated:                                    No
Support host page-locked memory mapping:       Yes
Compute mode:                                  Default
   (multiple host threads can use this device simultaneously)
Concurrent kernel execution:                   No
Device has ECC support enabled:                No
Device is using TCC driver mode:               No
```

Typical output of `deviceQuery`

```
Maximum sizes of each dimension of a block:    512 x 512 x 64
Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
Maximum memory pitch:                          2147483647 bytes
Texture alignment:                             256 bytes
Clock rate:                                    1.04 GHz
Concurrent copy and execution:                 Yes
Run time limit on kernels:                     Yes
Integrated:                                    No
Support host page-locked memory mapping:       Yes
Compute mode:                                  Default
   (multiple host threads can use this device simultaneously)
Concurrent kernel execution:                   No
Device has ECC support enabled:                No
Device is using TCC driver mode:               No
```

Error control available only on Teslas, Quadro 5000 and 6000.

Typical output of `deviceQuery`

```
Maximum sizes of each dimension of a block:    512 x 512 x 64
Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
Maximum memory pitch:                          2147483647 bytes
Texture alignment:                             256 bytes
Clock rate:                                    1.04 GHz
Concurrent copy and execution:                 Yes
Run time limit on kernels:                     Yes
Integrated:                                    No
Support host page-locked memory mapping:       Yes
Compute mode:                                  Default
   (multiple host threads can use this device simultaneously)
Concurrent kernel execution:                   No
Device has ECC support enabled:                No
Device is using TCC driver mode:               No
```

TCC : Tesla Compute Cluster / Available on Tesla only

Dynamic allocation on device

On the contrary of the host, you will NOT get any error message if you perform an out-of-bound write on the device.

You can write even in a memory zone reserved by another process, or in the card OS.

The memory is not reset upon completion. Your data remain visible to others.