New features for CUDA GPUs

Tutorial at 18th IEEE CSE'15 and 13th IEEE EUC'15 conferences

October, 20th, 2015

Manuel Ujaldón

A/Prof. @ University of Málaga (Spain)

Conjoint Senior Lecturer @ Univ. de Newcastle (Australia)

CUDA Fellow @ Nvidia

NVIDIA

Talk outline [30 slides]

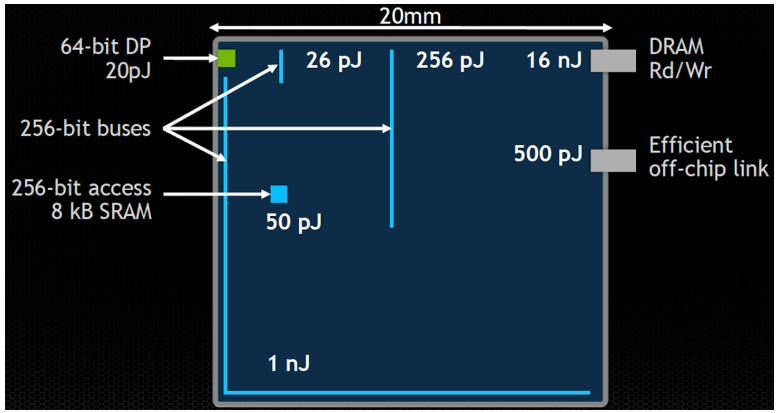
1. Optimizing power on GPUs [8 slides]
2. Dynamic parallelism [6]
3. Hyper-Q [6]
4. Unified memory [8]
5. NV Link [1]
6. Summary [1]

I. Optimizing power on GPUs

NVIDIA

The cost of data movement

- Communication takes more energy than arithmetic.



Values for 32 nm. manufacturing process.

Energy shopping list: Past, present, future

Processor technology	40 nm. (2005)	10 nm. (2020)	Overall reduction factor
Voltage (nominal)	0.9 v.	0.7 v.	
DFMA (double fused multiply-add) energy	50 pJ.	7.6 pJ.	6.57 x
64 bits 8 KB. SRAM read (cache memory)	14 pJ.	2.1 pJ.	6.66 x
Wire energy (256 bits wide, 10 mm. long)	310 pJ.	174.0 pJ.	1.78 x

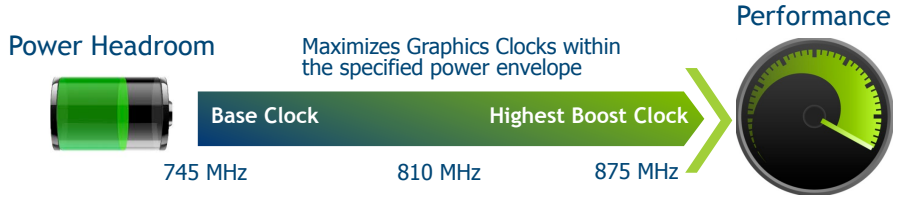
Memory technology	45 nm.	16 nm.	Overall reduction factor
DRAM interface pin bandwidth	4 Gbps.	50 Gbps.	
DRAM interface energy (read/write bandwidth)	20-30 pJ/bit	2 pJ/bit	10-15 x
DRAM access energy (latency)	8-15 pJ/bit	2.5 pJ/bit	3-6 x

A regular floating-point operation requires a minimum of 4 pJ.

Source: Vogelsang [Micro 2010], Keckler [Micro2011]

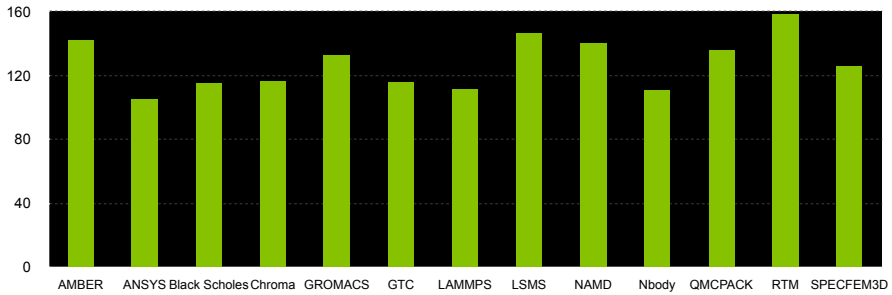
GPU Boost

- Allows to speed-up the GPU clock up to 17% if the power required by an application is low.
- The base clock will be restored if we exceed 235 W.
- We can set up a persistent mode which keep values permanently, or another one for a single run.



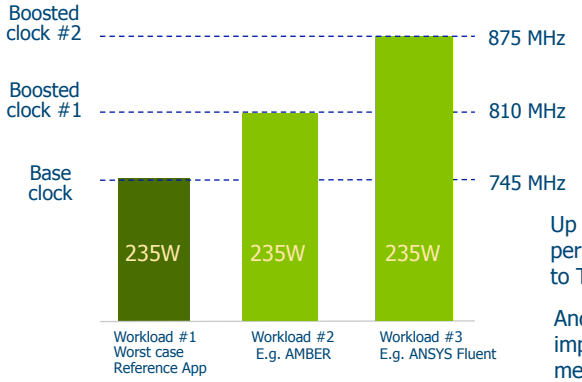
Every application has a different behaviour regarding power consumption

- Here we see the average power (watts) on a Tesla K20X for a set of popular applications within the HPC field:



Those applications which are less power hungry can benefit from a higher clock rate

- For the Tesla K40 case, 3 clocks are defined, 8.7% apart.

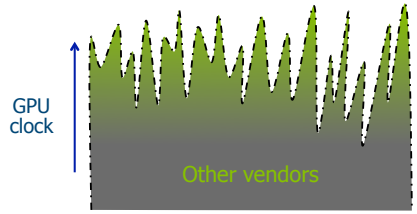


Up to 40% higher performance relative to Tesla K20X.

And not only GFLOPS are improved, but also effective memory bandwidth.

GPU Boost compared to other approaches

- It is better a stationary state for the frequency to avoid thermal stress and improve reliability.



Automatic clock switching

Deterministic Clocks

	Other vendors	Tesla K40
Default	Boost	Base
Preset options	Lock to base clock	3 levels: Base, Boost1 o Boost2
Boost interface	Control panel	Shell command: `nv-smi`
Target duration for boosts	Roughly 50% of run-time	100% of workload run time

GPU Boost - List of commands

Command	Effect
`nvidia-smi -q -d SUPPORTED_CLOCKS`	**View the clocks supported by our GPU**
`nvidia-smi -ac <MEM clock, Graphics clock>`	**Set one of the supported clocks**
`nvidia-smi -pm 1`	Enables persistent mode: The clock settings are preserved after restarting the system or driver
`nvidia-smi -pm 0`	Enables non-persistent mode: Clock settings revert to base clocks after restarting the system or driver
`nvidia-smi -q -d CLOCK`	**Query the clock in use**
`nvidia-smi -rac`	Reset clocks back to the base clock
`nvidia-smi -acp 0`	Allow non-root users to change clock rates

Example: Query the clock in use

- `nvidia-smi -q -d CLOCK –id=0000:86:00.0`

```
==============NVSMI LOG==============

Timestamp                           : Wed Jan 29 13:35:58 2014
Driver Version                      : 319.37

Attached GPUs                       : 5
GPU 0000:86:00.0
    Clocks
        Graphics                    : 875 MHz
        SM                          : 875 MHz
        Memory                      : 3004 MHz
    Applications Clocks
        Graphics                    : 875 MHz
        Memory                      : 3004 MHz
    Default Applications Clocks
        Graphics                    : 745 MHz
        Memory                      : 3004 MHz
    Max Clocks
        Graphics                    : 875 MHz
        SM                          : 875 MHz
        Memory                      : 3004 MHz
```


II. Dynamic parallelism

What is dynamic parallelism?

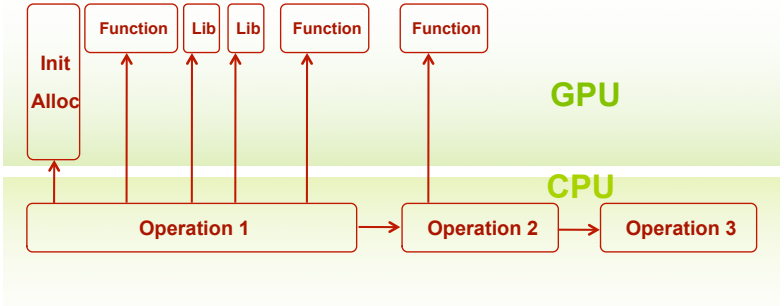
- The ability to launch new grids from the GPU:
  - Dynamically: Based on run-time data.
  - Simultaneously: From multiple threads at once.
  - Independently: Each thread can launch a different grid.

Fermi: Only CPU can generate GPU work.

Kepler: GPU can generate work for itself.

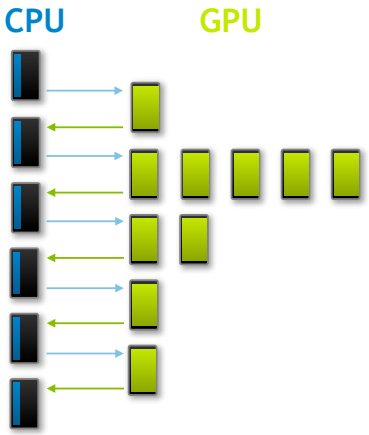
The way we did things in the pre-Kepler era: The GPU was a slave for the CPU

- High data bandwidth for communications:
  - External: More than 10 GB/s (PCI-express 3).
  - Internal: More than 100 GB/s (GDDR5 video memory and 384 bits, which is like a six channel CPU architecture).

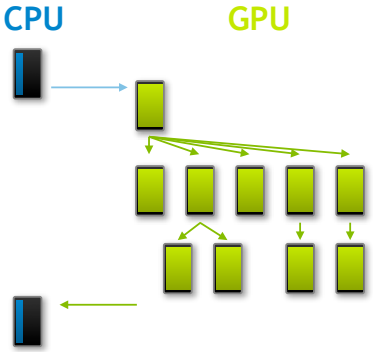


The way we do things in Kepler: GPUs launch their own kernels

The pre-Kepler GPU is a co-processor



The Kepler GPU is autonomous: Dynamic parallelism

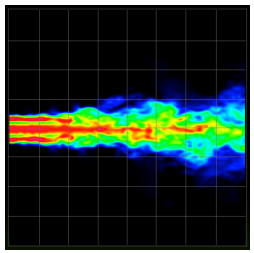


Now programs run faster and are expressed in a more natural way.

Example 1: Dynamic work generation

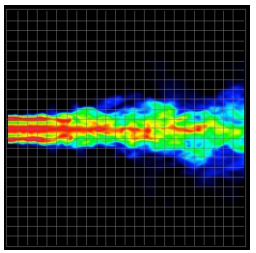
- Assign resources dynamically according to real-time demand, making easier the computation of irregular problems on GPU.
- It broadens the application scope where it can be useful.

Coarse grid



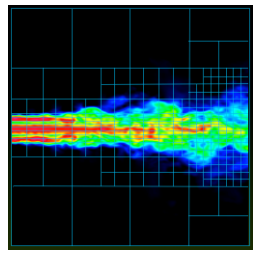
Higher performance, lower accuracy

Fine grid



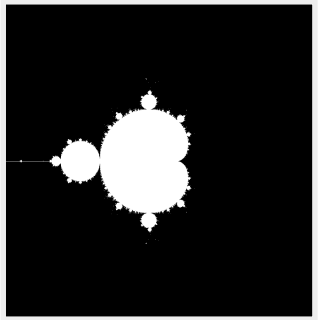
Lower performance, higher accuracy

Dynamic grid

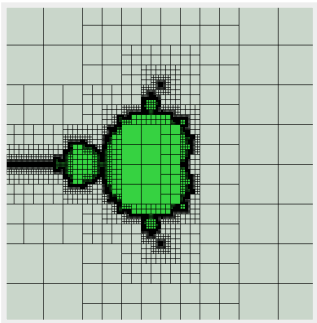


Target performance where accuracy is required

Example 2: Deploying parallelism based on level of detail



Computational power allocated to regions of interest



CUDA until 2012:
- The CPU launches kernels regularly.
- All pixels are treated the same.

CUDA on Kepler:
- The GPU launches a different number of kernels/blocks for each computational region.

Warnings when using dynamic parallelism

- It is a much more powerful mechanism than it suggests from its simplicity in the code. However...
- What we write within a CUDA kernel is replicated for **all** threads. Therefore, a kernel call will produce millions of launches if it is not used within an IF statement (which, for example, limits the launch to a single one from thread 0).
- If a father block launches sons, can they use the shared memory of their father?
  - No. It would be easy to implement in hardware, but very complex for the programmer to guarantee the code correctness (avoid race conditions).

III. Hyper-Q

Hyper-Q

- In Fermi, several CPU processes can send thread blocks to the same GPU, but the concurrent execution of kernels was severely limited by hardware constraints.
- In Kepler, we can execute simultaneously up to 32 kernels launched from different:
  - MPI processes, CPU threads (POSIX threads) or CUDA streams.
- This increments the % of temporal occupancy on the GPU.

An example: 3 streams, each composed of 3 kernels

```
__global__ kernel_A(pars) {body} // Same for B...Z
cudaStream_t stream_1, stream_2, stream_3;
...
cudaStreamCreatewithFlags(&stream_1, ...);
cudaStreamCreatewithFlags(&stream_2, ...);
cudaStreamCreatewithFlags(&stream_3, ...);
...
kernel_A <<< dimgridA, dimblockA, 0, stream_1 >>> (pars);
kernel_B <<< dimgridB, dimblockB, 0, stream_1 >>> (pars);
kernel_C <<< dimgridC, dimblockC, 0, stream_1 >>> (pars);
...
kernel_P <<< dimgridP, dimblockP, 0, stream_2 >>> (pars);
kernel_Q <<< dimgridQ, dimblockQ, 0, stream_2 >>> (pars);
kernel_R <<< dimgridR, dimblockR, 0, stream_2 >>> (pars);
...
kernel_X <<< dimgridX, dimblockX, 0, stream_3 >>> (pars);
kernel_Y <<< dimgridY, dimblockY, 0, stream_3 >>> (pars);
kernel_Z <<< dimgridZ, dimblockZ, 0, stream_3 >>> (pars);
```

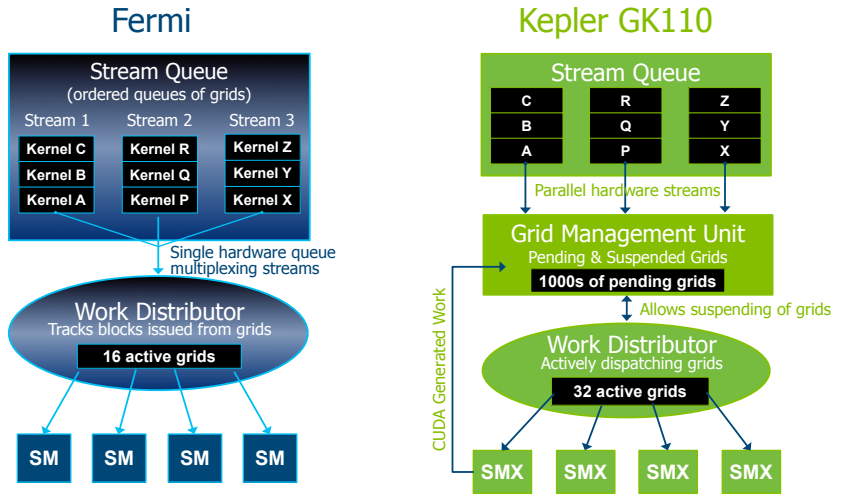
stream 1 / stream 2 / stream 3

stream_1
kernel_A
kernel_B
kernel_C

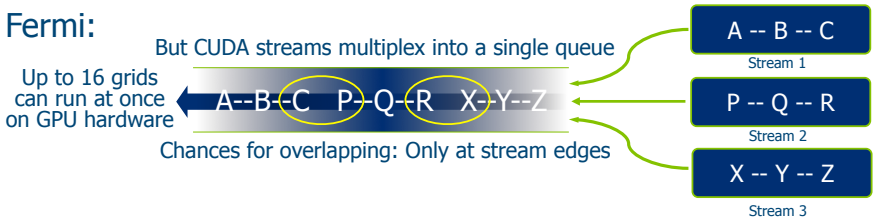
stream_2
kernel_P
kernel_Q
kernel_R

stream_3
kernel_X
kernel_Y
kernel_Z

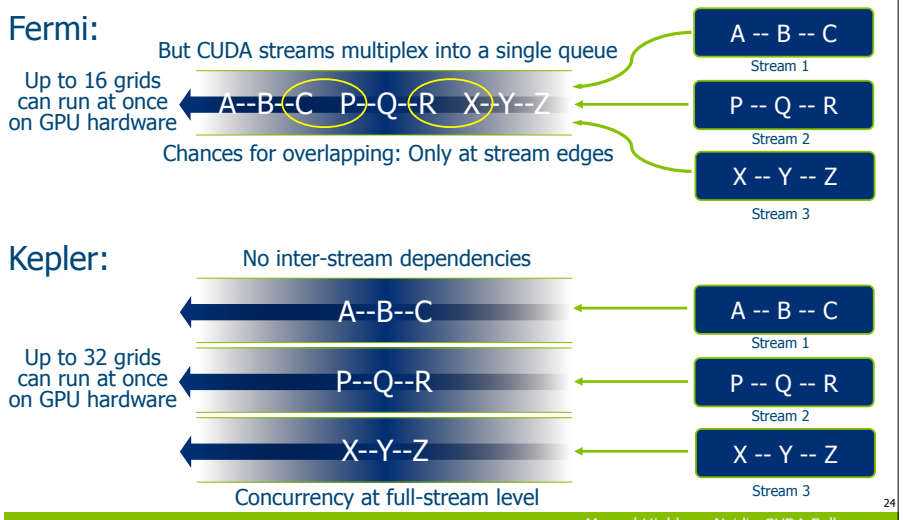
Grid management unit: Fermi vs. Kepler



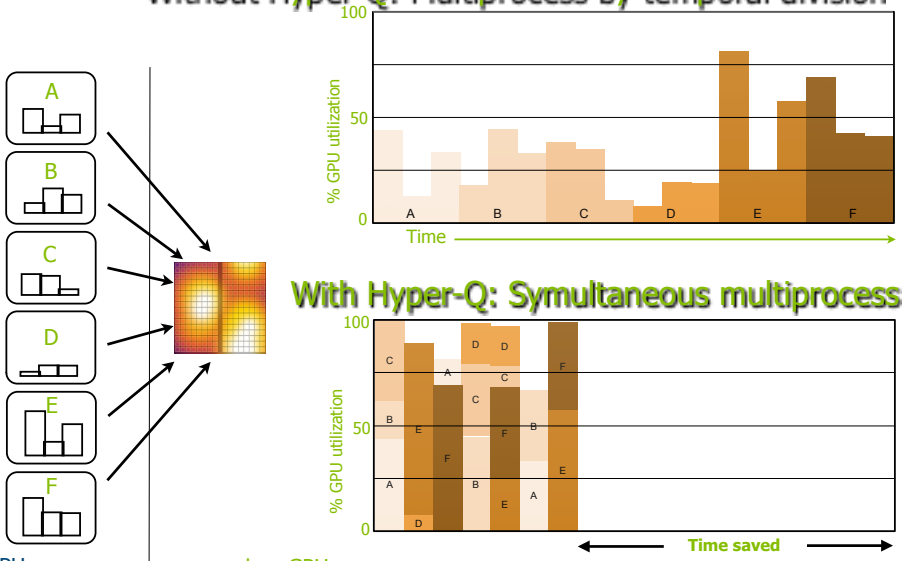
The relation between software and hardware queues



The relation between software and hardware queues



Without Hyper-Q: Multiprocess by temporal division

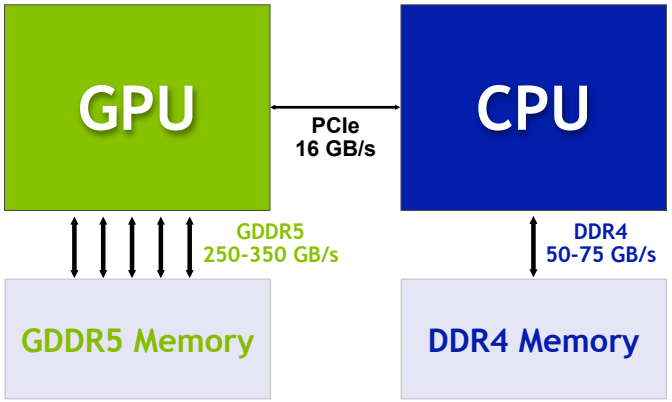


With Hyper-Q: Symultaneous multiprocess

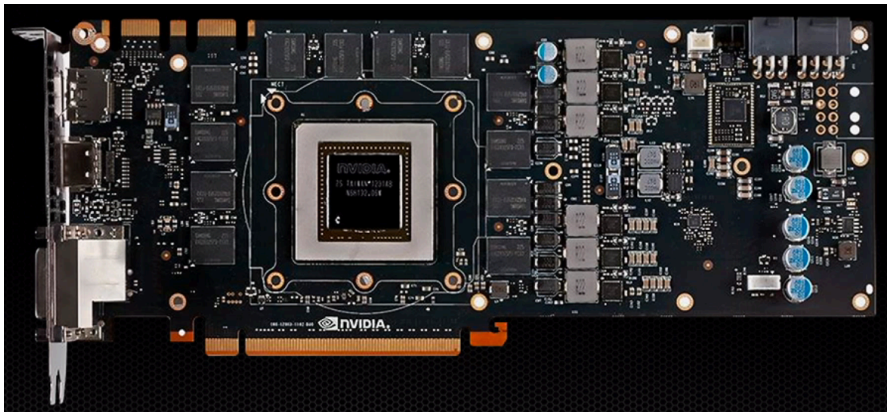
CPU processes... ...mapped on GPU

IV. Unified memory in Maxwell

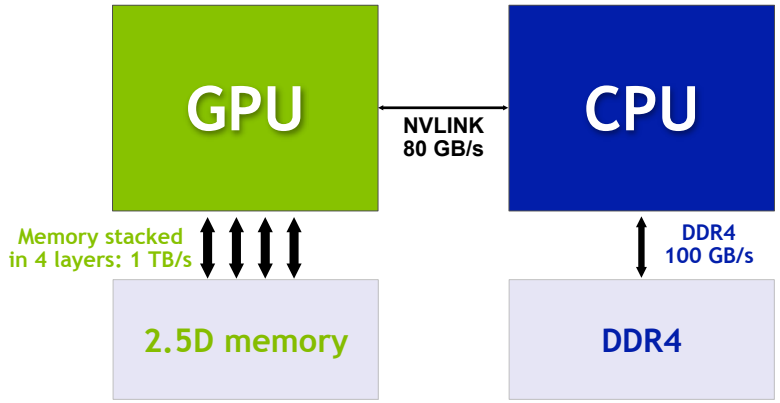
Today



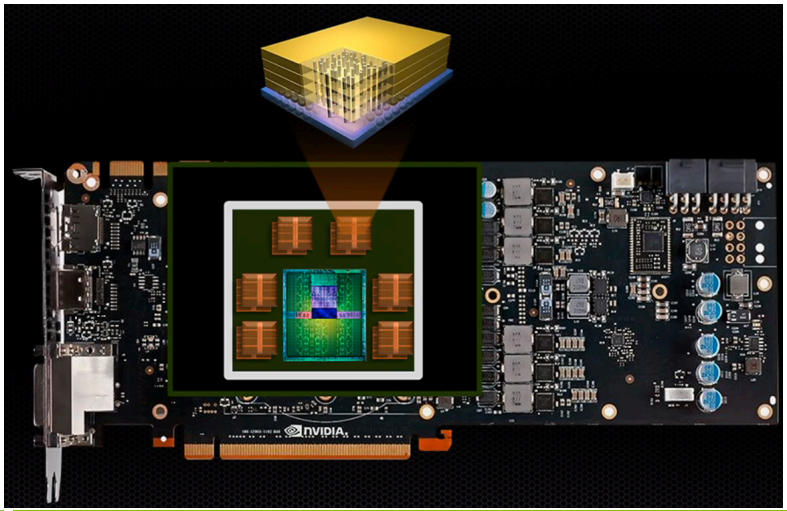
A 2014/15 graphics card: Kepler/Maxwell GPU with GDDR5 memory



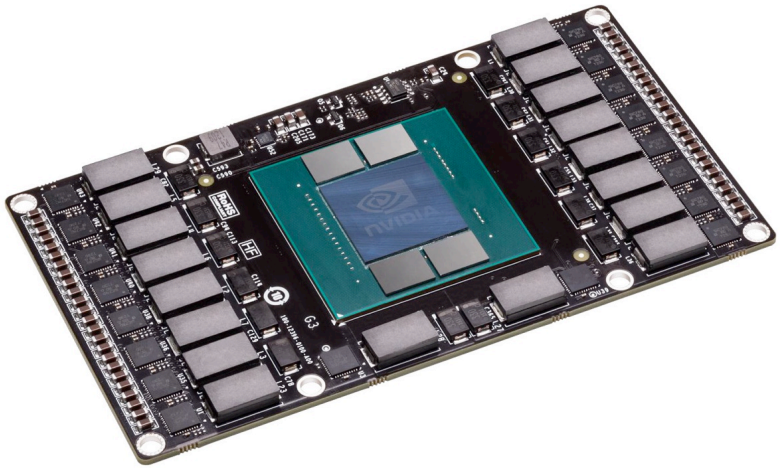
In two years



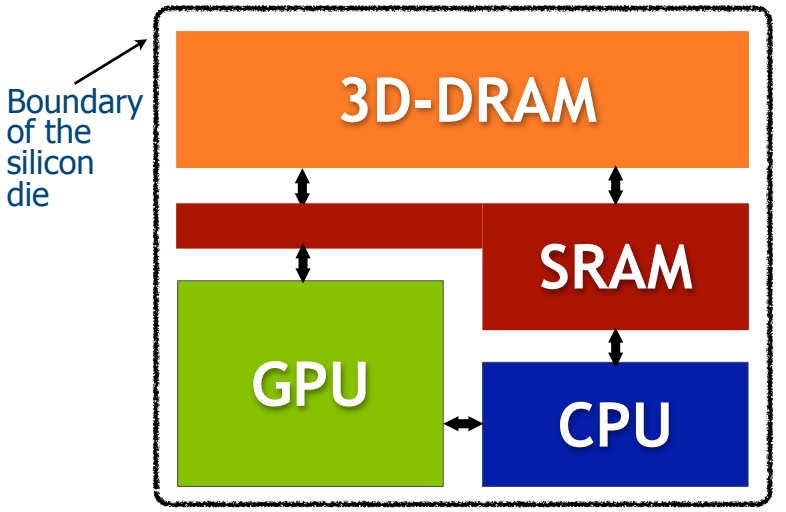
A 2016 graphics card: Pascal GPU with Stacked DRAM



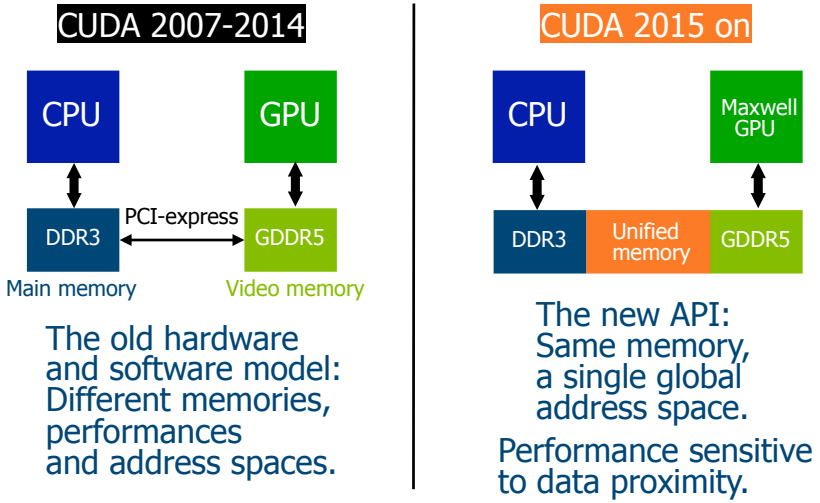
A Pascal GPU prototype



In four years: All communications internal to the 3D chip



The idea: Accustom the programmer to see the memory that way

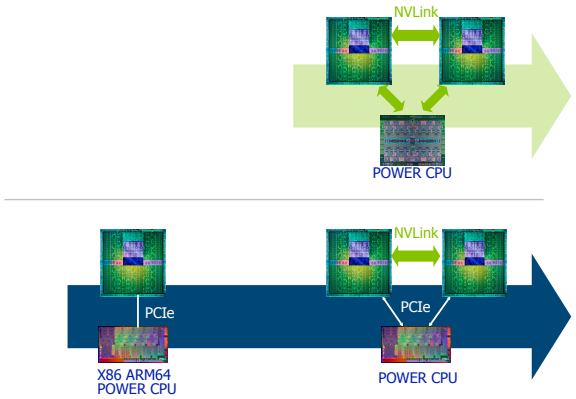


The old hardware and software model: Different memories, performances and address spaces.

The new API: Same memory, a single global address space.

Performance sensitive to data proximity.

NV-Link: High-speed GPU interconnect



2014/15: Kepler 2016/17: Pascal

Summary

- Kepler contributes to irregular computing. Now, more applications and domains can adopt CUDA. Focus: **Functionality.**
- Maxwell simplifies the GPU model to reduce power consumption and programming effort. Focus: **Low power and memory friendly.**
- **NV-Link** helps to communicate CPUs and GPUs on a transition phase towards SoC (System-on-Chip), where all main components of a computer are integrated on a single chip: CPU, GPU, SRAM, DRAM and all controllers.

Thanks for coming!

- You can always reach me in Spain at the Computer Architecture Department of the University of Malaga:
  - e-mail: ujaldon@uma.es
  - Phone: +34 952 13 28 24.
  - Web page: http://manuel.ujaldon.es (english/spanish versions available).
- Or, more specifically on GPUs, visit my web page as Nvidia CUDA Fellow:
  - http://research.nvidia.com/users/manuel-ujaldon