GPU Computing

Edwin Carlinet, Joseph Chazalon

`firstname.lastname@epita.fr`

Fall 2023

EPITA Research Laboratory (LRE)

Slides generated on September 8, 2023

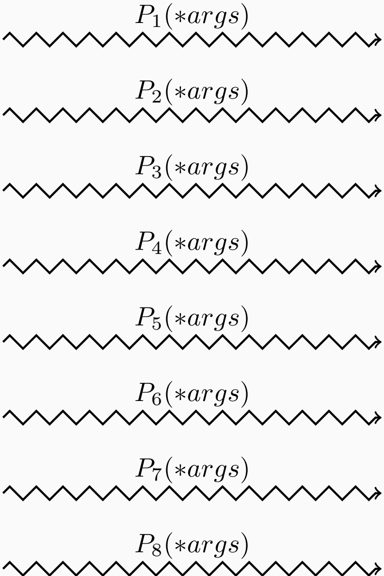
From programming model to hardware parallelism (🌶️🌶️ & 🔔)

GPU memory model

From programming model to hardware parallelism (🌶️🌶️ & 🔔)

Programming Model (1D)

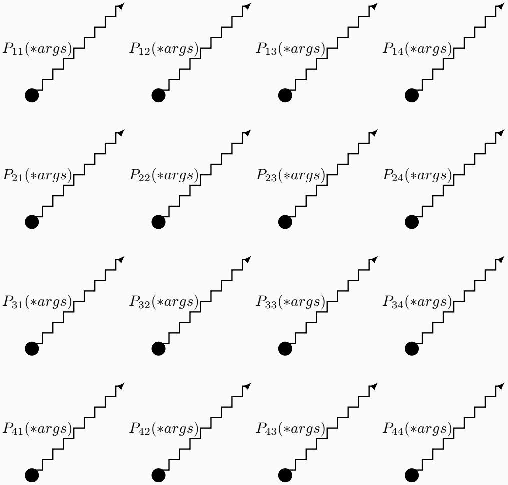
The **same** program *K* (kernel) is executed by thousands of threads The only thing that changes is the thread identifier.




```
Program P_i(in, out):
  x ← in[i]
  y ← Compute something(x)
  out[i] ← y
```

Programming Model (2D)

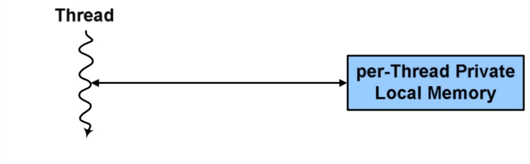
The same holds with compute grid in 2D and 3D




```
Program P_ij(in, out):
  x ← in[i][j]
  y ← Compute something(x)
  out[i][j] ← y
```

Thread

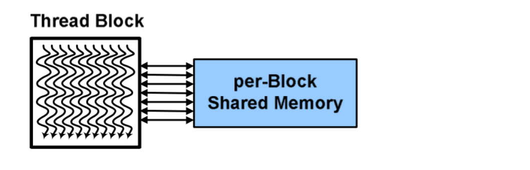
- Instance of one *kernel* (list of instructions)
- If *active*, it has a *Program Counter*, *registers*, *private memory*, IO
- Thread ID = ID a *block*



Block

A set of *threads* that cooperate:

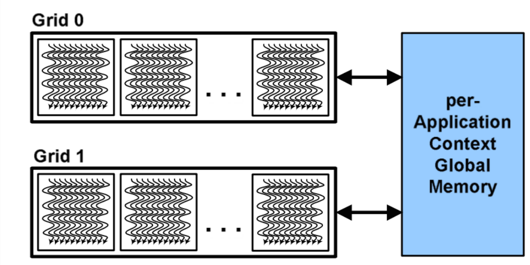
- Synchronisation
- Shared memory
- Block ID = ID in a **grid**



Grid

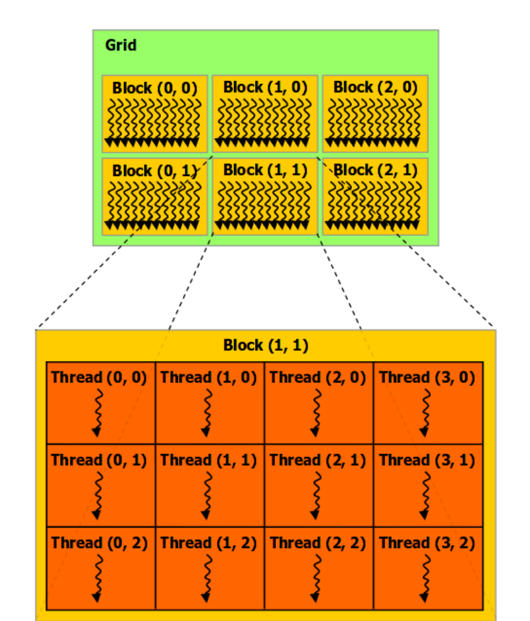
Array of blocks executing same *kernel*:

- Access to global GPU memory
- Sync. by stop and start a new kernel

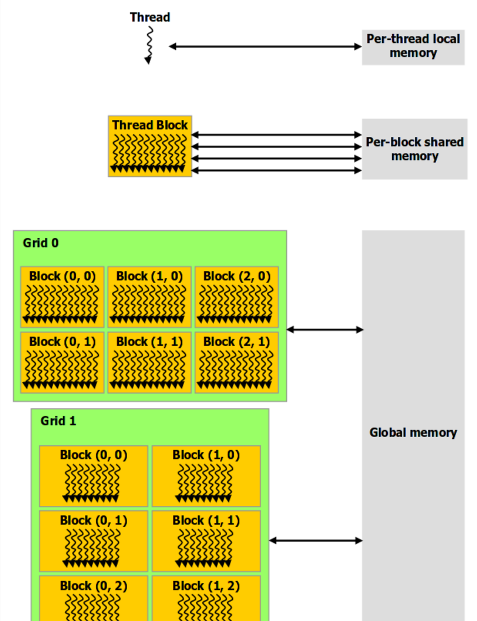


Programming Model - Summary

Hierarchy



Memory



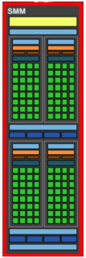
Mapping Programming model to hardware - the SMs

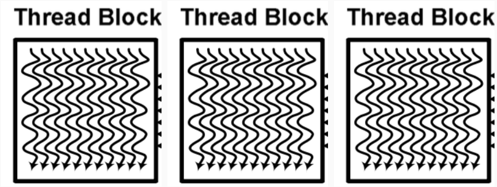
- CUDA's Thread/block/grid is mapped to GPU (N°Threads ≠ N°Cores)

- GTX 980 - 16 Streaming Multi-processors (SM)
- SMs are rather independant

Mapping Programming model to hardware - the SMs

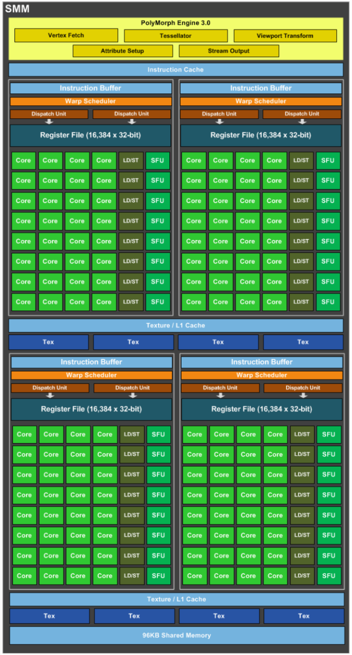
- *Any block* of *any grid* can be mapped to *any SM*





- Once *thread block* is affected to a SM, it won't move to another one.

Zoom on the SM



- SM organizes **blocks** into **warps**
- 1 warp = group of 32 threads

GTX 920:

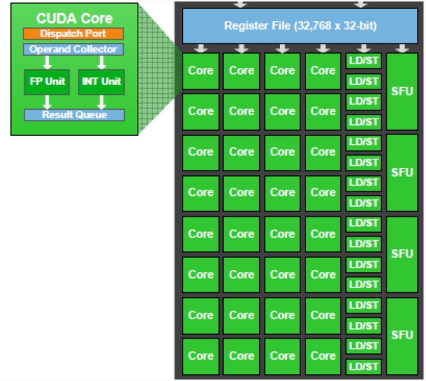
- 128 cores = 4 x 32 cores
- Quad warp scheduler selects **4** warps (**TLP**)
- And **2** independent instructions per warp can be dispatched each cycle (**ILP**)

Example:

- 1 (logical) *block* of 96 threads maps to: 3 (physical) *warps* of 32 threads

Zoom on the CUDA cores

1 core = 1 thread



- A *warp* executes 32 threads on the 32 CUDA cores
- The threads executes **the same** instruction (**DLP**)
- All instructions are SIMD
  (width = 32) instructions

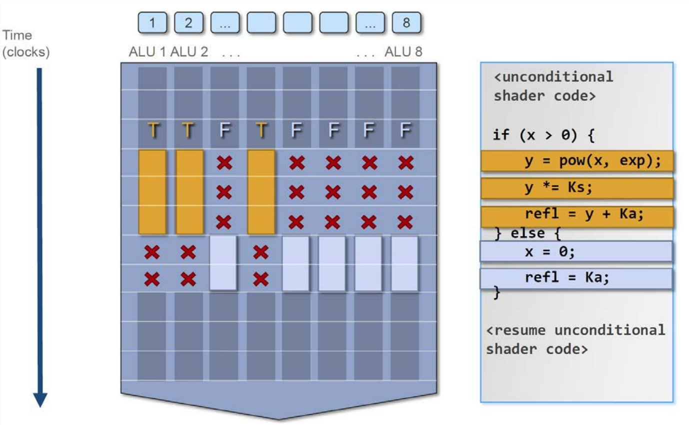
Each core:

- Floating point & Integer unit
- Fused multiply-add (FMA) instruction
- Logic unit
- Move, compare unit
- Branch unit
- The first IF/ID of the pipeline is done by the SM

Warning: SIMT allows to specify the execution and branching behavior of a single thread but maps to SIMD processors!

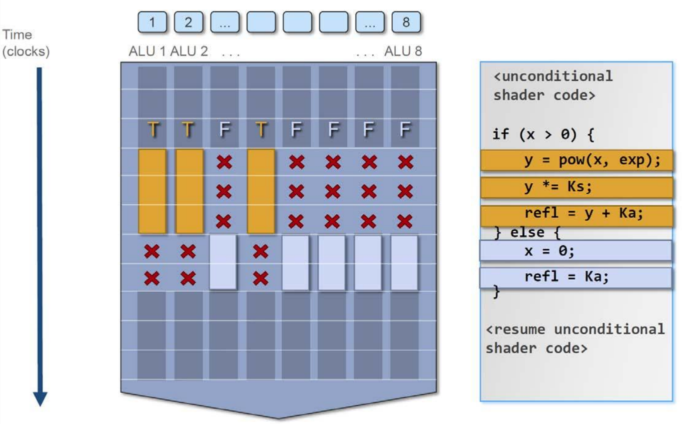
The SIMT Execution Model on CUDA Cores

- Divergent code paths (branching) pile up!



The SIMT Execution Model on CUDA Cores

- Divergent code paths (branching) pile up!



A mask allow to dis/activate threads:

If branch	1	1	0	1	0	0	0	0
Else branch	0	0	1	0	1	1	1	1

What is the latency (in term of inst) of this code in the better and worst case ?

```
if a > 0:
    inst-a
    if b > 0:
        inst-b;
    else
        inst-c
else:
    inst-d
```

The SIMT Execution Model on CUDA Cores

What is the latency (in term of inst) of this code in the better and worst case ?

```
if a > 0:
    inst-a
    if b > 0:
        inst-b;
    else
        inst-c
else:
    inst-d
```

- Best case: `a > 0` is false for every thread. For all threads: `inst-d`
- Worst case: `a > 0` and `b > 0` is true for some but not all threads. For all threads:

```
inst-a
inst-b
inst-c
inst-d
```

Loops

```
i = 0
while i < thread_id:
    i += 1
```

The SIMT Execution Model on CUDA Cores

Loops

```
i = 0
while i < thread_id:
    i += 1
```

Unrollable loops cost = max iterations, ie:

- Keep looping until all threads exit
- Mask out threads that have exited the loop

Execution trace	T0	T1	T2	T3
`i = 0`	0	0	0	0
`i < tid`	0	1	1	1
`i++`	0	1	1	1
`i < tid`	0	0	1	1
`i++`	0	1	2	2
`i < tid`	0	0	0	1
`i++`	0	1	2	3
`i < tid`	0	0	0	0

GPU memory model

Computation cost vs. memory cost

- Power measurements on NVIDIA GT200

	Energy/op (nJ)	Total power (W)
Instruction Control	1.8	18
Mult-add 32-wide warp	3.6	36
Load 128B from DRAM	80	90

With the same amount of energy:

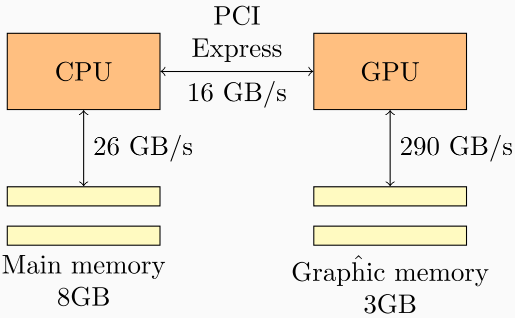
- Load 1 word from external memory (DRAM)
- Compute 44 flops

→ Must optimize memory first

External memory: discrete GPU

Classical CPU-GPU model

- Split memory space
- Highest bandwidth from GPU memory
- Transfers to main memory are slower

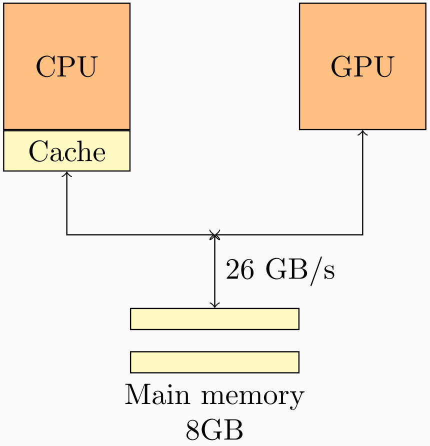


Intel i7 4770 / GTX 780

External memory: embedded GPU

Most GPUs today:

- Same memory
- May support memory coherence (GPU can read directly from CPU caches)
- More contention on external memory



Cache area in CPU vs GPU:

Figure 1-2. The GPU Devotes More Transistors to Data Processing

GPU: on-chip memory

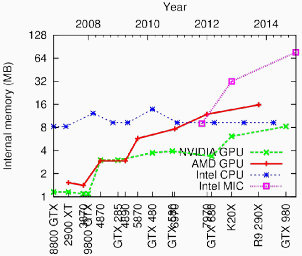
Cache area in CPU vs GPU:

Figure 1-2. The GPU Devotes More Transistors to Data Processing

But if we include registers:

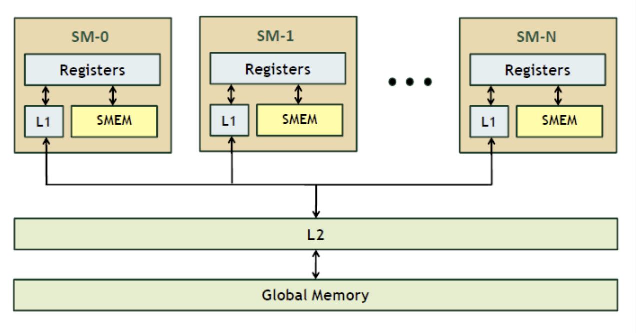
GPU	Registers files + caches
NVidia Maxwell	8.3 MB
AMD Hawaii GPU	15.8 MB
Core i7 CPU	9.3 MB

	CPU	GPU
Register / Core	256	65K



→ GPU has many more registers but made of simpler memory

Memory model hierarchy (hardware)

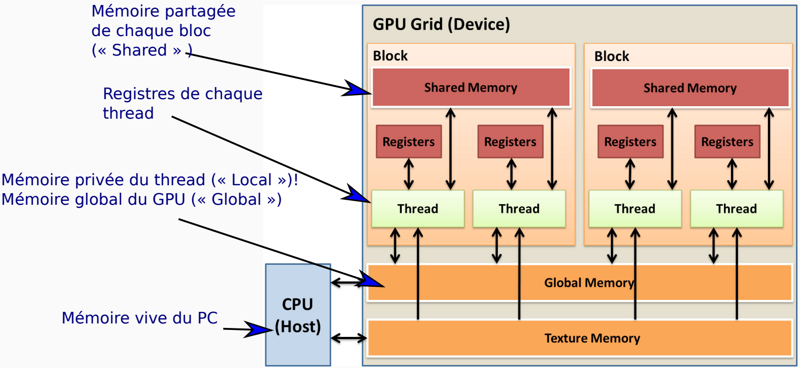


Cache hierarchy:

- Keep frequently-accessed data Core
- Reduce throughput demand on main memory L1
- Managed by hardware (L1, L2) or software (shared memory)

- On CPU, caches are designed to avoid memory latency
- On GPU, multi-threading deals with memory latency

Memory model hierarchy (software)



Memory	On chip	Cached	Access	Scope	Lifetime
Register	✓	n/a	RW	1 thread	Thread
Local	✗	✓	RW	1 thread	Thread
Shared	✓	n/a	RW	All threads block	Block
Global	✗	✓	RW	All threads + host	Host
Constant	✗	✓	R	All threads + host	Host
Texture	✗	✓	R	All threads + host	Host

Conclusion

The hierarchical programming model maps to the hierarchical hardware constraints

- Threads inside blocks can synchronise / use shared memory because they are on the same SM
- Threads in the same warp have synchronized execution because they are on the same “SIMD processor”
- Threads in different blocks are hard to synchronize/only communicate with the global memory because they may be on different SMs.

Next time...

CUDA crash course