21 Lezione

giovedì 11 aprile 2019 16:15

library driver

library to use OPENCL:

- → opencl-headers
- → Opencl-clhpp-headers
- → ocl-icd-opencl-dev
- → clinfo

libraries

Servono i driver per eseguire:

- github.com/intel/compute-runtime/releases
  ↳ source of the driver to run the opencl code

ACCELERATORS

Open CL

Interesse nel mercato GPU

INTEL APPLE

AMD NVIDIA → (CUDA)

Ati

Key concept in this development:

Host Memory

HOST

the device memory is divided in 3 parts: → Device Memory: global memory, local memory, private memory

Device Memory

PLATFORM

| has a number of

DEVICES

| contains

COMPUTE UNITS (cores)

Open CL in version 1.2 → 2.0 introduces the concept of shared virtual memory (S.V.M).

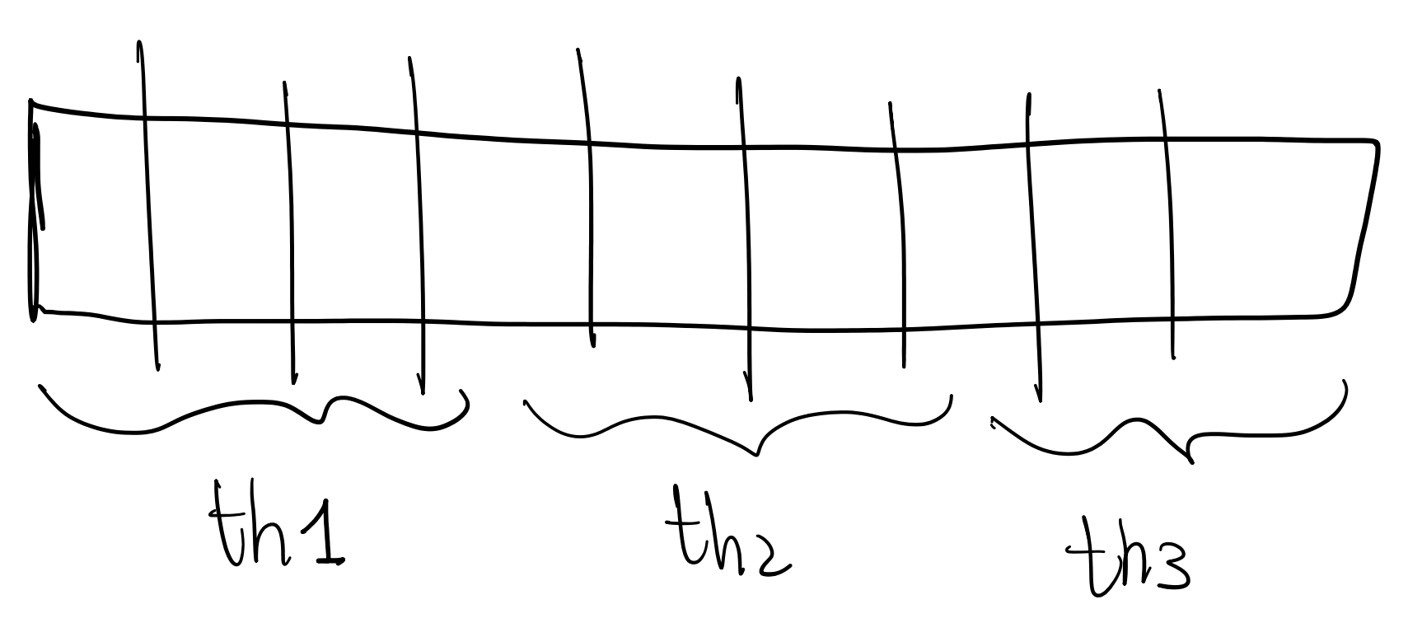
SVM is a memory abstraction.

Abstract on top of a point, 16:35

Most of the calls that you are obliged to do, now are not necessary.

2° important concept (in openCL but also in CUDA)

(16:40)



GRID of GROUPS of tasks

(16:42)

The fact that we have 1-D or 2D or 3D space → we have the ability to ask who you are depending on that dimension.

get_global_id(0) → index of the first dimension

the idea is that if I have to sum up items in 2 different vectors:

→ we have a name of the function (A, B, C) {
auto i = getglobal_ID(0);
c[i] = b[i] + a[i];
return;
}

The idea is that all the computation should be expressed in that way.

GRID
|
groups { synch
|
task

grid, groups, task

task partecipate in synchronization and this is releated with groups.

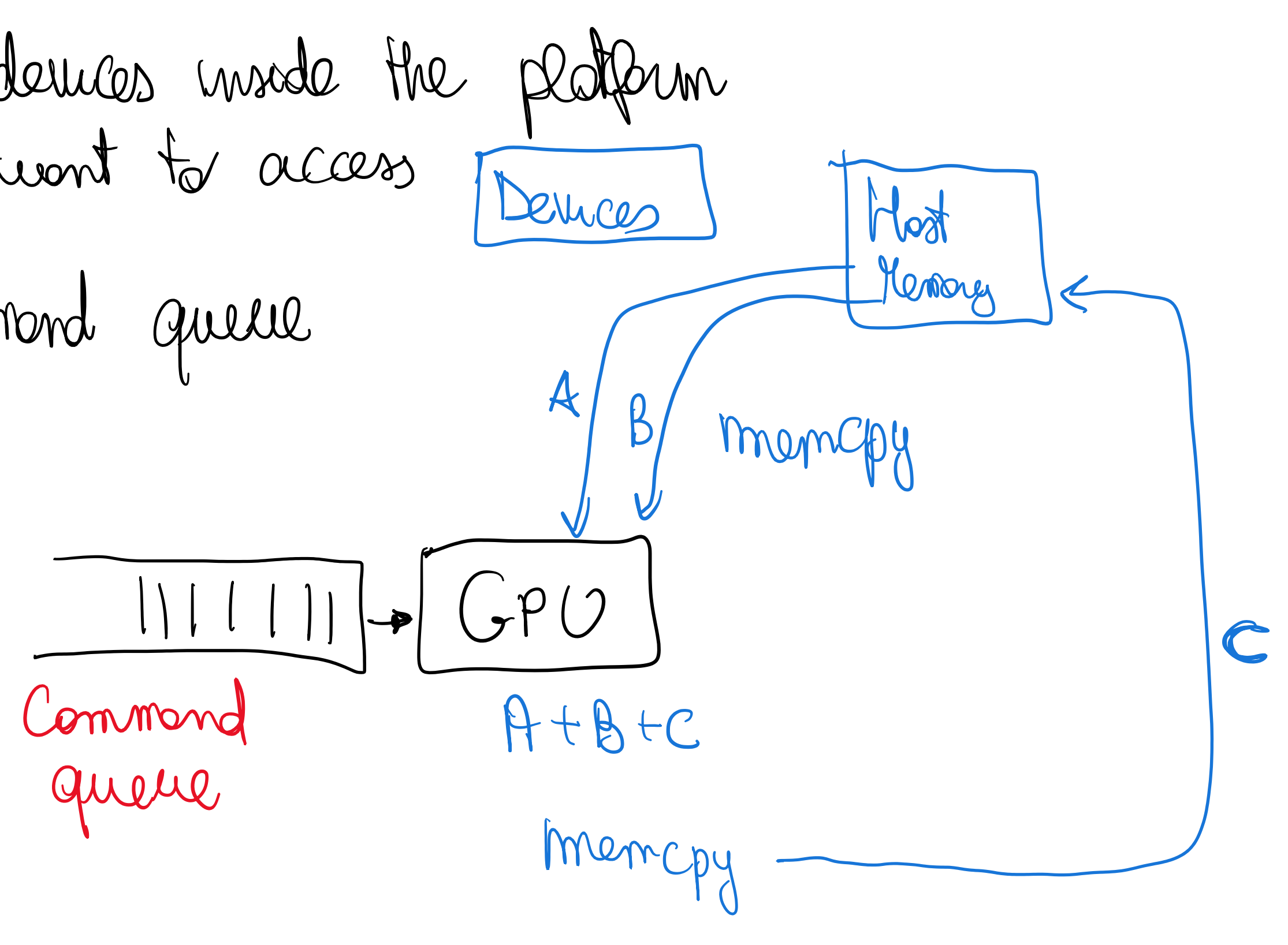
the groups are somehow intended to be (16:49)

Changing the correct group size determines the ~ (16:51)

How we direct the program to GPU?

We have different concept in openCL:

- → Platform: the devices inside the platform we want to access
- → Context: command queue



the command queue can be ordered or not ordered.

Its useful to have an unordered queue.

If we want to compute

$\vec{A} + \vec{B} \to \vec{C}$

$\vec{D} + \vec{E} \to \vec{F}$ (17:01)

Use an ordered queue and leave to GPU. ...

Some of operations can overlap.

```
"__kernel vecadd (        ){
        —
        —
        —
}
```

kernel code = code that we want to run.

Message to Conclude: **Ascoltare questa parte** ultimi 5/6 minuti della lezione

Consideriamo il vector addition code:

Quanto questo è specifico rispetto al fare le moltiplicazioni?

Apart from the kernel 15:55

I take two vector per operation ...

If I have the idea that the operation

2° message:

zip (+)

$\alpha(x^2)$

I want to take 2 vectors

$V_1 \quad V_2 \to V_3$

V_3^2

Can I use a procedure zip or can I do something more efficient?

If I use the zip function → moves the data to CPU twice.

Most of the improvement: improve the composition of the pattern.