IDIAP 2015-03-17

OpenCL

Crash Course

frederic.dubouchet@idiap.ch

Sources + slides

http://github.com/anirul/OpenCL_Crash_Course.git

- Known to work on Linux/OSX:
  - a C++ compiler (g++/clang++)
  - the OpenCL Header + libs cmake / OpenCV / Boost / GL / GLU / glut
  - Based on (Simple & Floyd-Warshall & Video):
    http://github.com/anirul/OpenCL_PA_2012.git
    http://github.com/anirul/OpenCL_Video.git
  - Warning! The Nvidia drivers on linux only support OpenCL 1.1!

Plan

- General overview (GPGPU -> OpenCL)
- Code dive (various examples)
- Conclusion (Optimisation tips)

General Overview

- GPGPU - Technology overview
- OpenCL
  - Language and API
  - Device Model
  - Objects

GPGPU

General Purpose GPU

- Using Graphical Processing Unit to compute.
- Shader languages (GLSL / DirectX)
- CUDA (Nvidia proprietary)
- DirectCompute (Windows)
- OpenACC (no free compiler support yet)

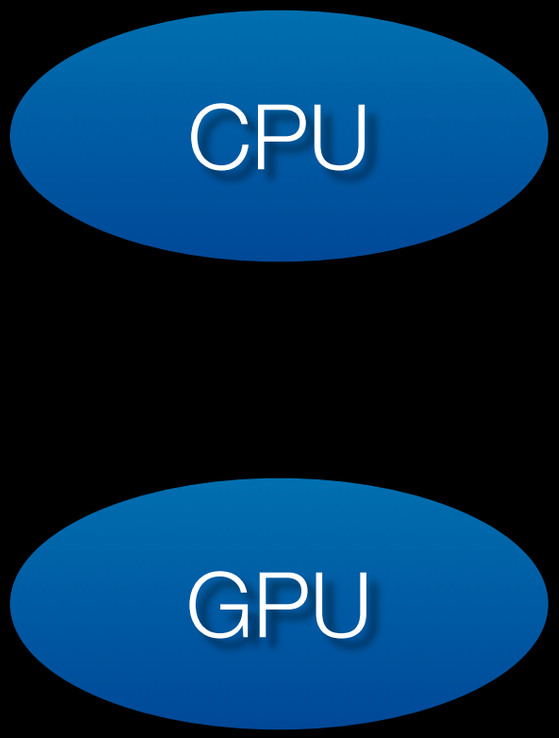
OpenCL

heterogeneous computing platforms

- Khronos (OpenGL, Vulkan, COLLADA, etc…)
  - Intel, QUALCOMM, AMD, Altera Corporation, Vivante Corporation, Xilinx, Inc., MediaTek Inc, ARM Limited, Imagination Technologies, Apple, Inc., STMicroelectronics International NV, ARM, IBM Corporation, Creative Labs, NVIDIA, Samsung Electronics.
- Work on CPU / GPU / DSP / FPGA …
- Open Standard

OpenCL

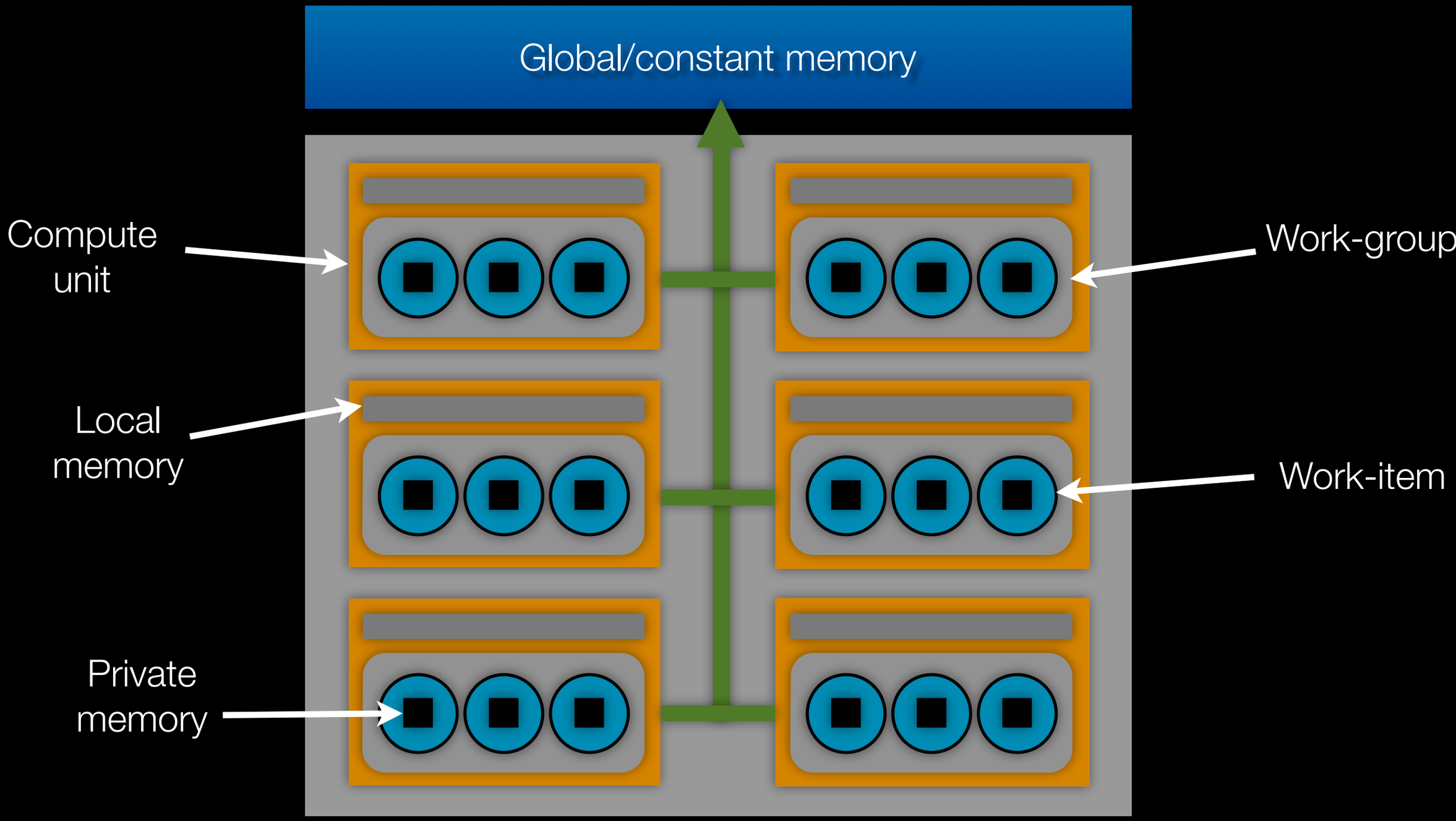
Language and API



- An API (run on the Host)
  - in C but with interface to many other languages
- A language (run on the Device)
  - vector oriented
  - C99 inspired (2.1 -> C++14)

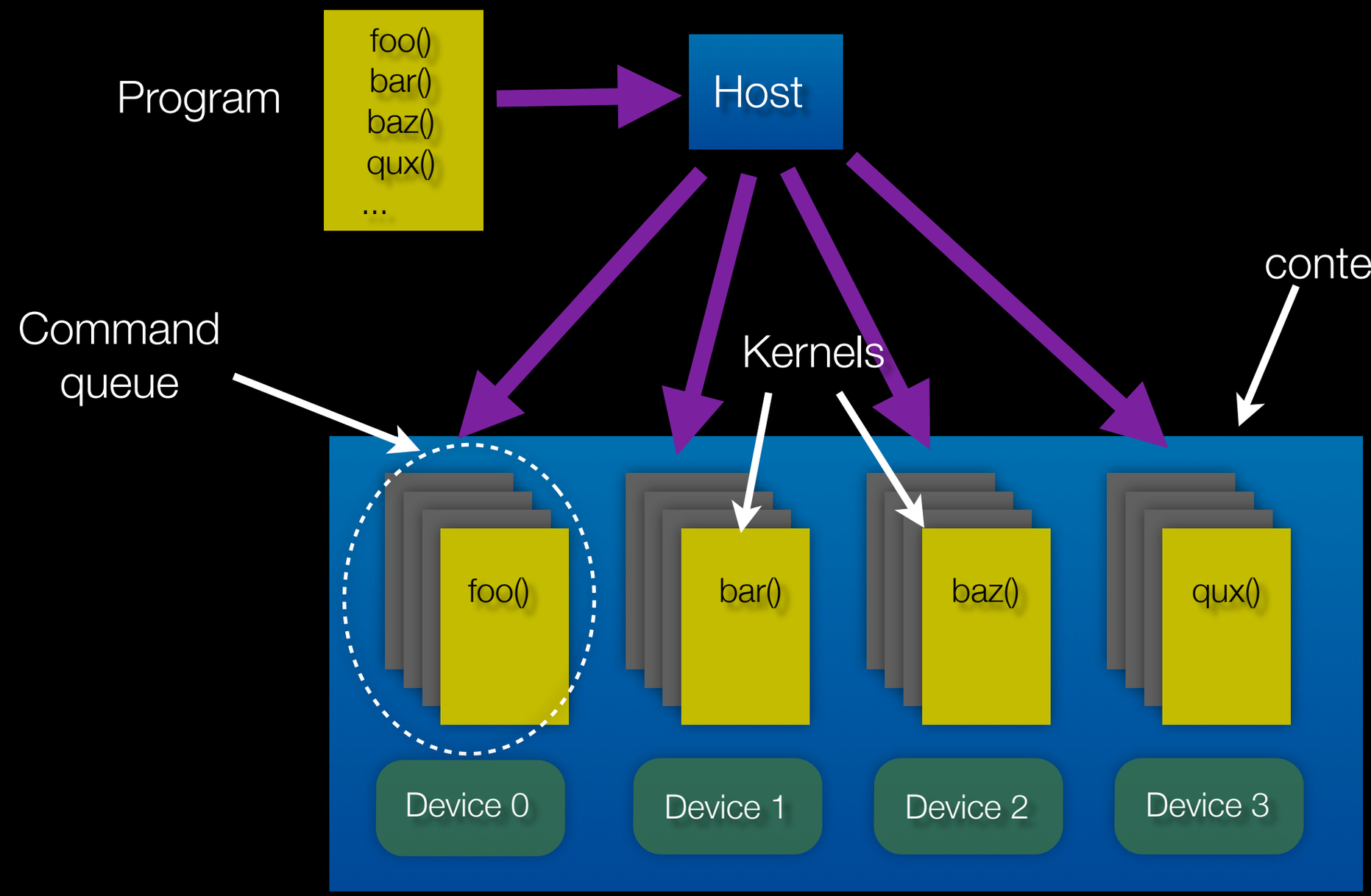
OpenCL

Device Architecture



OpenCL

Objects

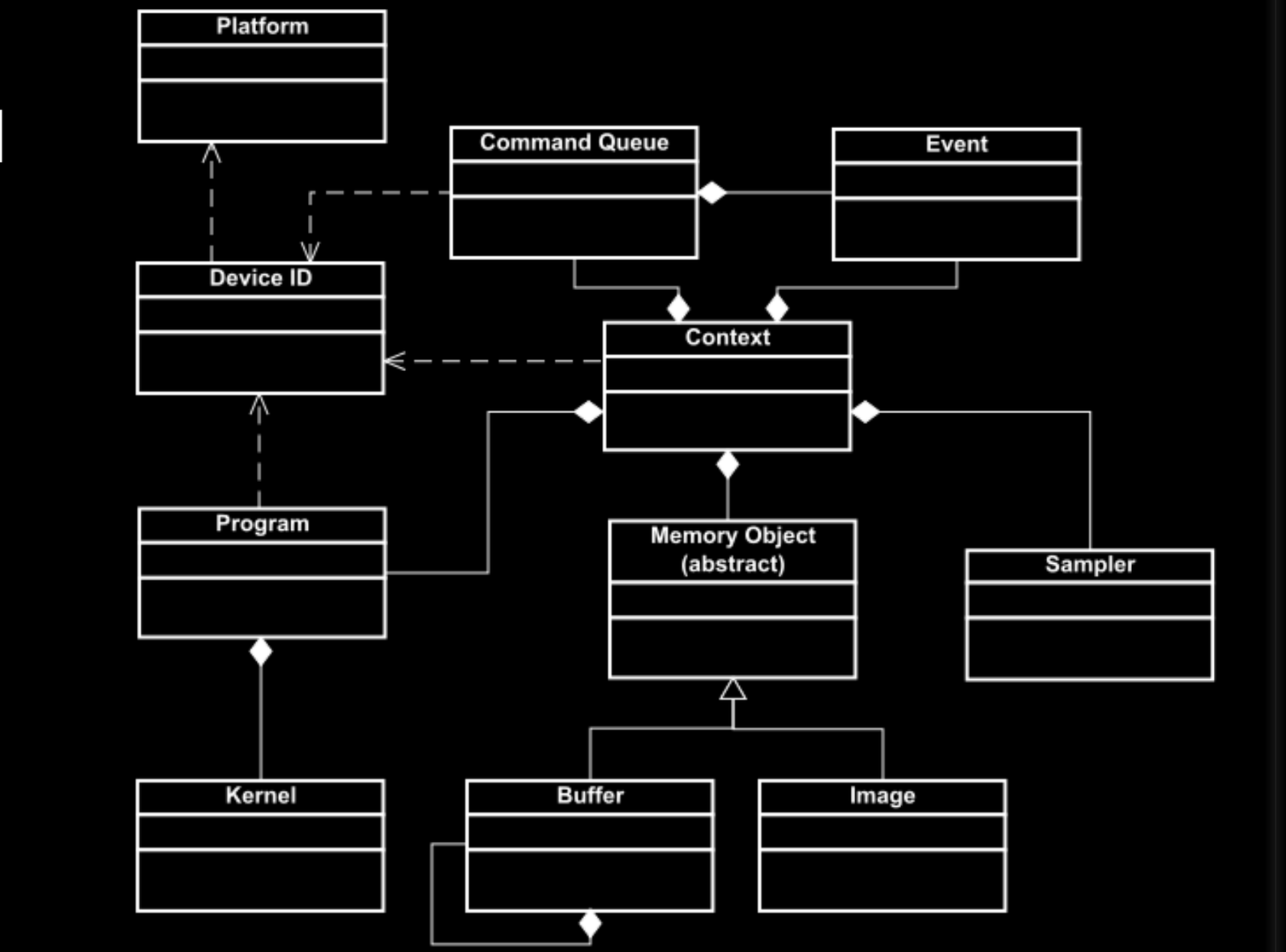


Code Dive

- Khronos C++ wrapper
- Simple - (as it can be!)
- Floyd-Warshall - (memory coalescing)
- Histogram (reduce, local memory)
- Video (the full stack)

Khronos C++ wrapper

- Officially published by Khronos
- Template based
- Header only
- 100% portable
- Object Oriented



Simple

- Very simple example using the C++ wrapper
- Compute the product of 2 vectors
- Single file only OpenCL dependent
  - simple.cpp
- Not a practical example!

Device code

```
// very simple kernel

kernel void simple(
     global read_only float* in1,
     global read_only float* in2,
     global write_only float* out)
{
   const uint pos = get_global_id(0);
   out[pos] = in1[pos] * in2[pos];
}
```

- Simple product of two vectors

Host code (1)

- Add exception support to OpenCL

```
#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>
```

- Get Platform and device

```
std::vector<cl::Platform> platforms;
cl::Platform::get(&platforms);
std::vector<cl::Device> devices_;
platforms[platform_id].getDevices(
    CL_DEVICE_TYPE_ALL,
    &devices_);
```

Host code (2)

- Generate a context

```
cl_context_properties properties[] = {
    CL_CONTEXT_PLATFORM,
    (cl_context_properties)(platforms[platform_id])(),
    0
};
cl::Context context_ = cl::Context(CL_DEVICE_TYPE_ALL, properties);
cl::CommandQueue queue_(context_, devices_[device_id], 0 , nullptr);
```

- Get the source build and select a kernel

```
cl::Program::Sources source(
    1,
    std::make_pair(
        kernel_source.c_str(),
        kernel_source.size()));
cl::Program program_(context_, source);
program_.build(devices_);
cl::Kernel kernel_(program_, "simple");
```

Host code (3)

- Create a buffer from a STL vector

```
cl::Buffer buf_in1_ = cl::Buffer(
    context_,
    CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
    sizeof(cl_float) * in1.size(),
(void*)&in1[0]);
```

- Set the arguments of the kernel

```
kernel_.setArg(0, buf_in1_);
        kernel_.setArg(1, buf_in2_);
        kernel_.setArg(2, buf_out_);
```

Host code (4)

- Execute the kernel

```
queue_.enqueueNDRangeKernel(
    kernel_,
    cl::NullRange,
    cl::NDRange(vector_size),
    cl::NullRange);
```

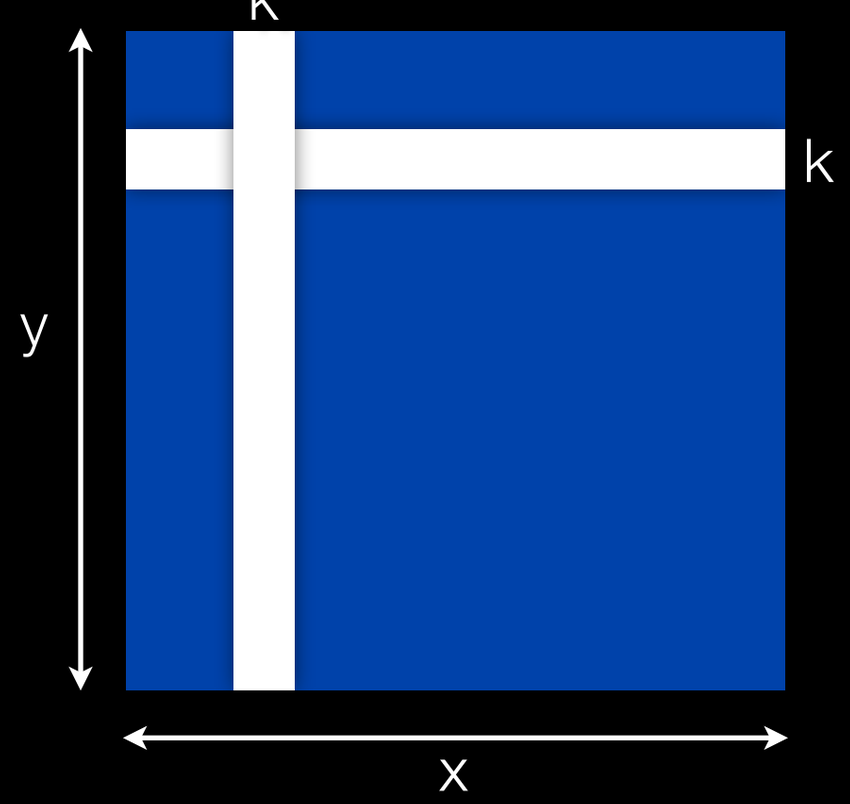
- Get the result

```
queue_.enqueueReadBuffer(
    buf_out_,
    CL_TRUE,
    0,
    vector_size * sizeof(float),
    &out[0]);
```

Floyd Warshall

For each values of k (in 0..N -1)

The Matrix

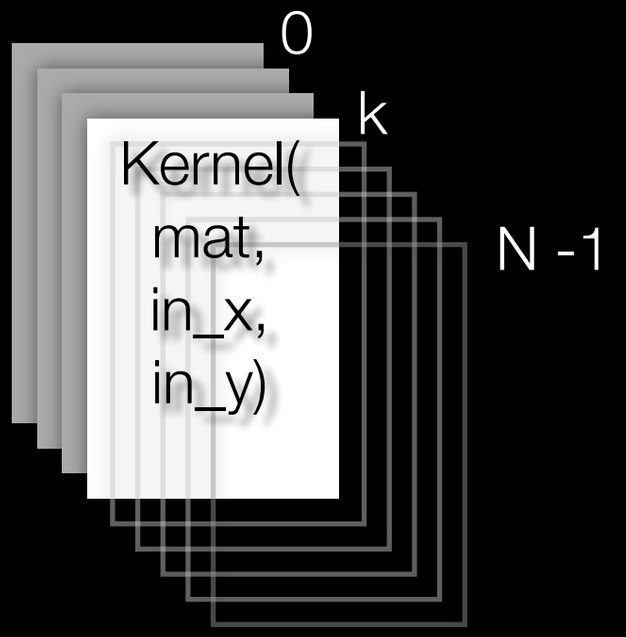


mat (whole)

in_x = y[k]

in_y = x[k]

The Kernel Stack



Device code

```
// mat the matrix
// in_x column
// in_y line
kernel void floyd_warshall_buffer(
      global float* mat,
      global float* in_x,
      global float* in_y)
{
   const int2 d = (int2)(get_global_id(0), get_global_id(1));
   const int2 m = (int2)(get_global_size(0), get_global_size(1));

   int position = d.y * m.x + d.x;
   float val1 = mat[position];
   float val2 = in_x[d.x] + in_y[d.y];
   mat[position] = (val1 < val2) ? val1 : val2;
}
```

Host code

```
for (int i = 0; i < mdx_; ++i) {
    […]
    queue_.enqueueCopyBufferRect(
        cl_buffer_mat_,
        cl_buffer_in_x_,
        […]);
    […]
    queue_.enqueueCopyBuffer(
        cl_buffer_mat_,
        cl_buffer_in_y_,
        […]);
    err_ = queue_.enqueueNDRangeKernel(
        kernel_,
        cl::NullRange,
        cl::NDRange(mdx_, mdy_),
        cl::NullRange,
        NULL,
        &event_);
    […]
}
```

Iterate on the stack

Copy Column X into buffer

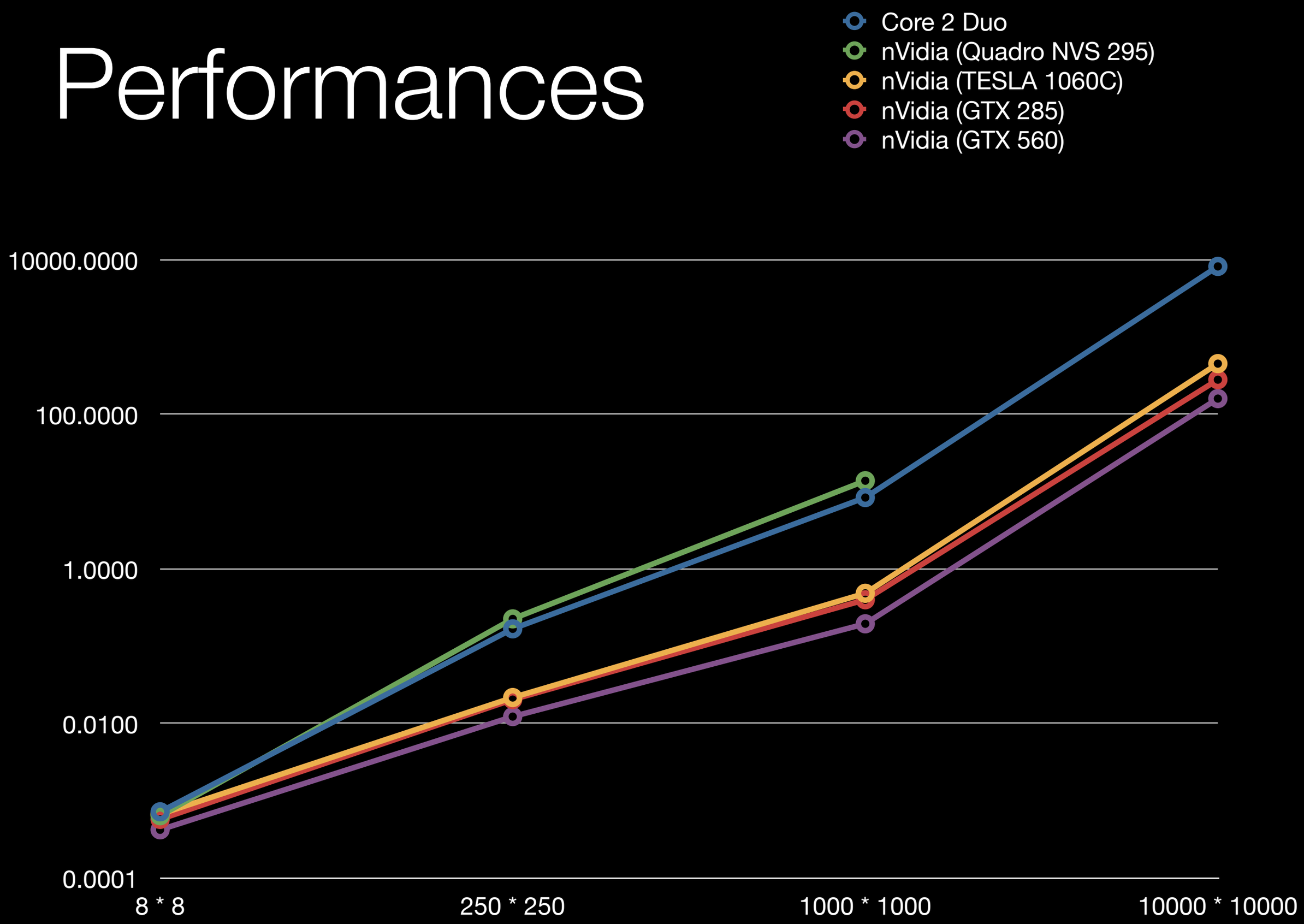
Copy line Y into buffer

Execute kernel

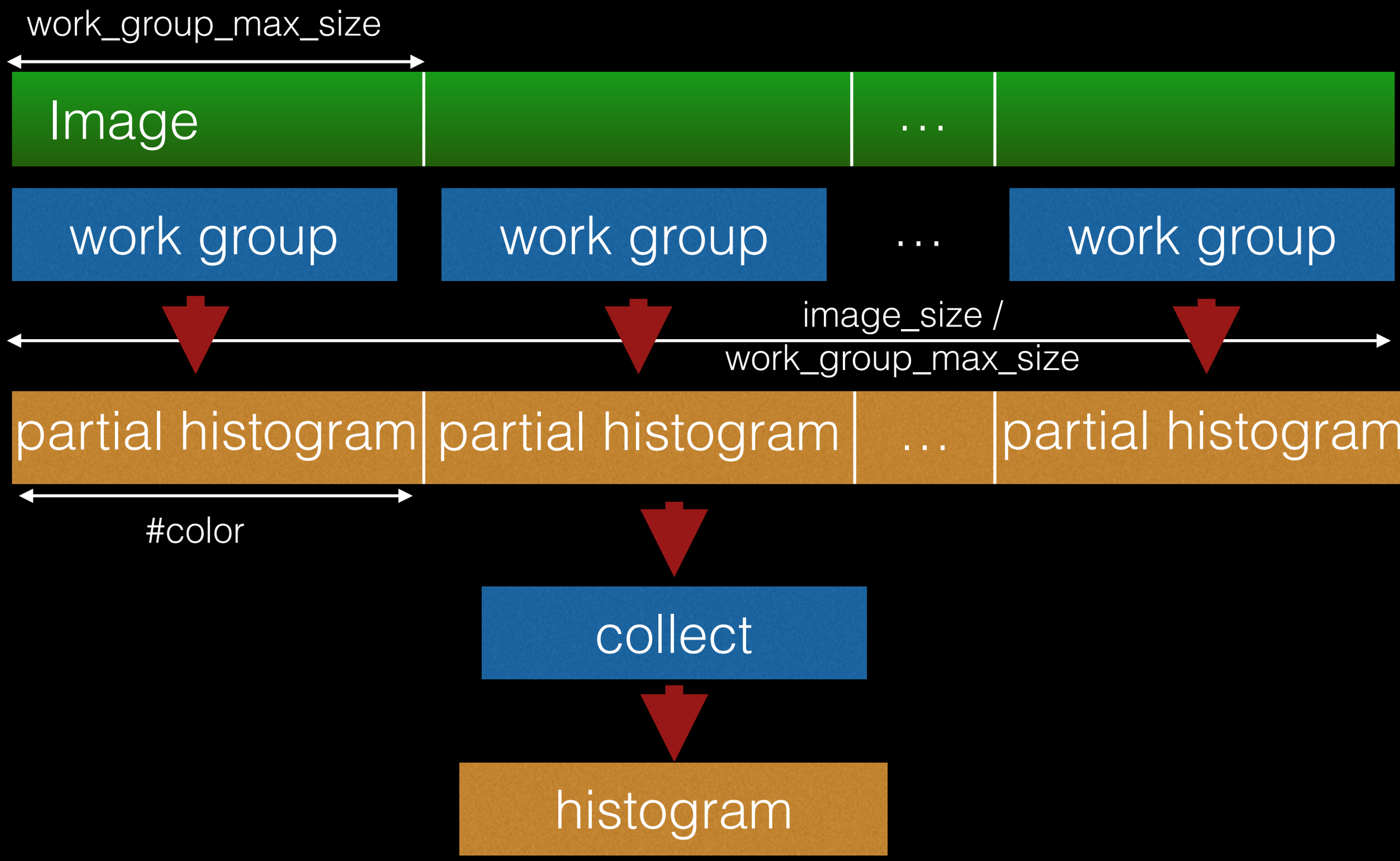
Remarks

- Memory coalescing
- Loop on kernel
- local memory is typically 64k (on Nvidia)
    - Could be computed by bloc
    - Could use local memory (per bloc)

Performances



Histogram


work_group_max_size
Image
…
work group
work group
…
work group
image_size /
work_group_max_size
partial histogram
partial histogram
…
partial histogram
#color
collect
histogram

Histogram

- Split into 4 kernels:
  - compute luminosity (on image2d)
  - clean the partial histogram buffer
  - partial histogram (to avoid atomic hell)
  - collect all partial histogram (could be split more if needed)

Device code (1)

```
constant sampler_t format =
    CLK_NORMALIZED_COORDS_FALSE |
    CLK_FILTER_NEAREST |
    CLK_ADDRESS_CLAMP;

static float luminosity_from_color(const float4 col)
{
    return 0.21f * col.x + 0.72f * col.y + 0.07f * col.z;
}

kernel void pixel_luminosity(
    read_only image2d_t img,
    global uchar* luminosity)
{
    int2 d = (int2)(get_global_id(0), get_global_id(1));
    int l = get_global_id(0) + get_global_id(1) * get_global_size(0);
    float4 col = read_imagef(img, format, d);
    float lum = luminosity_from_color(col);
    luminosity[l] = convert_uchar_sat(min(lum, 1.0f) * 255.0f);
}
```

Device code (2)

```
kernel void histogram_partial(
    global const uchar* luminosity,
    global uint* partial_histogram)
{
    int image_len = get_global_size(0) * get_global_size(1);
    int group_indx = get_global_id(1) * NB_COLOR;
    int linear_index = get_global_id(1) * get_global_size(0) +
get_global_id(0);

    if (linear_index < image_len) {
        uchar col_indx = luminosity[linear_index];
        atomic_inc(&partial_histogram[group_indx + col_indx]);
    }
}
```

Device code (3)

```
kernel void histogram_reduce(
    global const uint *partial_histogram,
    const int num_groups,
    global uint *histogram)
{
    int tid = (int)get_global_id(0);
    int group_indx;
    int n = num_groups;

    int tid_histogram = 0;

    for (int i = 0; i < num_groups * NB_COLOR; i += NB_COLOR)
        tid_histogram += partial_histogram[i + tid];

    // cumulative histogram
    histogram[tid] = tid_histogram;
}
```

Host code (1)

- Create the different kernel from the same program

```
kernel_luminosity_ = cl::Kernel(program_, "histogram_luminosity");
kernel_init_ = cl::Kernel(program_, "histogram_init");
kernel_partial_ = cl::Kernel(program_, "histogram_partial");
kernel_reduce_ = cl::Kernel(program_, "histogram_reduce");
```

Host code (2)

- Create an image2d from a pointer

```
cl::ImageFormat format = cl::ImageFormat(CL_BGRA, CL_UNORM_INT8);
cl_buffer_image_ = cl::Image2D(
context_,
CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
format,
mdx_,
mdy_,
0,
(void*)&input[0],
&err_);
```

Remarks

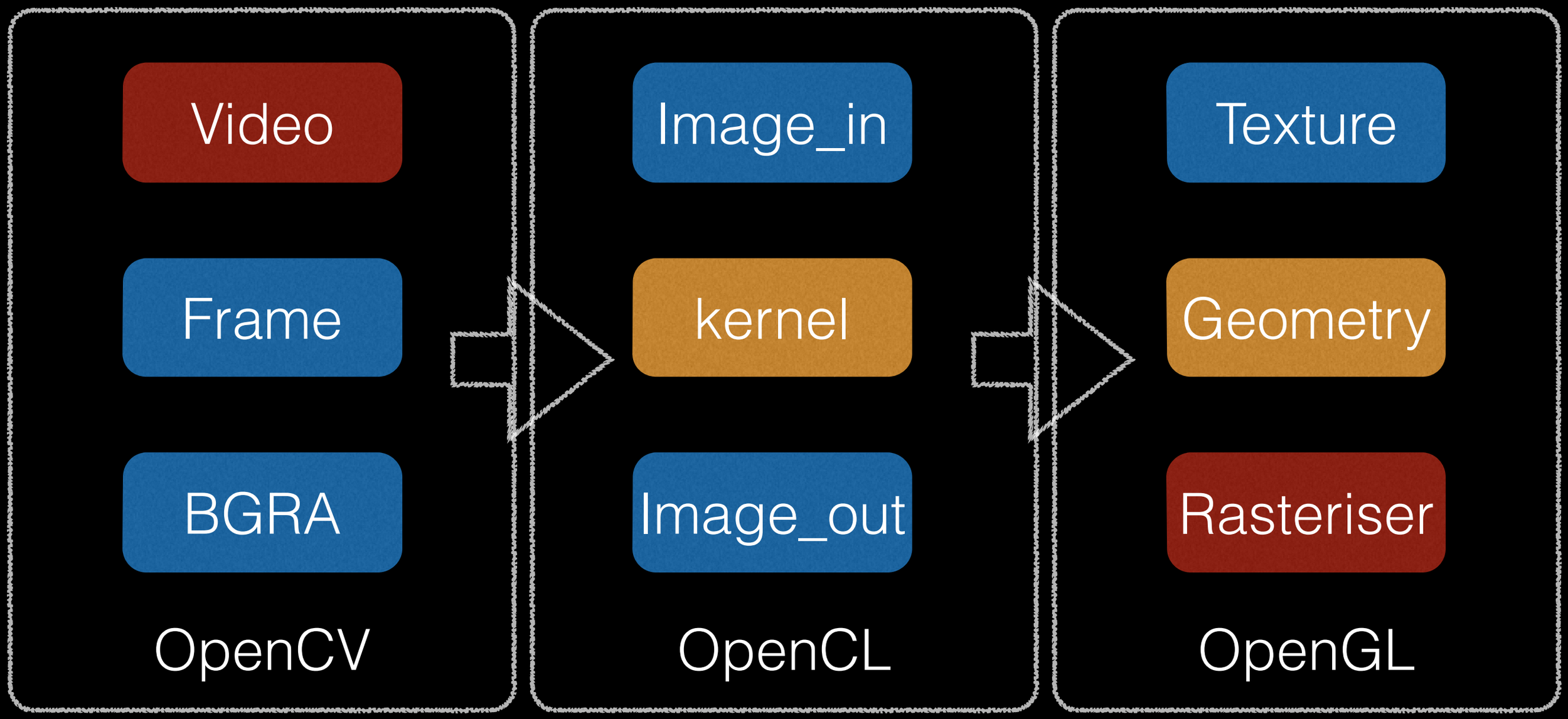
- multiple kernel in a single program
- usage of image2d!
- map and collect technique
- atomic (warning atomic only work on int!)
  (there is ways to make it work with float)
    - only work on image if:
      (image_size % WORK_GROUP_MAX_SIZE == 0)
      easy way around this
    - collect could be divided in many more step to improve performances

Performances

Computed on a full HD image (1080p)

Device	Time [s]
i7-3720QM CPU	0.004870
i7-4930K CPU	0.016011
HD Graphics 4000	0.057108
GeForce GT 650M	0.014480
GeForce GTX TITAN	0.000512

Video


Video
Frame
BGRA
OpenCV
Image_in
kernel
Image_out
OpenCL
Texture
Geometry
Rasteriser
OpenGL

Video

- OpenCV to capture video
- OpenCL to modify it
    - copy to OpenGL could be avoided with OpenCL interop (OS / Hardware dependent)
- OpenGL to draw it

Device code

- simple code to copy from image_in to image_out

```
kernel void video_image(
    read_only image2d_t in,
    write_only image2d_t out)
{
    const sampler_t format =
        CLK_NORMALIZED_COORDS_FALSE |
        CLK_FILTER_NEAREST |
        CLK_ADDRESS_CLAMP;
    const int2 d = (int2)(get_global_id(0), get_global_id(1));
    float4 col = read_imagef(in, format, d);
    write_imagef(out, d, col);
}
```

Remarks

- Kernel can be swap from the command line
- Direct feed back from the window
- Could improve the copying of the video to the OpenCL image2D (1 less copy)
- Could use OpenGL interop to directly bind the out image to the OpenGL texture (2 less copies)

Conclusion

- Third part (conclusion)
  - Optimisation tips
  - OpenGL <-> OpenCL interop
  - Vulkan / OpenACC
  - Questions?

Optimisation Tips (1)

From Nvidia

- Overall
  - Maximizing parallel execution
  - Optimizing memory usage to achieve maximum memory bandwidth
  - Optimizing instruction usage to achieve maximum instruction throughput

Optimisation Tips (2)

From Me

- Don’t trust, test!
- Sometime using image is faster than using buffers
- Use barrier([memory type]) to synchronize inside a kernel
- local memory is not playing well on CPU even if the info tells you otherwise!
- The compiler is sometime (too) clever
- Watch out, branching can kill you (or not)

OpenCL / OpenGL

Interoperation

- very platform dependent
- still a moving target
- usually drawing to the screen won’t be your bottleneck

Vulkan / OpenACC

The future?

- OpenCL 2.1 build on top of Vulkan (SPIR-V)
  - subset of the C++14 language
- Vulkan is the fusion between OpenCL and OpenGL
  - access to the full pipeline in raw mode
  - OpenCL will compile to vulkan (SPIR-V)
- OpenMP type optimisation (#pragma acc)
  - Free compiler don’t implement it yet
  - Should be supported in gcc 5

Questions?