Hands-on with alpaka

1st - 4th July 2025

Andrea Bocci

CERN - EP/CMD

last updated July 2nd, 2025

performance portability

exercise

- rewrite using alpaka a simple image processing program
  - load one or more images
  - for each image
    - display the image on the terminal
    - resize it to 50% the original size
    - convert it to greyscale
    - make 3 coloured copies
    - combine them into a larger image
    - display the combined image on the terminal
    - save the combined image as a JPEG file

exercise

- rewrite using alpaka a simple image processing program
  - load one or more images
  - for each image
    - display the image on the terminal
    - resize it to 50% the original size
    - convert it to greyscale
    - make 3 coloured copies
    - combine them into a larger image
    - display the combined image on the terminal
    - save the combined image as a JPEG file

use Sean Barret's stb image library

exercise

- rewrite using alpaka a simple image processing program
  - load one or more images
  - for each image
    - display the image on the terminal
    - resize it to 50% the original size
    - convert it to greyscale
    - make 3 coloured copies
    - combine them into a larger image
    - display the combined image on the terminal
    - save the combined image as a JPEG file

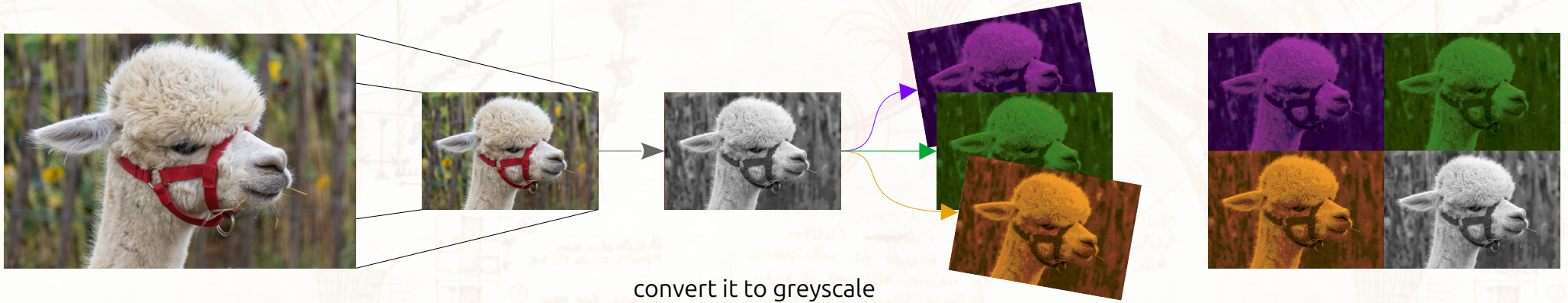
use Hayaki Saito's libsixel library

exercise

- rewrite using alpaka a simple image processing program
  - load one or more images
  - for each image
    - display the image on the terminal
    - resize it to 50% the original size
    - convert it to greyscale
    - make 3 coloured copies
    - combine them into a larger image
    - display the combined image on the terminal
    - save the combined image as a JPEG file

this is just C++
port it to GPUs using alpaka !

workflow



load an image

resize it to 50% the original size

convert it to greyscale

make 3 coloured copies

combine them into a larger image

… can run many of these workflows in parallel …

CPU-only code

- clone or update the repository with the sample code
  `git clone https://github.com/fwyzard/intro_to_alpaka.git -b bologna2025`
- or
  `git pull`
- download and build the required libraries, then build the test program
  `cd images`
  `make -j$(nproc)`
- ~~make `libsixel.so.1` available in current directory~~
  ~~`ln -s libsixel/lib/libsixel.so.1.0.6 libsixel.so.1`~~
- run the test program
  `./test`

suggestions

- start processing one image at a time
  - later you can think how to process multiple images in parallel using different queues
- ~~modify the `Image` class to use an alpaka host buffer for `data_`~~
  - wrap the `Image` `data_` in an alpaka `View` when you need to copy it to or from the device
- introduce a `DeviceImage` class that uses an alpaka device buffer for `data_`
  - pass `data_` to the kernels as a pointer or an `std::span`
- write a kernel for each operation
  - scaling
  - making a greyscale
  - tinting
  - composing
    - try to use `alpaka::memcpy(queue, dst, src)` to compose individual lines, like the original code uses `memcpy(dst, src, size)`
- do not copy images back and forth between the cpu and gpu
  - perform a single copy at the beginning, a single copy at the end, and only one synchronisation
- use a single queue to process all operations for a given image
  - later try to use multiple queues to tint and compose images in parallel, using event to synchronise them

suggestion: host object

- alpaka functions require a “device” object to operate
  - *e.g.*, to create a an alpaka `View` to some host memory you need an object representing the host
  - the host object should be unique throughout the program, and should always be available
- suggestion:
  - create a host object once as a global variable, so you don’t need to pass it around

```
// global objects for the host and device platforms
HostPlatform host_platform;
Host host = alpaka::getDevByIdx(host_platform, 0u);
```

- caveat:
  - in a “real” application we try to avoid global variables
  - wrap it inside a global accessor function

suggestion: host and device images

```
struct Image {
  ...

  auto view() {
    return alpaka::createView(host, data_, Vec1D{width_ * height_ * channels_});
  }

  auto span() {
    return std::span<unsigned char>(data_, width_ * height_ * channels_);
  }
};

struct ImageDevice {
  alpaka::Buf<Device, unsigned char, Dim1D, uint32_t> data_;
  int width_ = 0;
  int height_ = 0;
  int channels_ = 0;

  ImageDevice(Queue &queue, int width, int height, int channels)
      : data_{alpaka::allocAsyncBuf<unsigned char, uint32_t>(queue, Vec1D{width * height * channels})},
        width_{width}, height_{height}, channels_{channels}
  {}

  auto view() {
    return data_;
  }

  auto span() {
    return std::span<unsigned char>(data_.data(), width_ * height_ * channels_);
  }
};
```

suggestion: 2-dimensional loops

- we can run a single 2-dimensional loop:

```
// 2-dimensional srided loop
Vec2D size{height, width};
for (auto idx: alpaka::uniformElementsND(acc, size)) {
  int y = idx.y();
  int x = idx.x();
  ...
}
```

- or split it into inner and outer loops:

```
// outer strided loop along the Y direction
for (int y : alpaka::uniformElementsAlongY(acc, height)) {
  ...

  // inner strided loop along the X direction
  for (int x : alpaka::uniformElementsAlongX(acc, width)) {
    ...
  }
}
```

questions ?

a possible solution

solution (definitely not optimised)

- clone a dedicated branch form the repository with the sample code

```
git clone https://github.com/fwyzard/intro_to_alpaka.git -b bologna2025_solution
```

- download and build the required libraries, then build the test program

```
cd intro_to_alpaka/images/alpaka
make -j$(nproc)
```

- run the test program

```
./test_cpu
./test_cuda
./test_tbb
./test_mt
```