An introduction to alpaka

performance portability with alpaka – 7-8 March 2023

Andrea Bocci

CERN - EP/CMD

who am I

- Dr. Andrea Bocci <andrea.bocci@cern.ch>, @fwyzard on Mattermost
  - applied physicist working on the CMS experiment for over 20 years
  - at CERN since 2010
  - I’ve held various roles related to the High Level Trigger
    - started out as the b-tagging HLT contact
    - joined as (what today is called) HLT STORM convener
    - deputy Trigger Coordinator and Trigger Coordinator
    - HLT Upgrade convener, and editor for the DAQ and HLT Phase-2 TDR
    - currently, “GPU Trigger Officer”
  - for the last 5 years, I’ve been working on GPUs and *performance portability*
    - together with Matti and a few CERN colleagues
    - “Patatrack” pixel track and vertex reconstruction running on GPUs
    - R&D projects on CUDA, Alpaka, SYCL and Intel oneAPI
    - support for CUDA, HIP/ROCm, and Alpaka in CMSSW
    - Patatrack Hackathons !

performance portability

what is *portability* ?

- what do we mean by software *portability* ?
  - the possibility of running a software application or library on different platforms
    - different hardware architectures, different operating systems
    - e.g. Windows running on x86, OSX running on ARM, Linux running on IBM Power, *etc.*

- how do we achieve software *portability* ?
  - write software using a standardised language
    - C++, python, Java, *etc.*
  - use standard features
    - IEEE floating point numbers
  - use standard or portable libraries
    - C++ standard library, Boost, Eigen, *etc.*

portability: an example

- for example

https://github.com/fwyzard/intro_to_alpaka/blob/master/portability/00_hello_world.cc

```
#include <cmath>
#include <cstdio>

void print_sqrt(double x) {
  printf("The square root of %g is %g\n", x, std::sqrt(x));
}

int main() {
  print_sqrt(2.);
}
```

should behave in the same way on all platforms that support a standard C++ compiler:

```
The square root of 2 is 1.41421
```

what about GPUs ?

- writing a program that offloads some of the computations to a GPU is somewhat different from writing a program that runs just on the CPU
  - inside a single application …
  - … different hardware architectures
  - … different memory spaces
  - … different way to call a function or launch a task
  - … different optimal algorithms
  - … different compilers
  - … different programming languages !

- sometimes it may help to think about a GPU like programming a remote machine
  - compile for completely different targets
  - launching a kernel is similar to running a complete program !

portability: the same example

https://github.com/fwyzard/intro_to_alpaka/blob/master/portability/01_hello_world.cu

```
#include <cmath>
#include <cstdio>
#include <cuda_runtime.h>

__device__
void print_sqrt(double x) {
  printf("The square root of %g is %g\n", x, std::sqrt(x));
}

__global__
void kernel() {
  print_sqrt(2.);
}

int main() {
  kernel<<<1, 1>>>();
  cudaDeviceSynchronize();
}
```

```
The square root of 2 is 1.41421
```

portability: side by side

```cpp
#include <cmath>
#include <cstdio>

void print_sqrt(double x) {
  printf("The square root of %g is %g\n", x, std::sqrt(x));
}

int main() {
  print_sqrt(2.);
}
```

```
The square root of 2 is 1.41421
```

```cpp
#include <cmath>
#include <cstdio>
#include <cuda_runtime.h>

__device__
void print_sqrt(double x) {
  printf("The square root of %g is %g\n", x, std::sqrt(x));
}

__global__
void kernel() {
  print_sqrt(2.);
}

int main() {
  kernel<<<1, 1>>>();
  cudaDeviceSynchronize();
}
```

```
The square root of 2 is 1.41421
```

- we could
  - wrap the differences in a few macros or classes
  - share the common parts

so… are we done ?

- not really
  - trivially extending our example to an expensive computation would give horrible performance !

- why ?
  - a CPU will run a single-threaded program very efficiently
  - a GPU would perform horribly
    - use a single thread out of a whole warp (32 threads): use *at most* 3% of its computing power
    - use a single block: loose any possibility of hiding memory latency
    - cannot take advantage of advanced capabilities like atomic operations, shared memory, *etc.*
  - and what about different GPU back-ends ?

- what we need is *performance portability*
  - write code in a way that can run on multiple platforms
  - leverage their potential
  - and achieve (almost) native performance on all of them

performance portability ?

OpenCL™

OpenMP®

OpenACC

alpaka

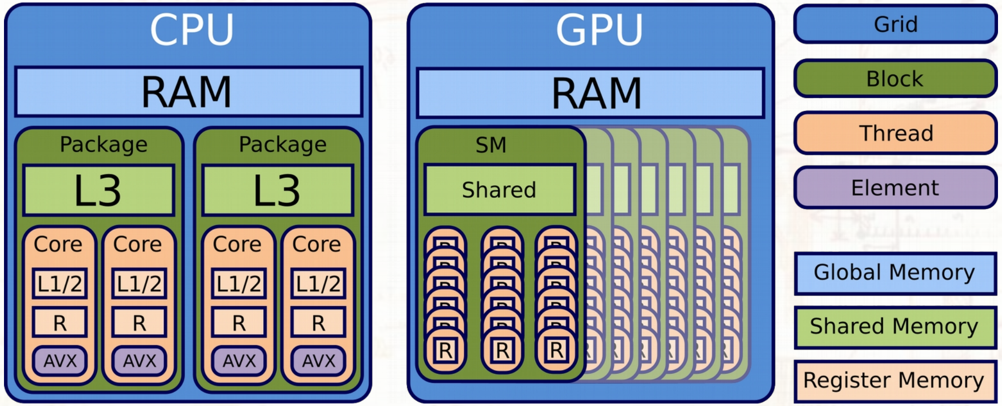
SYCL™

Kokkos

the alpaka performance portability library

what is alpaka ?

- alpaka is a header-only C++17 abstraction library for accelerator development
  - it aims to provide *performance portability* across accelerators through the abstraction of the underlying levels of parallelism
- it currently supports
  - CPUs, with serial and parallel execution
  - GPUs by NVIDIA, with CUDA
  - GPUs by AMD, with HIP/ROCm
  - support for Intel GPUs and FPGAs is *under development*, based on SYCL and Intel oneAPI



- it is easy to integrate in an existing project
  - write code once, use a Makefile of CMake to build it for multiple backends
  - a *single application* can supports all the different backends *at the same time*
- the latest documentation is available at https://alpaka.readthedocs.io/en/latest/index.html

setting up alpaka

- download the latest version of alpaka from GitHub
  - use the version that was current on March 1st 2023, to make sure the examples will work as expected
  - for a new project you should usually take the most recent version
  - these examples are likely to work anyway

```
# alpaka requires c++17 - we need a more recent version of gcc
source scl_source enable devtoolset-11

# alpaka requires Boost 1.74 or newer - you can find a prebuilt version at
export BOOST_BASE=~abocci/public/boost

# define a directory for the alpaka library
export ALPAKA_BASE=~/private/alpaka

# clone the latest version of alpaka into a predefined directory
git clone https://github.com/alpaka-group/alpaka $ALPAKA_BASE

# make sure to use a well-defined version of the library
cd $ALPAKA_BASE
git reset --hard 8ea325d3
cd -
```

setting up alpaka

- download the latest version of alpaka from GitHub
  - use the version that was current on March 1st 2023, to make sure the examples will work as expected
  - for a new project you should usually take the most recent version
  - these examples are likely to work anyway

```
# alpaka requires c++17 - we need a more recent version of gcc
source scl_source enable devtoolset-11

# alpaka requires Boost 1.74 or newer - you can find a prebuilt version at
export BOOST_BASE=~abocci/public/boost

# define a directory for the alpaka library
export ALPAKA_BASE=~/private/alpaka

# clone the latest version of alpaka into a predefined directory
git clone https://github.com/alpaka-group/alpaka $ALPAKA_BASE

# make sure to use a well-defined version of the library
cd $ALPAKA_BASE
git reset --hard 8ea325d3
cd -
```

this part sets up the environment

make sure to do it in every session

how does it work ?

- Alpaka internally uses preprocessor symbols to enable the different backends:
  - `ALPAKA_ACC_GPU_CUDA_ENABLED` for running on NVIDIA GPUs
  - `ALPAKA_ACC_GPU_HIP_ENABLED` for running on AMD GPUs
  - `ALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED` for running serially on a CPU
  - …

- in this tutorial we will build separate applications from each example
  - each application is compiled with the corresponding compiler (`g++`, `nvcc`, `hipcc`, …)
  - each application uses a single back-end

- it is also possible to enable more than one back-end at a time
  - however, the underlying CUDA and HIP header files will clash, so one needs to play some tricks with forward declarations, or use separate the compilation for the different backends
  - and separate the host and device parts

alpaka core concepts

Host-side API

- initialisation and device selection: `Platforms` and `Devices`
- asynchronous operations and synchronisation: `Queues` and `Events`
- owning memory `Buffers` and non-owning memory `Views`
- submitting work to devices: work division and `Accelerators`

Device-side API

- plain C++ for device functions and kernels
- shared memory, atomic operations, and memory fences
- primitives for mathematical operations
- warp-level primitives for synchronisation and data exchange *(not covered)*
- random number generator *(not covered)*

nota bene:

- most Alpaka API objects behave like `shared_ptrs`, and should be passed by value or by reference to const (*i.e.* `const&`)

platforms and devices

alpaka: initialisation and device selection

`Platform` and `Device`

- identify the type of hardware (*e.g.* host CPUs or NVIDIA GPUs) and individual devices (*e.g.* each single GPU) present on the machine
- the CPU device `DevCpu` serves two purposes:
  - as the "host" device, for managing the data flow (*e.g.* perform memory allocation and transfers, launch kernels, *etc.*)
  - as an "accelerator" device, for running heterogeneous code (*e.g.* to run an algorithm on the CPU)
- platforms cannot be instantiated, they are only used as a type
- devices should be created at the start of the program and used consistently
- some common cases

back end	alpaka platform	alpaka device
CPUs, serial or parallel	`PltfCpu`	`DevCpu`
NVIDIA GPU, with CUDA	`PltfCudaRt`	`DevCudaRt`
AMD GPUs, with HIP/ROCm	`PltfHipRt`	`DevHipRt`

platforms and devices

- Alpaka provides a simple API to enumerate the devices on a given platform:

  - `alpaka::getDevCount<Platform>()`
    - returns the number of devices on the given platform

  - `alpaka::getDevByIdx<Platform>(index)`
    - initialises the `index` device on the platform, and returns the corresponding `Device` object

  - `alpaka::getName(device)`
    - returns the name of the given device

your first alpaka application

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/00_enumerate.cc

```
int main() {
  // the host abstraction always has a single device
  Host host = alpaka::getDevByIdx<HostPlatform>(0u);

  std::cout << "Host platform: " << alpaka::core::demangled<HostPlatform> << '\n';
  std::cout << "Found 1 device:\n";
  std::cout << "  - " << alpaka::getName(host) << '\n';
  std::cout << std::endl;

  // enumerate the devices on the accelerator platform
  std::vector<Device> devices;
  std::size_t n = alpaka::getDevCount<Platform>();
  devices.reserve(n);
  for (std::size_t i = 0; i < n; ++i) {
    devices.push_back(alpaka::getDevByIdx<Platform>(i));
  }

  std::cout << "Accelerator platform: " << alpaka::core::demangled<Platform> << '\n';
  std::cout << "Found " << devices.size() << " device(s):\n";
  for (auto const& device: devices)
    std::cout << "  - " << alpaka::getName(device) << '\n';
  std::cout << std::endl;
}
```

your first alpaka application

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/00_enumerate.cc

```
int main() {
  // the host abstraction always has a single device
  Host host = alpaka::getDevByIdx<HostPlatform>(0u);

  std::cout << "Host platform: " << alpaka::core::demangled<HostPlatform> << '\n';
  std::cout << "Found 1 device:\n";
  std::cout << "  - " << alpaka::getName(host) << '\n';
  std::cout << std::endl;

  // enumerate the devices on the accelerator platform
  std::vector<Device> devices;
  std::size_t n = alpaka::getDevCount<Platform>();
  devices.reserve(n);
  for (std::size_t i = 0; i < n; ++i) {
    devices.push_back(alpaka::getDevByIdx<Platform>(i));
  }

  std::cout << "Accelerator platform: " << alpaka::core::demangled<Platform> << '\n';
  std::cout << "Found " << devices.size() << " device(s):\n";
  for (auto const& device: devices)
    std::cout << "  - " << alpaka::getName(device) << '\n';
  std::cout << std::endl;
}
```

- these are the *host* and *accelerator* platforms

your first alpaka application

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/00_enumerate.cc

```
int main() {
  // the host abstraction always has a single device
  Host host = alpaka::getDevByIdx<HostPlatform>(0u);

  std::cout << "Host platform: " << alpaka::core::demangled<HostPlatform> << '\n';
  std::cout << "Found 1 device:\n";
  std::cout << "  - " << alpaka::getName(host) << '\n';
  std::cout << std::endl;

  // enumerate the devices on the accelerator platform
  std::vector<Device> devices;
  std::::size_t n = alpaka::getDevCount<Platform>();
  devices.reserve(n);
  for (std::size_t i = 0; i < n; ++i) {
    devices.push_back(alpaka::getDevByIdx<Platform>(i));
  }

  std::cout << "Accelerator platform: " << alpaka::core::demangled<Platform> << '\n';
  std::cout << "Found " << devices.size() << " device(s):\n";
  for (auto const& device: devices)
    std::cout << "  - " << alpaka::getName(device) << '\n';
  std::cout << std::endl;
}
```

- `alpaka::core::demangled<T>` is a string with the "human readable" name of c++ type name

your first alpaka application

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/00_enumerate.cc

```
int main() {
  // the host abstraction always has a single device
  Host host = alpaka::getDevByIdx<HostPlatform>(0u);

  std::cout << "Host platform: " << alpaka::core::demangled<HostPlatform> << '\n';
  std::cout << "Found 1 device:\n";
  std::cout << "  - " << alpaka::getName(host) << '\n';
  std::cout << std::endl;

  // enumerate the devices on the accelerator platform
  std::vector<Device> devices;
  std::size_t n = alpaka::getDevCount<Platform>();
  devices.reserve(n);
  for (std::size_t i = 0; i < n; ++i) {
    devices.push_back(alpaka::getDevByIdx<Platform>(i));
  }

  std::cout << "Accelerator platform: " << alpaka::core::demangled<Platform> << '\n';
  std::cout << "Found " << devices.size() << " device(s):\n";
  for (auto const& device: devices)
    std::cout << "  - " << alpaka::getName(device) << '\n';
  std::cout << std::endl;
}
```

- query the number of devices on the platform

your first alpaka application

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/00_enumerate.cc

```
int main() {
  // the host abstraction always has a single device
  Host host = alpaka::getDevByIdx<HostPlatform>(0u);

  std::cout << "Host platform: " << alpaka::core::demangled<HostPlatform> << '\n';
  std::cout << "Found 1 device:\n";
  std::cout << "  - " << alpaka::getName(host) << '\n';
  std::cout << std::endl;

  // enumerate the devices on the accelerator platform
  std::vector<Device> devices;
  std::size_t n = alpaka::getDevCount<Platform>();
  devices.reserve(n);
  for (std::size_t i = 0; i < n; ++i) {
    devices.push_back(alpaka::getDevByIdx<Platform>(i));
  }

  std::cout << "Accelerator platform: " << alpaka::core::demangled<Platform> << '\n';
  std::cout << "Found " << devices.size() << " device(s):\n";
  for (auto const& device: devices)
    std::cout << "  - " << alpaka::getName(device) << '\n';
  std::cout << std::endl;
}
```

- get the n^{th} device for the given platform

your first alpaka application

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/00_enumerate.cc

```
int main() {
  // the host abstraction always has a single device
  Host host = alpaka::getDevByIdx<HostPlatform>(0u);

  std::cout << "Host platform: " << alpaka::core::demangled<HostPlatform> << '\n';
  std::cout << "Found 1 device:\n";
  std::cout << "  - " << alpaka::getName(host) << '\n';
  std::cout << std::endl;

  // enumerate the devices on the accelerator platform
  std::vector<Device> devices;
  std::size_t n = alpaka::getDevCount<Platform>();
  devices.reserve(n);
  for (std::size_t i = 0; i < n; ++i) {
    devices.push_back(alpaka::getDevByIdx<Platform>(i));
  }

  std::cout << "Accelerator platform: " << alpaka::core::demangled<Platform> << '\n';
  std::cout << "Found " << devices.size() << " device(s):\n";
  for (auto const& device: devices)
    std::cout << "  - " << alpaka::getName(device) << '\n';
  std::cout << std::endl;
}
```

- get the name of the device

some important details

```
/*
 * g++ -std=c++17 -O2 -g -DALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED -I$BOOST_BASE/include -I$ALPAKA_BASE/include 00_enumerate.cc -o
00_enumerate_cpu
 * nvcc -x cu -std=c++17 -O2 -g --expt-relaxed-constexpr -DALPAKA_ACC_GPU_CUDA_ENABLED -I$BOOST_BASE/include -I$ALPAKA_BASE/include
00_enumerate.cc -o 00_enumerate_cuda
 */

#include <iostream>
#include <vector>

#include <alpaka/alpaka.hpp>

#include "config.h"

...
```

- grab all the examples from GitHub

```
git clone https://github.com/fwyzard/intro_to_alpaka.git
```

let's build it …

- using the CPU as the "accelerator"
  - the CPU acts as both the "host" and the "device"
  - the application runs entirely on the CPU

```
g++ -DALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED \
    -std=c++17 -O2 -g -I$BOOST_BASE/include -I$ALPAKA_BASE/include \
    00_enumerate.cc \
    -o 00_enumerate_cpu
```

- using the CUDA GPUs as the "accelerator"
  - the CPU acts as the "host", the GPUs act as the "devices"
  - the application launches kernels that run on the GPUs

```
nvcc -x cu -expt-relaxed-constexpr -DALPAKA_ACC_GPU_CUDA_ENABLED \
    -std=c++17 -O2 -g -I$BOOST_BASE/include -I$ALPAKA_BASE/include \
    00_enumerate.cc \
    -o 00_enumerate_cuda
```

… and run it

```
$ ./00_enumerate_cpu
Host platform: alpaka::PltfCpu
Found 1 device:
  - AMD EPYC 7352 24-Core Processor

Accelerator platform: alpaka::PltfCpu
Found 1 device(s):
  - AMD EPYC 7352 24-Core Processor
```

```
$ ./00_enumerate_cuda
Host platform: alpaka::PltfCpu
Found 1 device:
  - AMD EPYC 7352 24-Core Processor

Accelerator platform: alpaka::PltfUniformCuda…
Found 2 device(s):
  - Tesla T4
  - Tesla T4
```

where is the magic ?

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/config.h

```
#if defined(ALPAKA_ACC_GPU_CUDA_ENABLED)
// CUDA backend
using Device = alpaka::DevCudaRt;
using Platform = alpaka::Pltf<Device>;

#elif defined(ALPAKA_ACC_GPU_HIP_ENABLED)
// HIP/ROCm backend
using Device = alpaka::DevHipRt;
using Platform = alpaka::Pltf<Device>;

#elif defined(ALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED)
// CPU serial backend
using Device = alpaka::DevCpu;
using Platform = alpaka::Pltf<Device>;

#else
// no backend specified
#error Please define one of ALPAKA_ACC_GPU_CUDA_ENABLED, ALPAKA_ACC_GPU_HIP_ENABLED, ALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED

#endif
```

back end	alpaka platform	alpaka device
CPUs, serial or parallel	PltfCpu	DevCpu
NVIDIA GPU, with CUDA	PltfCudaRt	DevCudaRt
AMD GPUs, with HIP/ROCm	PltfHipRt	DevHipRt

where is the magic ?

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/config.h

```
#if defined(ALPAKA_ACC_GPU_CUDA_ENABLED)
// CUDA backend
using Device = alpaka::DevCudaRt;
using Platform = alpaka::Pltf<Device>;

#elif defined(ALPAKA_ACC_GPU_HIP_ENABLED)
// HIP/ROCm backend
using Device = alpaka::DevHipRt;
using Platform = alpaka::Pltf<Device>;

#elif defined(ALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED)
// CPU serial backend
using Device = alpaka::DevCpu;
using Platform = alpaka::Pltf<Device>;

#else
// no backend specified
#error Please define one of ALPAKA_ACC_GPU_CUDA_ENABLED, ALPAKA_ACC_GPU_HIP_ENABLED, ALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED

#endif
```

- depending on which back-end is enabled ...

where is the magic ?

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/config.h

```
#if defined(ALPAKA_ACC_GPU_CUDA_ENABLED)
// CUDA backend
using Device = alpaka::DevCudaRt;
using Platform = alpaka::Pltf<Device>;

#elif defined(ALPAKA_ACC_GPU_HIP_ENABLED)
// HIP/ROCm backend
using Device = alpaka::DevHipRt;
using Platform = alpaka::Pltf<Device>;

#elif defined(ALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED)
// CPU serial backend
using Device = alpaka::DevCpu;
using Platform = alpaka::Pltf<Device>;

#else
// no backend specified
#error Please define one of ALPAKA_ACC_GPU_CUDA_ENABLED, ALPAKA_ACC_GPU_HIP_ENABLED, ALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED

#endif
```

depending on which back-end is enabled, `Device` and `Platform` are aliased to different types

queues and events

alpaka: asynchronous operations

Queues:

- identify a “work queue” where tasks (memory operations, kernel executions, …) are executed in order
  - for example, a queue could represent an underlying CUDA stream or a CPU thread
  - from the point of view of the host , queues can be synchronous or asynchronous
- with a synchronous (or *blocking*) queue:
  - any operation is executed immediately, before returning to the caller
  - the host automatically waits (blocks) until each operation is complete
- with an asynchronous (or *non-blocking*) queue:
  - any operation is executed in the background, and each call returns immediately, without waiting for its completion
  - the host needs to synchronize explicitly with the queue, before accessing the results of the operations
- in general, prefer using a synchronous queue on a CPU, and an asynchronous queue on a GPU
- queues are always associated to a specific device
- most Alpaka operations (memory ops, kernel launches, etc.) are associated to a queue
- Alpaka does not provide a “default queue”, create one explicitly

common operations on queues

- creating a queue of the predefined type associated to a device is as simple as

```
auto queue = Queue(device);
```

- waiting for all the asynchronous operations in a queue to complete is as simple as

```
alpaka::wait(queue);
```

- enqueue a host function

```
alpaka::enqueue(queue, host_function);
```

- enqueue a device function (launch a kernel)

```
…
```

- allocate, set, or copy memory host and device memory

```
…
```

alpaka: events and synchronisation

Events:

- events identify points in time along a work queue
- can be used to query or wait for the readiness of a task submitted to a queue
- can be used to synchronise different queues
- like queues, events are always associated to a specific device

common operations on events

- events associated to a given device can be created with:

```
auto event = Event(device);
```

- events are enqueued to mark a given point along the queue:

```
alpaka::enqueue(queue, event);
```

  - an event is “complete” once all the work submitted to the queue before the event has been completed

- an event can be used to block the execution on the host until it is complete:

```
alpaka::wait(event);
```

  - blocks the execution on the host

- or to make an other queue wait until a given event (in a different queue) is complete:

```
alpaka::wait(other_queue, event);
```

  - does not block execution on the host
  - further work submitted to `other_queue` will only start after `event` is complete

- an event’s status can also be queried without blocking the execution:

```
alpaka::isComplete(event);
```

more magic

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/config.h

```
#if defined(ALPAKA_ACC_GPU_CUDA_ENABLED)
// CUDA backend
using Queue = alpaka::Queue<Device, alpaka::NonBlocking>;
using Event = alpaka::Event<Queue>;

#elif defined(ALPAKA_ACC_GPU_HIP_ENABLED)
// HIP/ROCm backend
using Queue = alpaka::Queue<Device, alpaka::NonBlocking>;
using Event = alpaka::Event<Queue>;

#elif defined(ALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED)
// CPU serial backend
using Queue = alpaka::Queue<Device, alpaka::Blocking>;
using Event = alpaka::Event<Queue>;

#else
// no backend specified
#error Please define one of ALPAKA_ACC_GPU_CUDA_ENABLED, ALPAKA_ACC_GPU_HIP_ENABLED, ALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED

#endif
```

more magic

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/config.h

```
#if defined(ALPAKA_ACC_GPU_CUDA_ENABLED)
// CUDA backend
using Queue = alpaka::Queue<Device, alpaka::NonBlocking>;
using Event = alpaka::Event<Queue>;

#elif defined(ALPAKA_ACC_GPU_HIP_ENABLED)
// HIP/ROCm backend
using Queue = alpaka::Queue<Device, alpaka::NonBlocking>;
using Event = alpaka::Event<Queue>;

#elif defined(ALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED)
// CPU serial backend
using Queue = alpaka::Queue<Device, alpaka::Blocking>;
using Event = alpaka::Event<Queue>;

#else
// no backend specified
#error Please define one of ALPAKA_ACC_GPU_CUDA_ENABLED, ALPAKA_ACC_GPU_HIP_ENABLED, ALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED

#endif
```

- prefer asynchronous queues for a GPU
- prefer synchronous queues for a CPU

fun with queues

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/01_blocking_queue.cc

```cpp
int main() {
  // the host abstraction always has a single device
  Host host = alpaka::getDevByIdx<HostPlatform>(0u);

  std::cout << "Host platform: " << alpaka::core::demangled<HostPlatform> << '\n';
  std::cout << "Found 1 device:\n";
  std::cout << "  - " << alpaka::getName(host) << '\n';
  std::cout << std::endl;

  // create a blocking host queue and submit some work to it
  alpaka::Queue<Host, alpaka::Blocking> queue{host};

  std::cout << "Enqueue some work\n";
  alpaka::enqueue(queue, []() noexcept {
      std::cout << "  - host task running...\n";
      std::this_thread::sleep_for(std::chrono::seconds(5u));
      std::cout << "  - host task complete\n";
  });

  // wait for the work to complete
  std::cout << "Wait for the enqueue work to complete...\n";
  alpaka::wait(queue);
  std::cout << "All work has completed\n";
}
```

fun with queues

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/01_blocking_queue.cc

```
int main() {
  // the host abstraction always has a single device
  Host host = alpaka::getDevByIdx<HostPlatform>(0u);

  std::cout << "Host platform: " << alpaka::core::demangled<HostPlatform> << '\n';
  std::cout << "Found 1 device:\n";
  std::cout << "  - " << alpaka::getName(host) << '\n';
  std::cout << std::endl;

  // create a blocking host queue and submit some work to it
  alpaka::Queue<Host, alpaka::Blocking> queue{host};

  std::cout << "Enqueue some work\n";
  alpaka::enqueue(queue, []() noexcept {
      std::cout << "  - host task running...\n";
      std::this_thread::sleep_for(std::chrono::seconds(5u));
      std::cout << "  - host task complete\n";
  });

  // wait for the work to complete
  std::cout << "Wait for the enqueue work to complete...\n";
  alpaka::wait(queue);
  std::cout << "All work has completed\n";
}
```

this part we know

fun with queues

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/01_blocking_queue.cc

```
int main() {
  // the host abstraction always has a single device
  Host host = alpaka::getDevByIdx<HostPlatform>(0u);

  std::cout << "Host platform: " << alpaka::core::demangled<HostPlatform> << '\n';
  std::cout << "Found 1 device:\n";
  std::cout << "  - " << alpaka::getName(host) << '\n';
  std::cout << std::endl;

  // create a blocking host queue and submit some work to it
  alpaka::Queue<Host, alpaka::Blocking> queue{host};

  std::cout << "Enqueue some work\n";
  alpaka::enqueue(queue, []() noexcept {
      std::cout << "  - host task running...\n";
      std::this_thread::sleep_for(std::chrono::seconds(5u));
      std::cout << "  - host task complete\n";
  });

  // wait for the work to complete
  std::cout << "Wait for the enqueue work to complete...\n";
  alpaka::wait(queue);
  std::cout << "All work has completed\n";
}
```

- create a *blocking* queue on the Host

fun with queues

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/01_blocking_queue.cc

```
int main() {
  // the host abstraction always has a single device
  Host host = alpaka::getDevByIdx<HostPlatform>(0u);

  std::cout << "Host platform: " << alpaka::core::demangled<HostPlatform> << '\n';
  std::cout << "Found 1 device:\n";
  std::cout << "  - " << alpaka::getName(host) << '\n';
  std::cout << std::endl;

  // create a blocking host queue and submit some work to it
  alpaka::Queue<Host, alpaka::Blocking> queue{host};

  std::cout << "Enqueue some work\n";
  alpaka::enqueue(queue, []() noexcept {
      std::cout << "  - host task running...\n";
      std::this_thread::sleep_for(std::chrono::seconds(5u));
      std::cout << "  - host task complete\n";
  });

  // wait for the work to complete
  std::cout << "Wait for the enqueue work to complete...\n";
  alpaka::wait(queue);
  std::cout << "All work has completed\n";
}
```

- this syntax introduces a *lambda expression* …

fun with queues

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/01_blocking_queue.cc

```
int main() {
  // the host abstraction always has a single device
  Host host = alpaka::getDevByIdx<HostPlatform>(0u);

  std::cout << "Host platform: " << alpaka::core::demangled<HostPlatform> << '\n';
  std::cout << "Found 1 device:\n";
  std::cout << "  - " << alpaka::getName(host) << '\n';
  std::cout << std::endl;

  // create a blocking host queue and submit some work to it
  alpaka::Queue<Host, alpaka::Blocking> queue{host};

  std::cout << "Enqueue some work\n";
  alpaka::enqueue(queue, []() noexcept {
      std::cout << "  - host task running...\n";
      std::this_thread::sleep_for(std::chrono::seconds(5u));
      std::cout << "  - host task complete\n";
  });

  // wait for the work to complete
  std::cout << "Wait for the enqueue work to complete...\n";
  alpaka::wait(queue);
  std::cout << "All work has completed\n";
}
```

this syntax introduces a *lambda expression*
that performs these operations

togethwer with `alpaka::enqueue(...)`, this part
- creates an object that encapsulates some operations
- submits those opertations to run in a queue

fun with queues

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/01_blocking_queue.cc

```
int main() {
  // the host abstraction always has a single device
  Host host = alpaka::getDevByIdx<HostPlatform>(0u);

  std::cout << "Host platform: " << alpaka::core::demangled<HostPlatform> << '\n';
  std::cout << "Found 1 device:\n";
  std::cout << "  - " << alpaka::getName(host) << '\n';
  std::cout << std::endl;

  // create a blocking host queue and submit some work to it
  alpaka::Queue<Host, alpaka::Blocking> queue{host};

  std::cout << "Enqueue some work\n";
  alpaka::enqueue(queue, []() noexcept {
      std::cout << "  - host task running...\n";
      std::this_thread::sleep_for(std::chrono::seconds(5u));
      std::cout << "  - host task complete\n";
  });

  // wait for the work to complete
  std::cout << "Wait for the enqueue work to complete...\n";
  alpaka::wait(queue);
  std::cout << "All work has completed\n";
}
```

- wait for the enqueued operations to complete

let's build it and run it

- in this example we are not making use of any accelerator
  - let's build it only for the CPU back-end

```
g++ -DALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED \
    -std=c++17 -O2 -g -I$BOOST_BASE/include -I$ALPAKA_BASE/include \
    01_blocking_queue.cc \
    -o 01_blocking_queue_cpu
```

  - and run it

```
$ ./01_blocking_queue_cpu
Host platform: alpaka::PltfCpu
Found 1 device:
  - AMD EPYC 7352 24-Core Processor

Enqueue some work
  - host task running...
  - host task complete
Wait for the enqueue work to complete...
All work has completed
```

an async example

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/02_nonblocking_queue.cc

```
int main() {
  // the host abstraction always has a single device
  Host host = alpaka::getDevByIdx<HostPlatform>(0u);

  std::cout << "Host platform: " << alpaka::core::demangled<HostPlatform> << '\n';
  std::cout << "Found 1 device:\n";
  std::cout << "  - " << alpaka::getName(host) << '\n';
  std::cout << std::endl;

  // create a non-blocking host queue and submit some work to it
  alpaka::Queue<Host, alpaka::NonBlocking> queue{host};

  std::cout << "Enqueue some work\n";
  alpaka::enqueue(queue, []() noexcept {
      std::cout << "  - host task running...\n";
      std::this_thread::sleep_for(std::chrono::seconds(5u));
      std::cout << "  - host task complete\n";
  });

  // wait for the work to complete
  std::cout << "Wait for the enqueue work to complete...\n";
  alpaka::wait(queue);
  std::cout << "All work has completed\n";
}
```

an async example

https://github.com/fwyzard/intro_to_alpaka/blob/master/alpaka/02_nonblocking_queue.cc

```
int main() {
  // the host abstraction always has a single device
  Host host = alpaka::getDevByIdx<HostPlatform>(0u);

  std::cout << "Host platform: " << alpaka::core::demangled<HostPlatform> << '\n';
  std::cout << "Found 1 device:\n";
  std::cout << "  - " << alpaka::getName(host) << '\n';
  std::cout << std::endl;

  // create a non-blocking host queue and submit some work to it
  alpaka::Queue<Host, alpaka::NonBlocking> queue{host};

  std::cout << "Enqueue some work\n";
  alpaka::enqueue(queue, []() noexcept {
      std::cout << "  - host task running...\n";
      std::this_thread::sleep_for(std::chrono::seconds(5u));
      std::cout << "  - host task complete\n";
  });

  // wait for the work to complete
  std::cout << "Wait for the enqueue work to complete...\n";
  alpaka::wait(queue);
  std::cout << "All work has completed\n";
}
```

- create a *non-blocking* queue on the Host

let's build it and run it

- in this example, too, we are not making use of any accelerator
  - let's build it only for the CPU back-end – with POSIX threads

```
g++ -DALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED \
    -std=c++17 -O2 -g -I$BOOST_BASE/include -I$ALPAKA_BASE/include -pthread \
    02_nonblocking_queue.cc \
    -o 02_nonblocking_queue_cpu
```

  - and run it

```
$ ./02_nonblocking_queue_cpu
Host platform: alpaka::PltfCpu
Found 1 device:
  - AMD EPYC 7352 24-Core Processor

Enqueue some work
Wait for the enqueue work to complete...
  - host task running...
  - host task complete
All work has completed
```

blocking vs non-blocking

```
$ ./01_blocking_queue_cpu
Host platform: alpaka::PltfCpu
Found 1 device:
  - AMD EPYC 7352 24-Core Processor

Enqueue some work
  - host task running...
  - host task complete
Wait for the enqueue work to complete...
All work has completed
```

```
$ ./02_nonblocking_queue_cpu
Host platform: alpaka::PltfCpu
Found 1 device:
  - AMD EPYC 7352 24-Core Processor

Enqueue some work
Wait for the enqueue work to complete...
  - host task running...
  - host task complete
All work has completed
```

blocking vs non-blocking

```
$ ./01_blocking_queue_cpu
Host platform: alpaka::PltfCpu
Found 1 device:
  - AMD EPYC 7352 24-Core Processor

Enqueue some work
  - host task running...
  - host task complete
Wait for the enqueue work to complete...
All work has completed
```

```
$ ./02_nonblocking_queue_cpu
Host platform: alpaka::PltfCpu
Found 1 device:
  - AMD EPYC 7352 24-Core Processor

Enqueue some work
Wait for the enqueue work to complete...
  - host task running...
  - host task complete
All work has completed
```

- with a synchronous (or *blocking*) queue:
  - any operation is executed immediately, before returning to the caller
  - the host automatically waits (blocks) until each operation is complete
- with an asynchronous (or *non-blocking*) queue:
  - any operation is executed in the background, and each call returns immediately, without waiting for its completion
  - the host needs to synchronize explicitly with the queue, before accessing the results of the operations

what’s next ?

summary

- today we have learned
  - what *performance portability* means and discovered the Alpaka library
  - how to set up Alpaka for a simple project
  - how to compile a single source file for different back-ends
  - what are Alpaka platforms, devices, queues and events

- tomorrow we will see
  - how to work with host and device memory
  - how to write device functions and kernels
  - how to use an Alpaka accelerator and work division to launch a kernel
  - a complete example !

(more) questions ?