Lesson 32: The Accelerator Concept

Introduction

- alpaka’s Accelerator concept is an important tool
- Accelerator hides hardware specifics behind alpaka’s abstract API
- Chosen by programmer:
  ```cpp
  using Acc = acc::AccGpuCudaRt<Dim, Idx>;
  ```
- Used on both Host and Device
- Inside Kernel: contains thread state, provides access to alpaka’s device-side API
- On Host: Meta-parameter for choosing correct physical device and dependent types
Lesson 32: The Accelerator Concept

Accelerators and devices

- Accelerator concept is an abstraction of concrete devices and programming models
- The programmer changes the accelerator in just one line of code
- In the background, an entirely different code path for the “new” device is chosen
- Accelerator provides portable access to device-specific functions

/* Before the code change */
using Acc = acc::AccCpuOmp2Blocks<Dim, Idx>;
/* Kernels will run on CPUs */
/* Parallelism provided by OpenMP 2.x */

/* After the code change */
using Acc = acc::AccGpuHipRt<Dim, Idx>;
/* Kernels will run on AMD + NVIDIA GPUs */
/* Parallelism provided by HIP */
Lesson 32: The Accelerator Concept

Grid navigation

• The Accelerator provides the means to navigate the grid:

```cpp
// get thread index on the grid
auto gridThreadId = idx::getIdx<Grid, Threads>(acc);

// get block index on the grid
auto gridBlockId = idx::getIdx<Grid, Blocks>(acc);

// get thread index on the block
auto blockThreadId = idx::getIdx<Block, Threads>(acc);

// get number of blocks on the grid
auto gridBlockExtent = workdiv::getWorkDiv<Grid, Blocks>(acc);

// get number of threads on the block
auto blockThreadExtent = workdiv::getWorkDiv<Block, Threads>(acc);
```
Lesson 32: The Accelerator Concept

Memory management and synchronization

- The Accelerator gives access to alpaka's shared memory (for threads inside the same block):

  ```
  // allocate a variable in block shared static memory
  auto & mySharedVar = block::shared::st::allocVar<int, __COUNTER__>(acc);

  // get pointer to the block shared dynamic memory
  float * mySharedBuffer = block::shared::dyn::getMem<float>(acc);
  ```

- It also enables synchronization on the block level:

  ```
  // synchronize all threads within the block
  block::sync::syncBlockThreads(acc);

  // synchronize some threads within the block and evaluate predicate
  block::sync::syncBlockThreadsPredicate(acc, predicate);
  ```
Lesson 32: The Accelerator Concept

Device-side functions

- Internally, the accelerator maps all device-side functions to their native counterparts
- Device-side functions require the accelerator as first argument:
  - `math::sqrt(acc, /* ... */);` (Math functions)
  - `atomic::atomicOp<atomic::op::Or>(acc, /* ... */, hierarchy::Grids);` (Atomics)
  - `rand::distribution::createNormalReal<float>(acc);` (Random-number generation)
  - `time::clock(acc);` (Clock cycles)