Gaining support for multiple device by migrating a CUDA code to oneAPI

Experiences with the Intel® DPC++ Compatibility Tool

Steffen Christgau, Marius Knaust

Supercomputing Department
Zuse Institute Berlin

oneAPI Developer Summit
November 12, 2020

ZUSE INSTITUTE BERLIN

MODAL FORSCHUNGSCAMPUS

HPC Lab

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

oneAPI and legacy CUDA code

- **oneAPI** – "common developer experience across accelerator architectures"
  - Data Parallel C++ as programming language of choice
  - new landmark in *parallel heterogeneous programming model landscape*
  - **expectation:** single code for different platforms, like CPUs and accelerators

- **challenge:** Existing codes targeting single hardware platforms, like CUDA

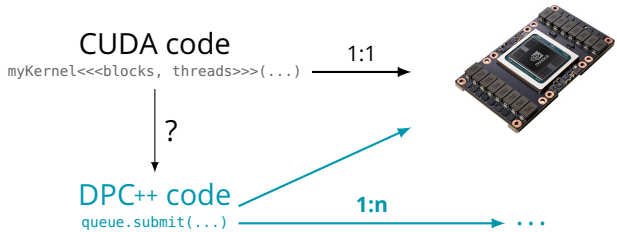
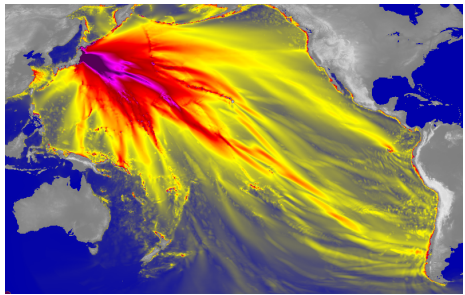


image source: https://www.nvidia.com/en-us/data-center/v100/

Case Study: easyWave

- German Research Center for GeoSciences (GFZ) and University of Potsdam
- open source tsunami simulation: arrival times and wave heights
- originally written in C++: 4470 LoC
  - OpenMP and CUDA support
  - classes for programming model abstraction
- memory bound **stencil kernels** on dynamically **growing compute domain**



Bringing CUDA code to oneAPI

- **convenient migration** assisted by **Compatibility Tool** (*dpct*)
  - input: CUDA-flavoured project code
  - output: automatically migrated code $\rightarrow$ **still readable + maintainable**
  - only CUDA-related parts touched
  - no manual and tedious boilerplate/syntax changes
  - migrated code good starting point for further development + optimizations

- result for easyWave: LOC **increase by 5%** (4470 vs 4674)

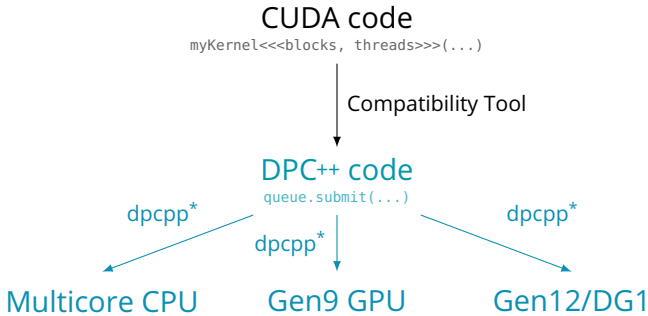
- **lots of improvements** in *dpct* over evolution of oneAPI beta releases based on feedback of users (including ourselves)

Migration Assistance

easyWave code migrations performed by the Compatibility Tool:

- kernel launch and refactoring
  - CUDA: `waveUpdateKernel<<<blocks, threads>>>(...)`
  - SYCL: `queue.submit([&](sycl::handler cgh) {cgh.parallel_for(...)})`
- kernel execution → different ways to specify work distribution (SYCL vs. CUDA)
- error checking: C++ exceptions vs. CPP macros
- memory allocation
  - CUDA explicit `malloc`/`free` for 1D and 2D data and transfers (`cudaMemcpy2D`)
  - DPC++: **Unified Shared Memory** `sycl::malloc_device/free` → **Pointers!**
- idiom conversions:
  - timing of kernel execution, usage of modern C++ library features
  - atomic operations

Towards Multiple Architectures



- Same DPC++ code can target different devices!
- valid computational results

* from Intel oneAPI Beta releases

Reusing CUDA hardware

… by using the migrated DPC++ code!

- almost **no adjustments** required

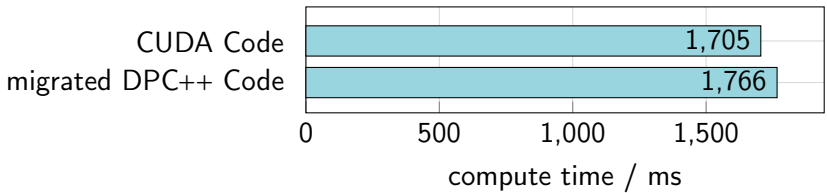
- use open source DPC++ compiler w/ CUDA support (contributed by Codeplay)
- scripts exists to build compiler

- compile application code with some special options, like:
  - `-fsycl` – enable SYCL support
  - `-fsycl-unnamed-lambda` – required for migrated code
  - `-fsycl-targets=nvptx64-nvidia-cuda-sycldevice` – target Nvidia GPUs

- set `SYCL_BE` environment variable to `PI_CUDA` for execution… **Done**

Reusing CUDA hardware

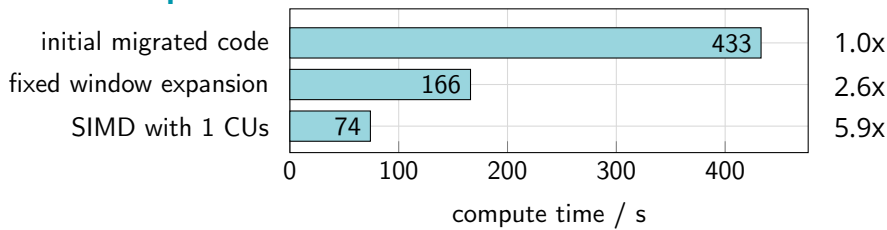
- What about **performance**?
- typical application run on Nvidia P100-SXM2-16GB
- **almost same runtime** compared to CUDA code (only 4% slower)



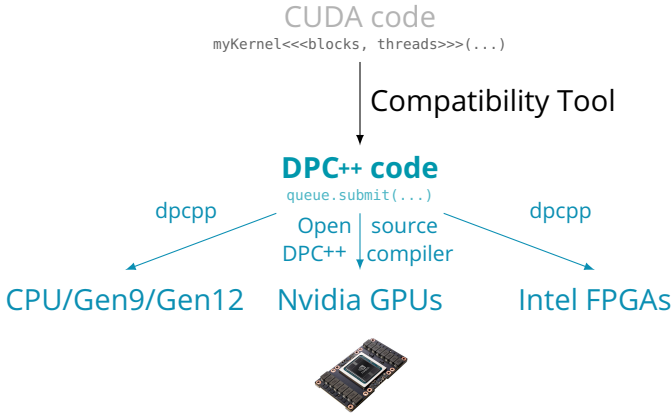
Compute Domain: approx. 2000 x 1400 cells; 10 hours simulation time, timestep = 5 minutes

Running on FPGAs

- use migrated code again
- test: with FPGA emulation → produced correct results
- build: use `dpcpp` compiler → no VHDL/Verilog, no manual synthesis
  - compile: `-fintelfpga`
  - link: `-fintelfpga -Xshardware -Xsboard=intel_s10sx_pac:pac_s10`
- run on actual Intel PAC / Stratix 10 SX: just run binary
  - produced correct values
  - small issue with atomics
- What about **performance**?



Summary



Thanks for your attention!

{christgau,knaust}@zib.de