From CUDA to DPC++ and back to Nvidia GPUs... and FPGAs

A oneAPI case study with the tsunami simulation easyWave

Steffen Christgau, Marius Knaust

Supercomputing Department
Zuse Institute Berlin

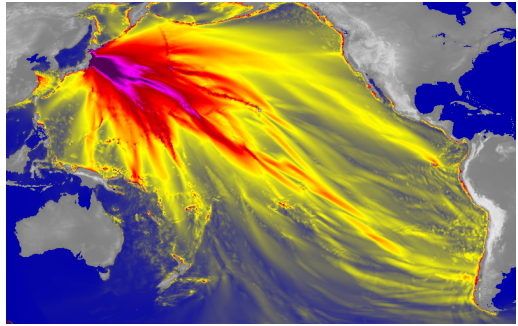
IXPUG Annual Conference
October 13, 2020

Where to start with oneAPI?

- new exciting programming environment for different hardware architectures
  - for data parallel applications $\rightarrow$ Data Parallel C++ (DPC++)
  - expectation: have single code for different platforms (CPUs, GPUs, etc.)
- What to do with existing applications, e.g. written (solely) for CUDA?
- Use **Compatibility Tool** to **migrate a real code** (not only vector addition)
- this talk: **easyWave**

easyWave

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



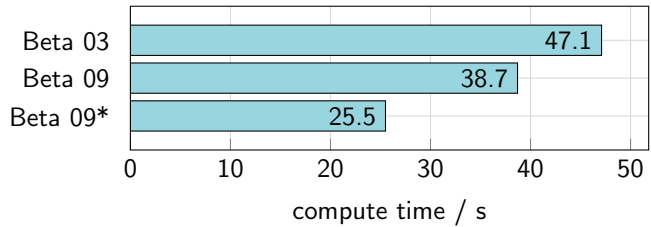
Heading towards oneAPI

- straightforward migration process, assisted by **Compatibility Tool (dpct)**
  - whole project or selected files as input to *dpct*
  - output is automatically migrated code $\rightarrow$ **still readable + maintainable**
  - only CUDA-related parts touched
  - comments added to mark (un)migrated code/migration issues

- **convienient automatic migration**
  - good starting point for further development
  - removes the burden of tedious boilerplate/syntax changes

Migration Result

- source code
  - LOC **increase by 5%** (4470 vs 4674) due to migrated SYCL kernel launch code
  - **kernels almost unmodified** by Compatibility Tool
- **Same code** produces **valid data** on CPU, Intel GPUs, and FPGA.
- oneAPI performance evolution on DevCloud Coffee Lake Gen9.5 GT2 iGPU:



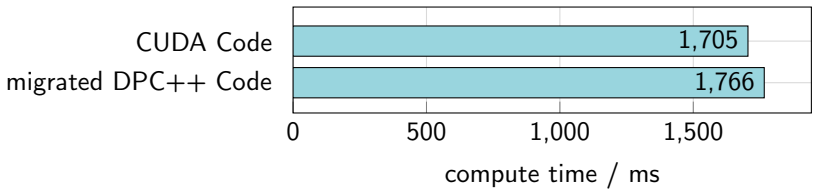
Compute Domain: approx. 2000 x 1400 cells; 10 hours simulation time
* self-made performance issue removed

Going back to Nvidia GPUs…

… using the migrated DPC++ code!

- almost **no adjustments** required, except workgroup size
- build with open source Intel LLVM w/ CUDA support (contribution by Codeplay)

- What about **performance**? Typical application run on Nvidia P100-SXM2-16GB:



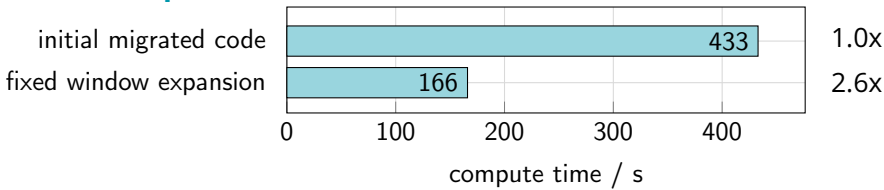
only 4% slower!

Compute Domain: approx. 2000 x 1400 cells; 10 hours simulation time

… and further to FPGAs

- use migrated code again
- build for FPGA using `dpcpp` compiler
- run on actual Intel PAC / Stratix 10 SX
  - produced correct values
  - but computed domain too large → atomics not working as expected
  - reduction-like code adjusted to use loop instead of atomics → correct results

- What about **performance**?



Summary

- Assisted migration of real world CUDA code to DPC++ feasible.
- Same DPC++ code can target different platforms (almost) without modifications.
- Performance is on par with architecture-specific CUDA code.

Thanks for your attention!
Questions?