



Explorations in heterogeneous computing

Summer Student Lightning Talks

Davide Marcato

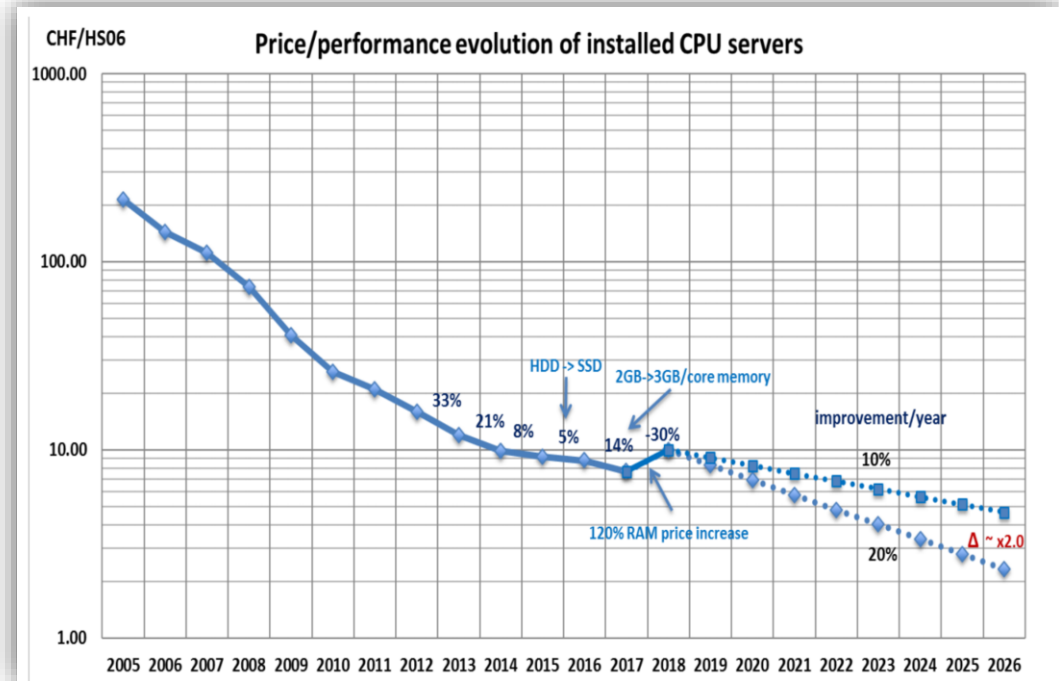
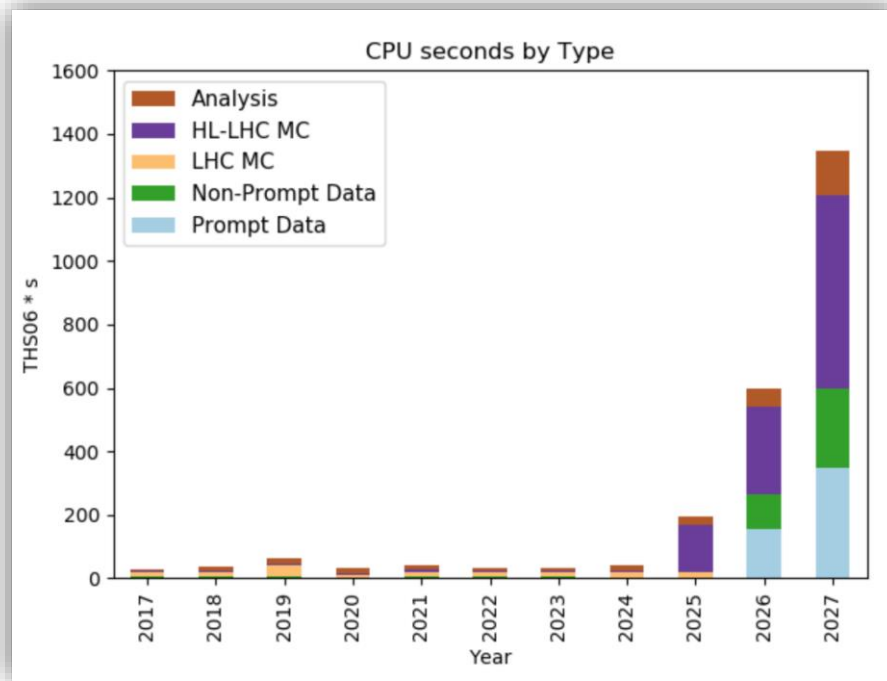
Supervisor:

Felice Pantaleo

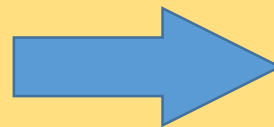
15/08/2019

LHC experiments future challenges

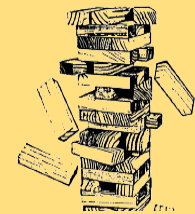
Computing requirements will grow exponentially



In 2027 4 million of today CPU cores would be needed

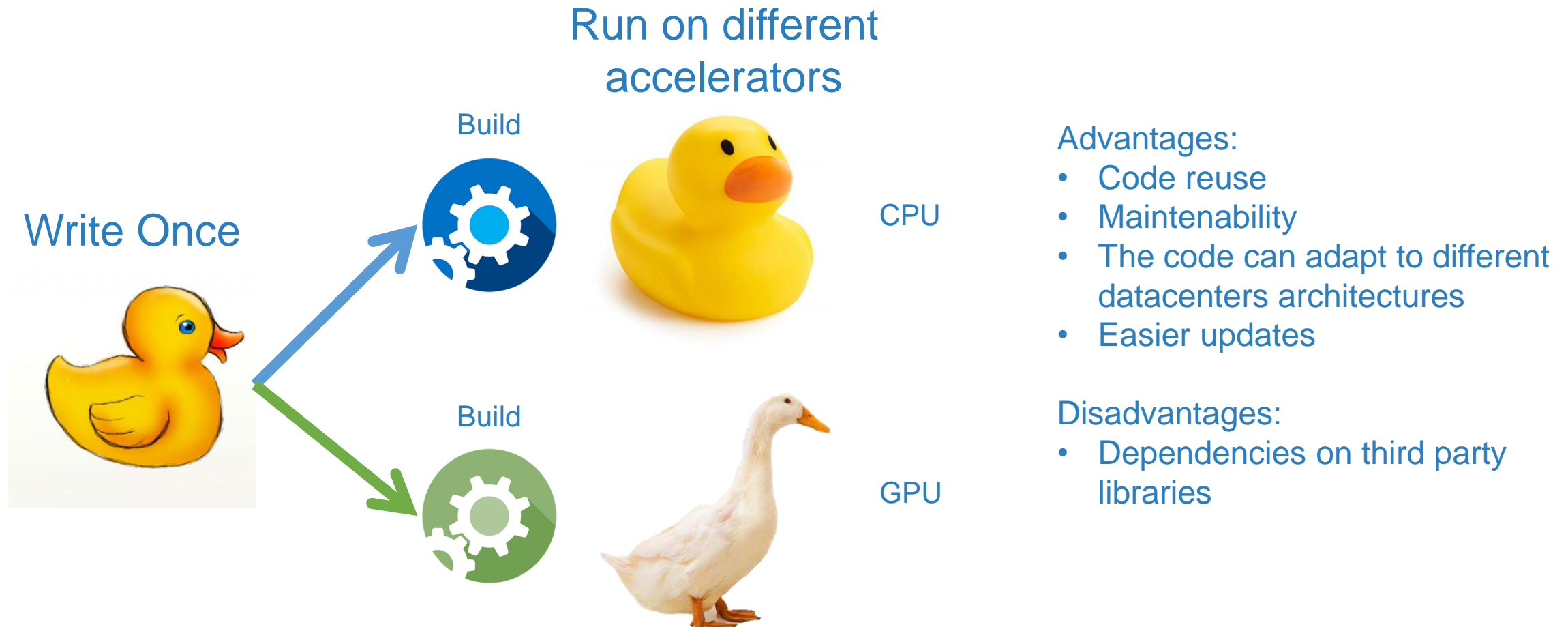


Scale out to GPUs
Patatrack Group for CMS



CUDA to Cupla

Evaluating the use of Alpaka + Cupla to write accelerator-independent code



CUDA to Cupla

Evaluating the use of Alpaka + Cupla to write accelerator-independent code



Successfully ported a first example with Eigen as a dependency



Verified the limits of the approach when porting large codebases (CMSSW)



Website with guide and problems to be solved

<http://patatrack.web.cern.ch/patatrack/wiki/cuda2cupla/>

The screenshot shows a web browser displaying the Patatrack Wiki page. The page title is "Transition from CUDA to cupla". The main content area contains the following text:

This is a report on the main steps to convert existing CUDA code to cupla and the main difficulties of the transition. Open problems will be highlighted.

The final goal is to obtain single source code which can be built both for CUDA and for CPU by simply configuring the building process, without modifying the source code. Cupla has been chosen for this task as it defines high level functions, as similar as possible to the CUDA calls, which work on both devices, easing the transition.

The theory

1. Remove all the CUDA headers and replace them with:

```
/* Do NOT include other headers that use CUDA runtime functions or variables
 * (see above) before this include.
 * The reason for this is that cupla renames CUDA host functions and device build
 * variables by using macros and macro functions.
 * Do NOT include other specific includes such as <cuda.h> (driver functions,
 * etc.).
 */
#include <cuda_to_cupla.hpp>
```

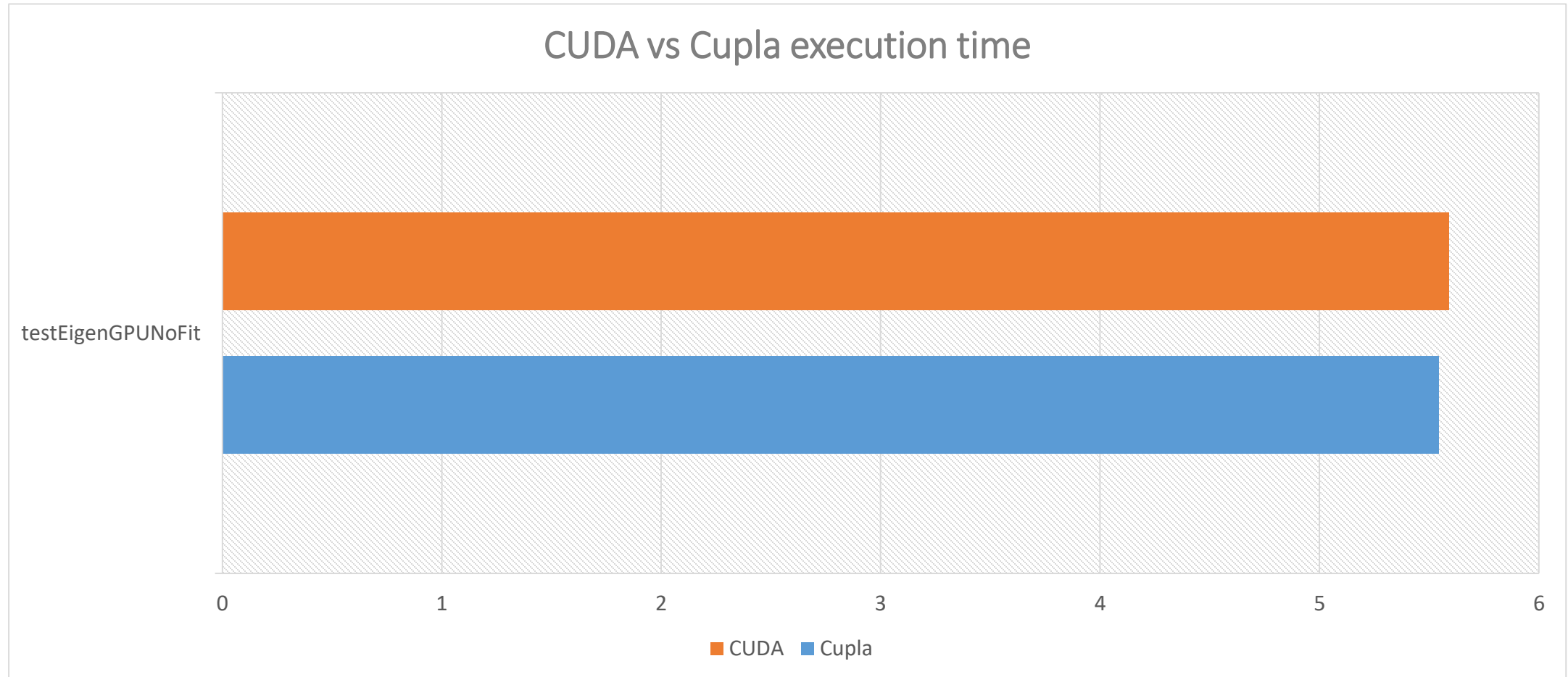
1. Transform the CUDA kernels to cupla functors. This is done by transforming the `__global__` functions to structures with a templated `const operator()`, which must have the `ALPAKA_FN_ACC` prefix and a first templated argument called `acc`. This is used by cupla to pass all the accelerator specific information.

The right sidebar contains a "Table of contents" with the following items:

- The theory
- A first practical example
- Build configuration
- Makefile
- Headers
- CUDA code porting
- Try it yourself
- Example takeaway
- Porting production code from CMSSW
- Main problems encountered

Performance

Performance are not affected by Cupla abstraction



Scaling on different GPUs

Automatic optimization of CUDA kernels launch parameters at runtime

- More and more datacenters offer accelerators
- The code needs to scale as much as possible on different (even future) GPU architectures
- Using `cudaOccupancyMaxPotentialBlockSize` to automatically choose at runtime the best parameters to launch each CUDA kernel

Performance improvements on GTX 1080Ti

Manual

920,6 ± 3,8
ev/s



Automatic

931,1 ± 5,1
ev/s



Winner of the 4th Patatrack Table Football Tournament

Davide Marcato
Adrian Alan Pol

6th Patatrack Hackathon
Geneva, 05/07/2019

Dr. Felice Pantaleo



QUESTIONS?

davide.marcato@cern.ch

marcato@infn.it