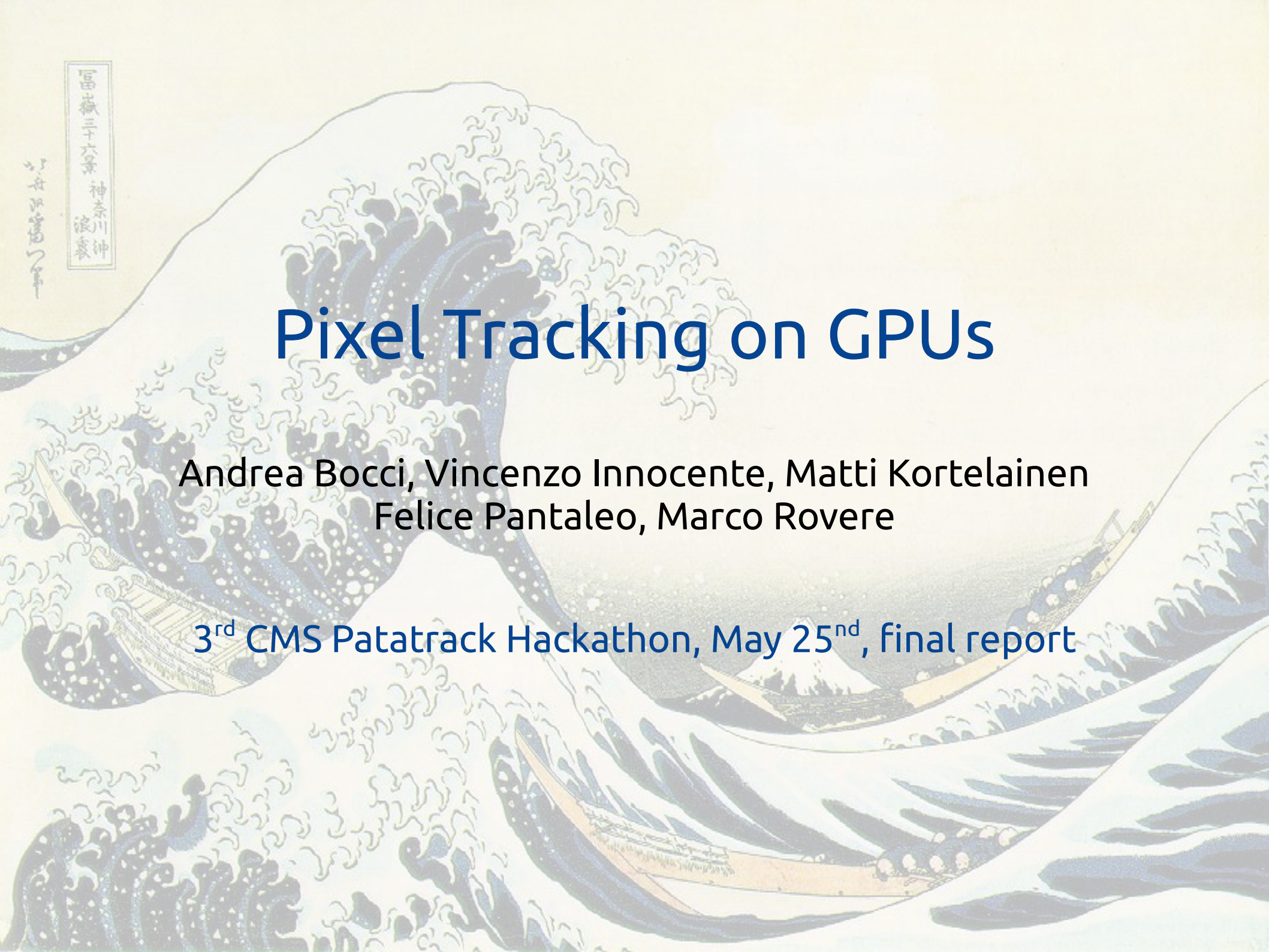

Pixel Tracking on GPUs

Andrea Bocci, Vincenzo Innocente, Matti Kortelainen
Felice Pantaleo, Marco Rovere

3rd CMS Patatrack Hackathon, May 25nd, final report

Heterogenous framework

- add Heterogeneous support to CMSSW
  - heterogeneous producer, based on the acquire / produce semantics
  - heterogeneous data products
- goals are efficiency and (relative) simplicity
  - aim to minimise data transfers between the host and the device(s)
    - track data locality
  - aim to allow the code to run on the best available device
    - currently CUDA and host CPU are supported
- some of the pixel tracking algorithms have been ported
  - “raw to cluster” algorithm
  - Cellular Automaton algorithm

Tracking on GPU

- port the “raw to cluster” to be a Heterogeneous producer
  - side product: “raw to cluster” approach implemented for the CPU
    - check if we save any processing time in the production worklow !
  - PR made

- port the Cellular Automaton to be a Heterogeneous producer
  - reimplement the algorithm for both CPU and CUDA
    - to be tested with the latest upstream changes and compiler
  - PR made

- port the Riemann Fit to CUDA
  - use one thread per fit / track
  - debugging work is ongoing, code is not 100% stable

make Eigen more CUDA friendly

- mark the code used by the Riemann Fit as `__host__ __device__`
  - generates many warnings
    - hopefully spurious, due to the templated code
  - attempt to silence the warning via `#pragmas`
    - leads to wrong code generation
  - attempt to use `constexpr` instead of `__host__ __device__`
    - not feasible (some functions are not really constexpr-compliant)

- next steps
  - check if more `__host__ __device__` functions are necessary
  - integrate the changes as a CMSSW external
  - submit the changes upstream to Eigen

compiling CUDA code with clang

- recent clang releases can compile CUDA code
  - compile from .cc / .cu to ptx
  - call `ptxas` to assemble it to CUDA architecture-specific object code
  - call `fatbinary` to merge these architecture-specific files
  - link the resulting object file with the host binary
- the missing feature is the possibility of splitting the device code in multiple .cu files, compile them separately, and link the device objects together
  - see Separate Compilation and Linking of CUDA C++ Device Code
- the latest development version of clang (7.0.0 from SVN) adds the possibility of compiling the individual .cu files to relocatable ptx and object code
- we can copy how `nvcc` links them into a single device binary:
  - call `nvlink` to link the relocatable object files into a single, architecture-specific object file
  - call `fatbinary` to merge these architecture-specific files
  - let `clang` link the resulting object file with the host binary
- a `clanvlink` script to help with this last step
  - contact the clang developers to integrate this functionality into clang