GPU code optimization and auto-tuning made easy with Kernel Tuner:

A hands-on, bring your own code tutorial

Ben van Werkhoven and Alessio Sclocco

Netherlands eScience Center

September 20, 2021

Introduction: Ben van Werkhoven

Ben van Werkhoven
Senior Research Engineer
Netherlands eScience Center
b.vanwerkhoven@esciencecenter.nl

Background:

- 2010-2014 PhD “Scientific Supercomputing with Graphics Processing Units” at the VU University Amsterdam in the group of prof. Henri Bal
- 2014-now working at the Netherlands eScience Center as the GPU expert in many different scientific research projects

GPU Programming since early 2009, worked on applications in computer vision, digital forensics, climate modeling, particle physics, geospatial databases, radio astronomy, and localization microscopy

Introduction: Alessio Sclocco

Alessio Sclocco
eScience Research Engineer
Netherlands eScience Center
a.sclocco@esciencecenter.nl

Background:

- 2011-2012 junior researcher at VU Amsterdam
  - Working on GPUs for radio astronomy
- 2012-2017 PhD "Accelerating Radio Astronomy with Auto-Tuning" at VU Amsterdam
  - Under the supervision or professors Henri Bal and Rob van Nieuwpoort
- 2015-2016 scientific programmer at ASTRON, the Netherlands Institute for Radio Astronomy
  - Designing and developing a real-time GPU pipeline for the Westerbork radio telescope
- 2019 visiting scholar at Nanyang Technological University in Singapore
- 2017-2020 eScience Research Engineer at the Netherlands eScience Center
  - Radio astronomy, climate modeling, biology, natural language processing

Mentors

- Hanno Spreeuw
  - eScience Research Engineer
  - h.spreeuw@esciencecenter.nl

- Victor Azizi
  - eScience Research Engineer
  - v.azizi@esciencecenter.nl

Outline for the day

9:30	–	9:35	Opening and welcome
9:35	–	10:30	Introduction to GPU code optimization and auto-tuning
10:30	–	10:45	First hands-on session
10:45	–	11:00	Coffee break
11:00	–	11:30	Getting started with Kernel Tuner: integrating code, multi-dimensional problems, user-defined metrics
11:30	–	12:15	Second hands-on session
12:15	–	13:15	Lunch break
13:15	–	13:45	Intermediate topics: search space restrictions, GPU code optimizations, output verification, caching tuning results
13:45	–	14:30	Third hands-on session
14:30	–	14:45	Coffee break
14:45	–	15:15	Advanced topics: search optimization strategies, custom observers, creating performance portable applications
15:15	–	16:00	Fourth hands-on session
16:00	–	16:15	Closing

Administrative announcements

- We will have four sessions in which we start with introducing some new concepts followed by a hands-on session and a break
- The hands-on sessions include example kernels, but you are also welcome to experiment with your own code
- We will use Google Colab for the hands-on sessions, so you don't need to have access to a GPU or install anything locally
- You can download the slides and the hands-on notebooks here:
  - https://github.com/benvanwerkhoven/kernel_tuner_tutorial

Outline for this session

- Auto-tuning success story
- Overview of auto-tuning technologies and where Kernel Tuner fits
- Introduction to Kernel Tuner
- Kernel Tuner installation and setup
- First hands-on session
- Break

netherlands eScience center

Auto-tuning success story

netherlands eScience center

Astronomy and radio astronomy

- “Astronomy is a natural science that studies celestial objects and phenomena. It applies mathematics, physics, and chemistry, in an effort to explain the origin of those objects and phenomena and their evolution.” — Wikipedia

- “Radio astronomy is a subfield of astronomy that studies celestial objects at radio frequencies.” — Wikipedia

Modern radio telescopes

- “Radio astronomy is conducted using large radio antennas referred to as radio telescopes, that are either used singularly, or with multiple linked telescopes utilizing the techniques of radio interferometry and aperture synthesis.” — Wikipedia

Westerbork Telescope, photo by Elodie Burrillon

netherlands eScience center

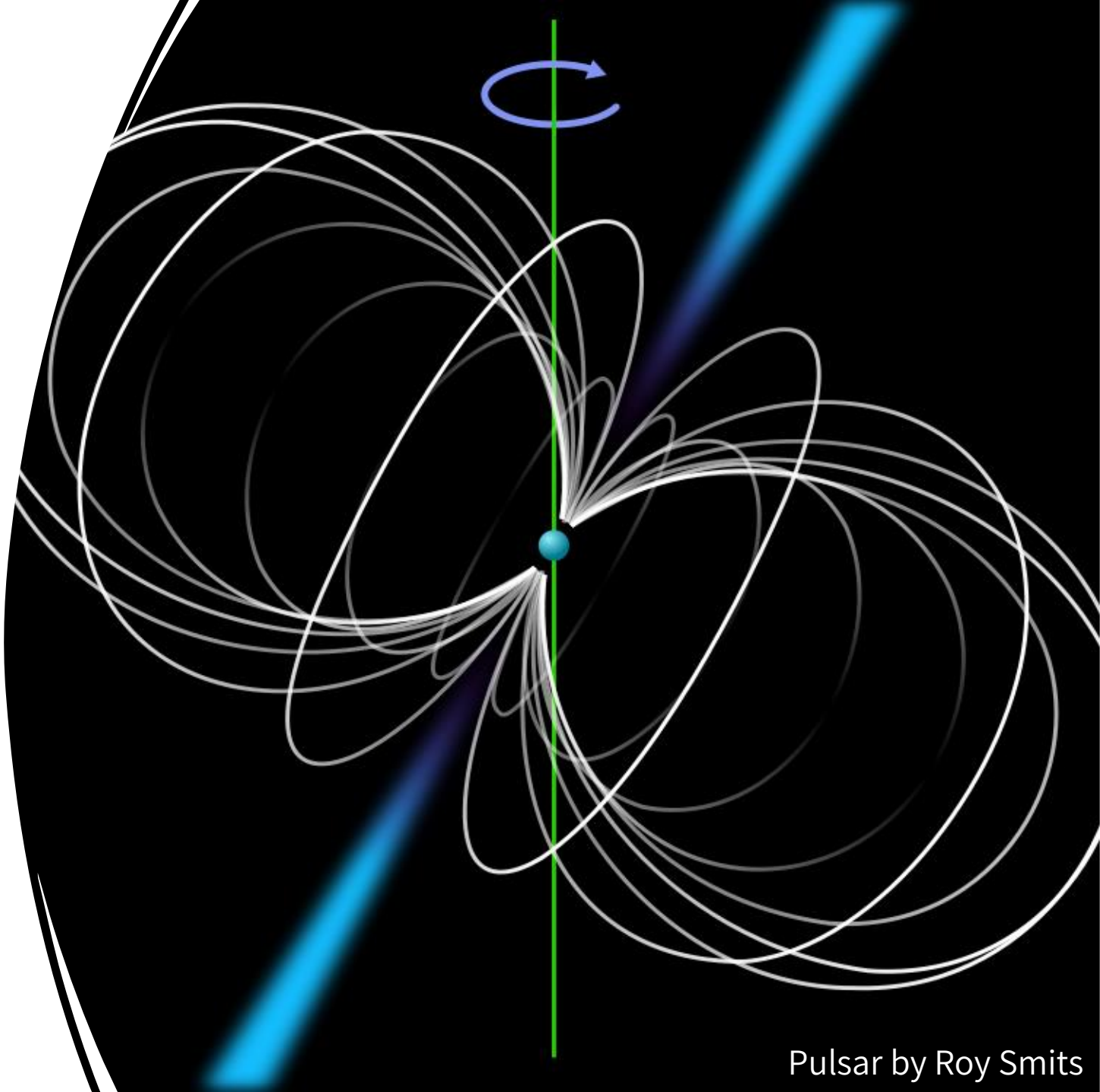
Challenges of radio astronomy

- Processing of data in real-time
- Many algorithms are memory bound
- Real applications seldom achieve high efficiency
- Portability
  - Telescopes operate for decades
  - Computing infrastructure has a much shorter lifespan
- Power constraints

Searching for transients

- Finding transients is a hot-topic in astronomy
  - Pulsars
  - Fast Radio Bursts
  - Supernovae

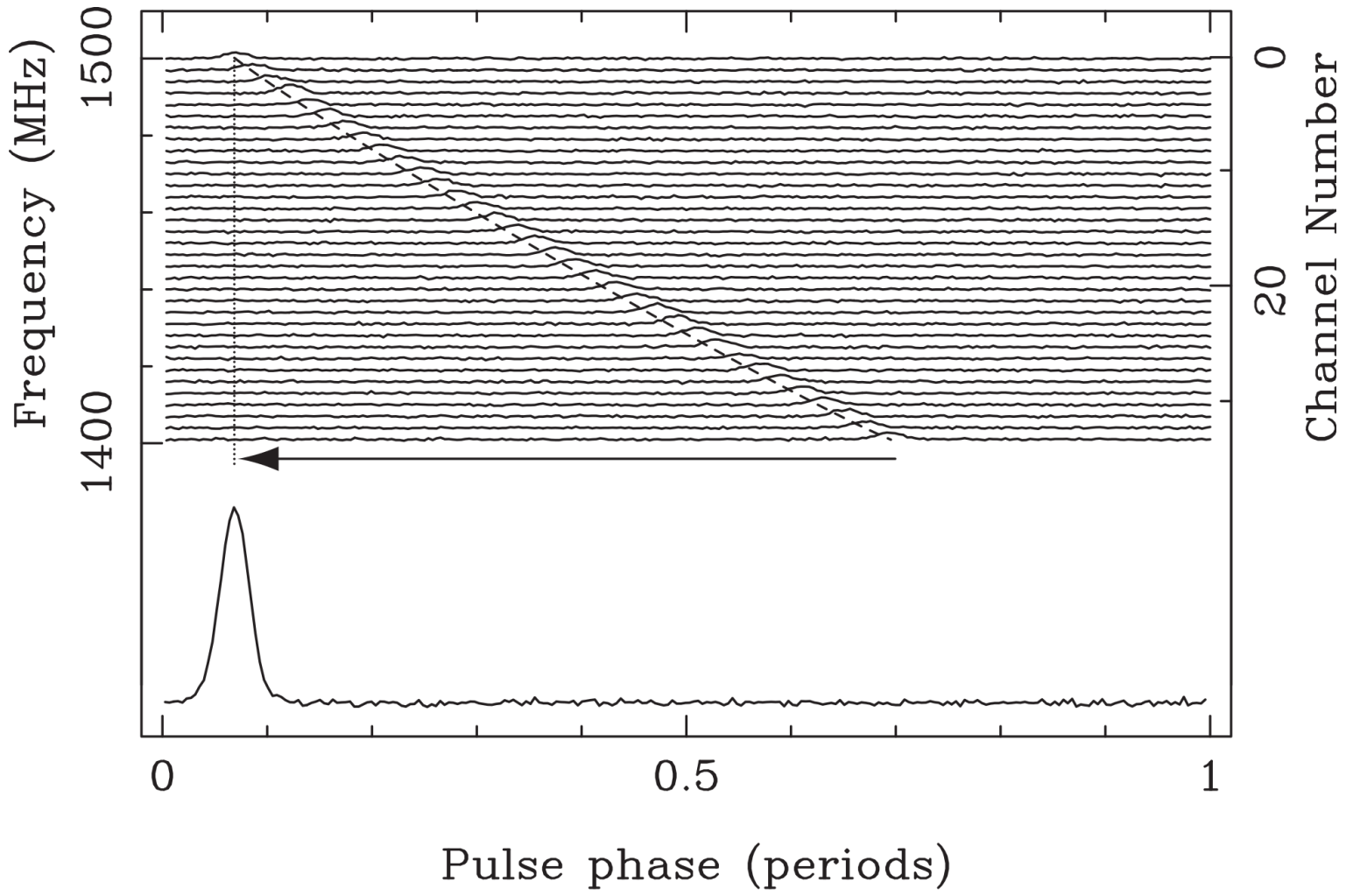
- Interesting objects, but difficult to find
  - Radio Frequency Interference (RFI)
  - Dispersion, scintillation, scattering
  - Gravitational interactions
  - Fast periods
  - Faint signals



Pulsar by Roy Smits

Dispersion and dedispersion

Dispersion by Lorimer



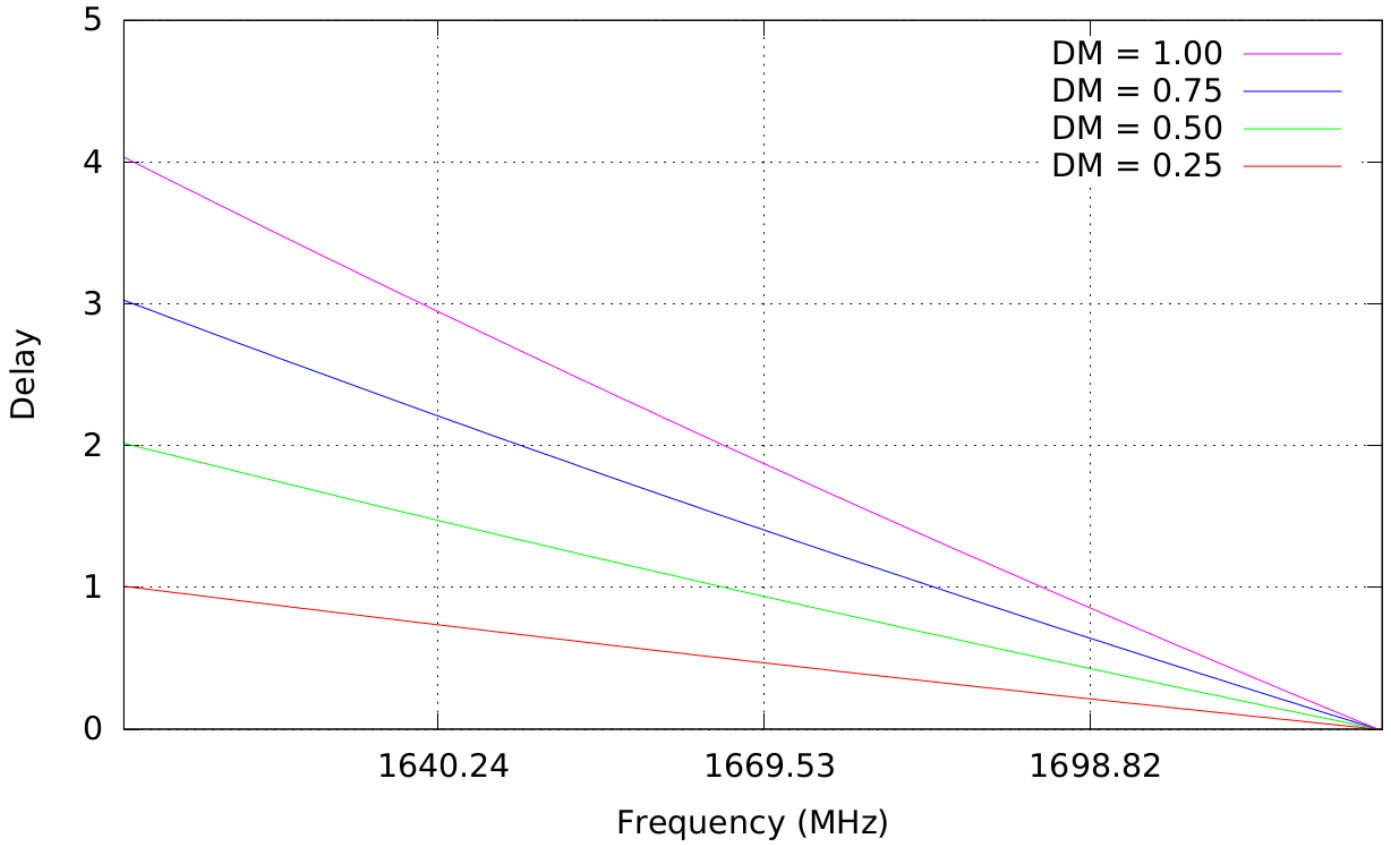
netherlands eScience center

The complexity of dedispersion

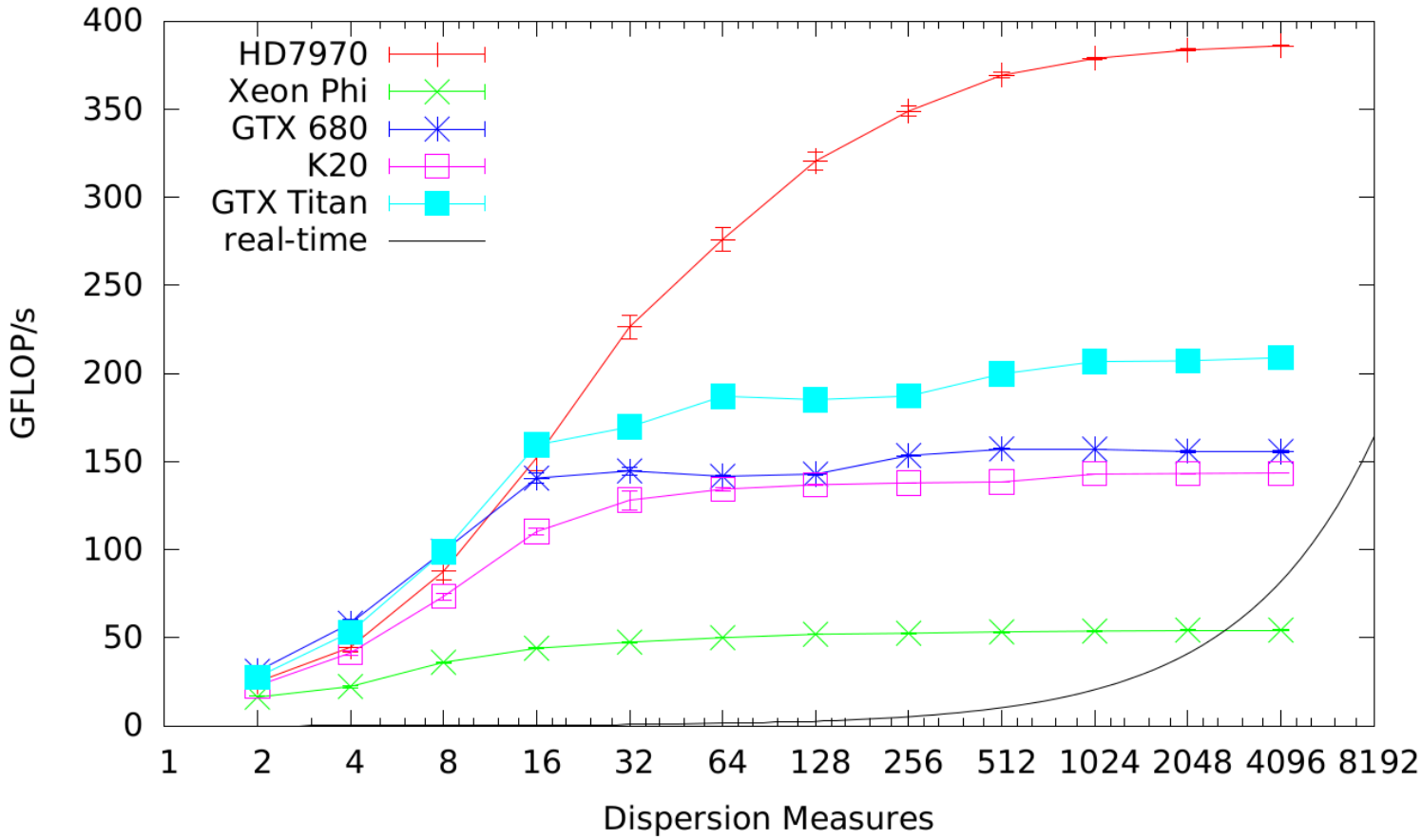
- Complexity:
  - Single step: O($f * t$)
  - Search: O($b * d * f * t$)
    - f is the number of frequency channels
    - t is the number of time steps
    - b is the number of beams
    - d is the number of dispersion measures (DMs)
- Naturally parallel
- Memory-bound
  - Arithmetic intensity less than 0.25

Dedispersion GPU optimizations

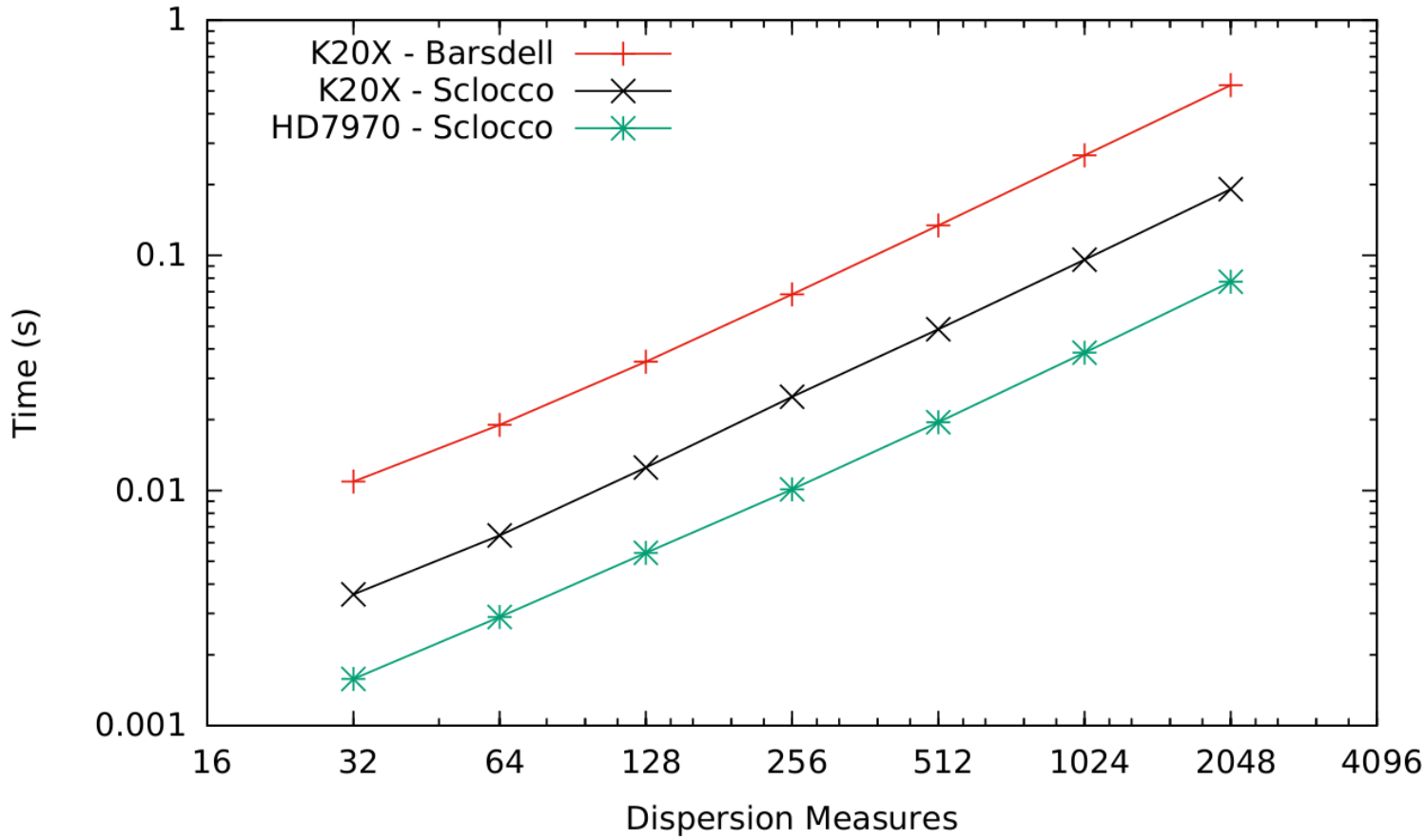
- Discrete delays partially overlap
  - Possible data reuse
- Data reuse affects arithmetic intensity
- Computing more than one DM per thread
- Auto-tuning necessary



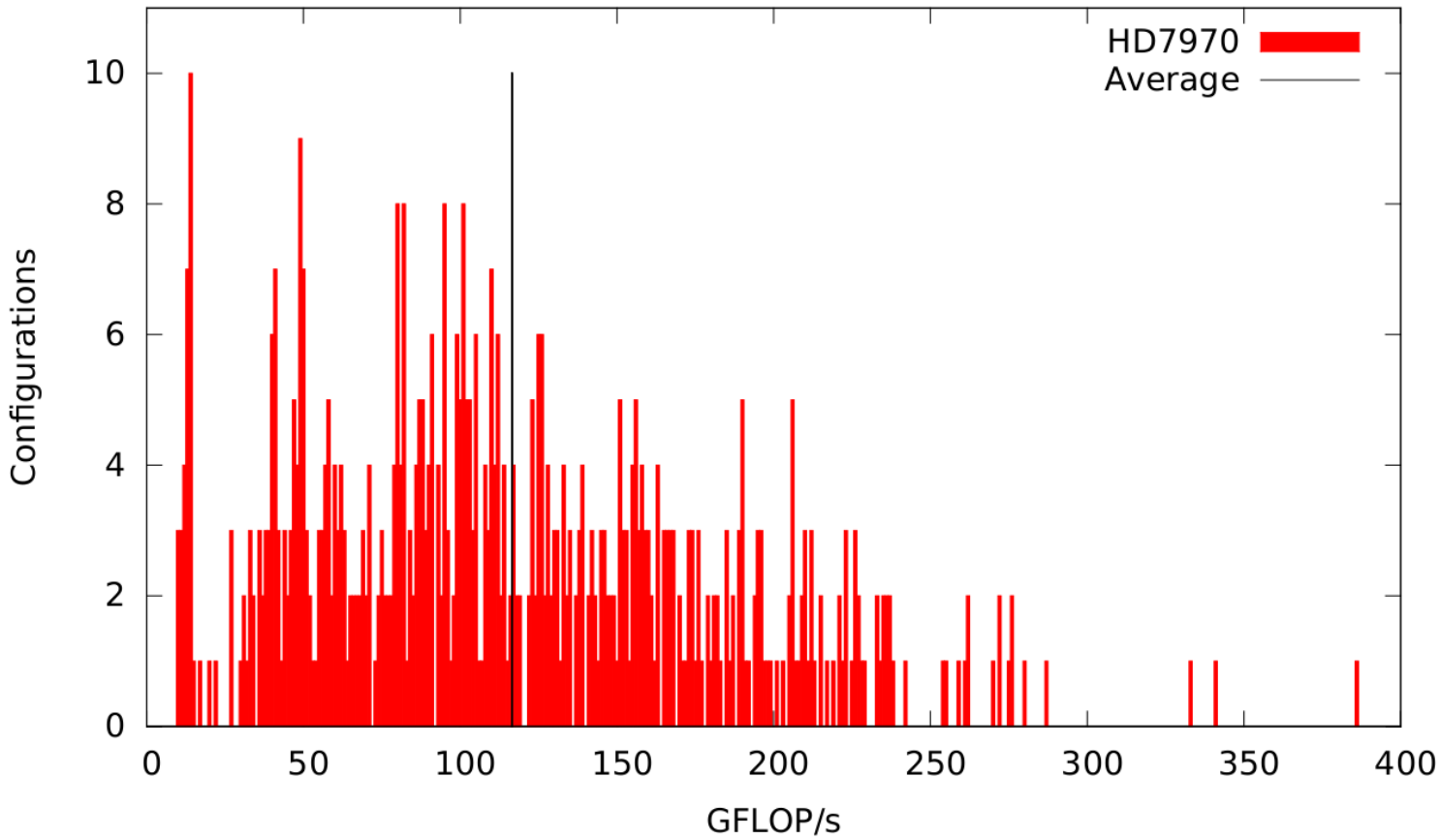
Performance results



Portability and auto-tuning



Auto-tuning challenges



netherlands eScience center

Takeaways

- Auto-tuning has been fundamental for dedispersion
- High performance for different GPUs achieved only via tuning
- Performance portability allows to choose the best hardware
- Auto-tuning is difficult

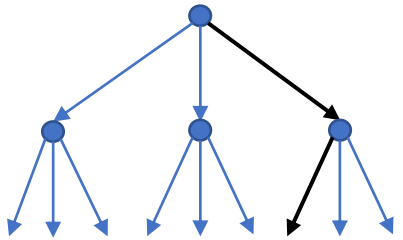
- This tuned dedispersion algorithm is part of AMBER, the time-domain transient search pipeline used at Westerbork
  - Recently contributed to a Nature paper

netherlands eScience center

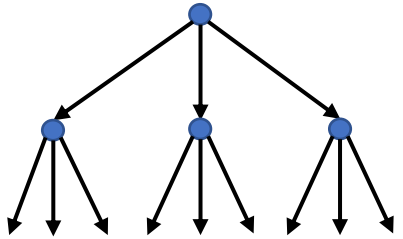
Overview of auto-tuning technologies

Manual optimization versus auto-tuning

- Optimizing code manually you iteratively:
  - Modify the code
  - Run a few benchmarks
  - Revert or accept the change



- With auto-tuning you:
  - Write a templated version of your code or a code generator
  - Benchmark the performance of all versions of the code



Compiler vs software auto-tuning

- Compiler auto-tuning
  - Does not modify the source code
  - Needs to work with assumptions the compiler can take
  - Usually tunes compiler flags for the application as a whole

- Software auto-tuning
  - Works with a code template or code generator
  - Can tune more diverse code optimizations and even competing implementations of the same algorithm
  - Usually tunes individual kernels or pipelines of kernels

Model-based versus empirical tuning

- Model-based auto-tuning
  - Requires a model or oracle to predict the performance of different kernel configurations on the target hardware
  - Typically used for relatively simple, regular, and well-understood kernels, such as stencils or Fourier transforms
  - Models are difficult construct and generalize to any kernel or any hardware
- Empirical auto-tuning
  - Uses benchmarking to determine performance of different configurations
  - Easier to implement, but requires access to the hardware, compilation, and benchmarking
  - Optimization strategies are used to speedup tuning process

Run-time vs compile-time auto-tuning

- Run-time auto-tuning
  - Performs benchmarking while the application is running
  - Host application needs substantial modification
  - Good when performance strongly depends on input data
- Compile-time auto-tuning
  - Allows to separate tuning code from the host application
  - Host application can be in any programming language and is usually not modified to support tuned kernels
  - Requires kernels to be compiled in isolation
  - Good for working on kernels with big optimization spaces in isolation

Summary

- Advantages of auto-tuning over manual optimization are:
  - Allows to see past local minima and unexpected interactions between different code optimizations
  - Simply rerun the tuner for different hardware
- Software auto-tuning is more flexible and powerful than only tuning compiler flags
- Tuning using performance models is much more efficient, but difficult to achieve
- Use run-time tuning for when the input data strongly influences kernel performance or when kernels are difficult to separate from host application
- Use compile-time tuning for heavily-optimized kernels that can be compiled and benchmarked in isolation
- Kernel Tuner is an empirical compile-time software auto-tuner

netherlands eScience center

Kernel Tuner

Auto-tuning GPU applications

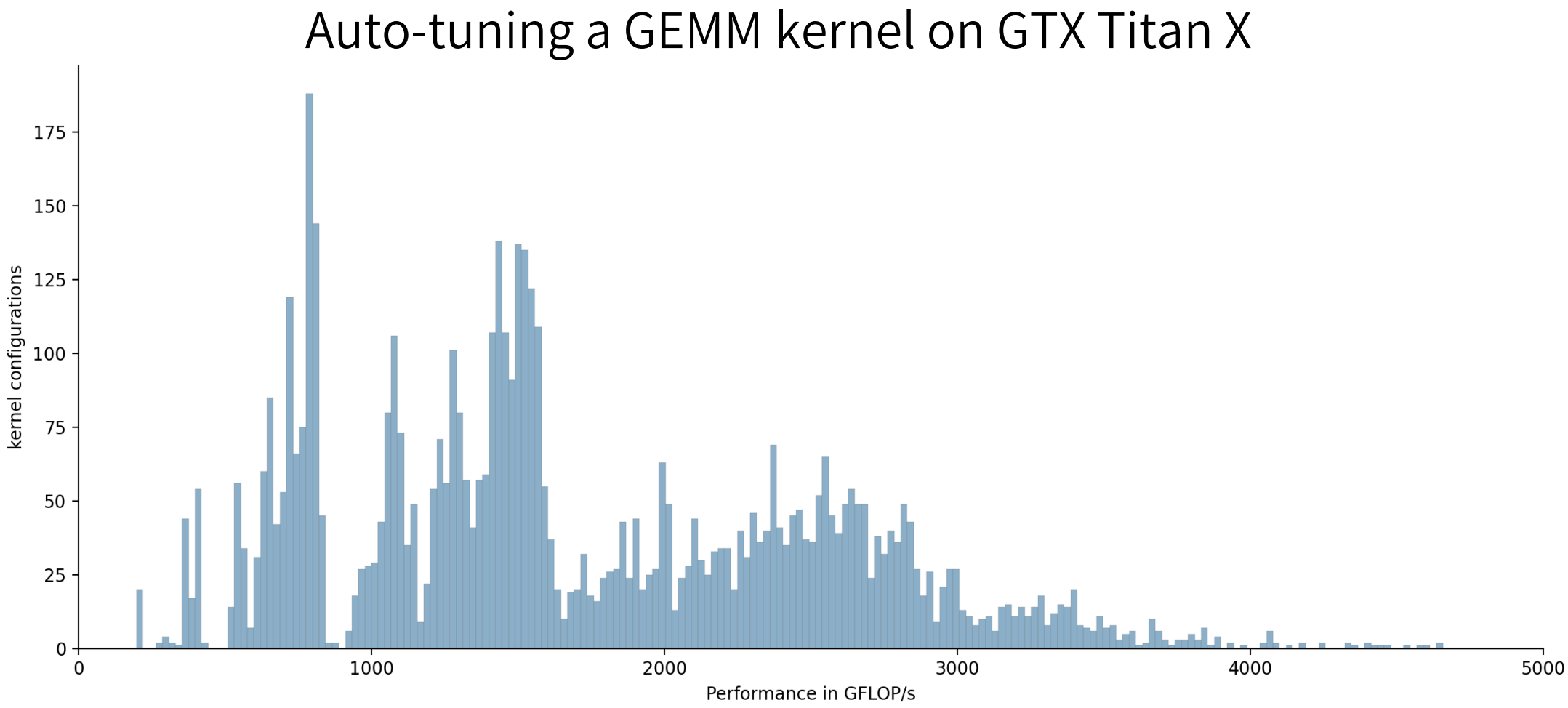
To maximize GPU code performance, you need to find the best combination of:

- Different mappings of the problem to threads and thread blocks
- Different data layouts in different memories (shared, constant, …)
- Different ways of exploiting special hardware features
- Thread block dimensions
- Work per thread in each dimension
- Loop unrolling factors
- Overlapping computation and communication
- …

Problem:

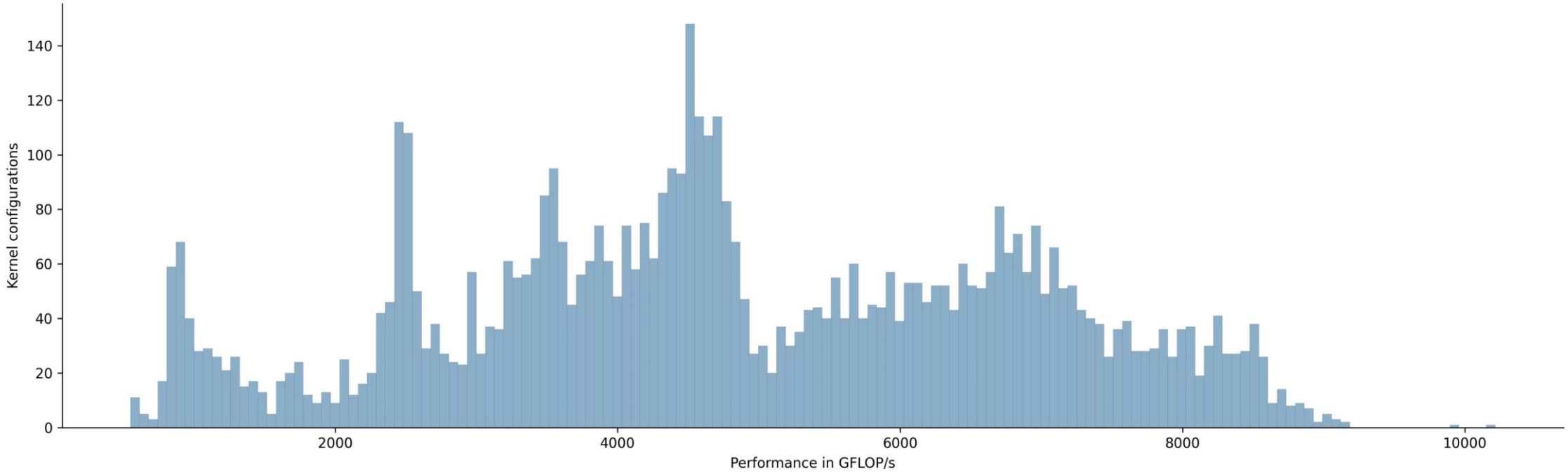
- Creates a very large design space

Large search space of kernel configurations



Large search space of kernel configurations

Auto-tuning a Convolution kernel on Nvidia A100



Kernel Tuner

Easy to use:

- Can be used directly on existing kernels and code generators
- Inserts no dependencies in the kernels or host application
- Kernels can still be compiled with regular compilers

Supports:

- Tuning functions in OpenCL, CUDA, C, and Fortran
- Large number of effective search optimizing algorithms
- Output verification for auto-tuned kernels and pipelines
- Tuning parameters in both host and device code
- Using code generators
- Unit testing GPU code
- …

https://github.com/benvanwerkhoven/kernel_tuner

Minimal example

```
import numpy
from kernel_tuner import tune_kernel

kernel_string = """
__global__ void vector_add(float *c, float *a, float *b, int n) {
    int i = blockIdx.x * block_size_x + threadIdx.x;
    if (i<n) {
        c[i] = a[i] + b[i];
    }
}"""

n = numpy.int32(1e7)
a = numpy.random.randn(n).astype(numpy.float32)
b = numpy.random.randn(n).astype(numpy.float32)
c = numpy.zeros_like(b)
args = [c, a, b, n]
tune_params = {"block_size_x": [32, 64, 128, 256, 512]}

tune_kernel("vector_add", kernel_string, n, args, tune_params)
```

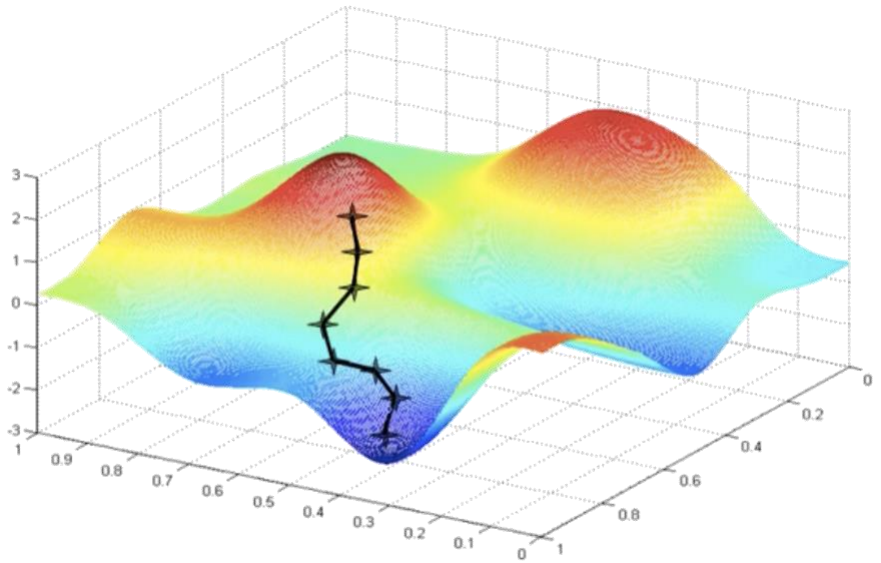
What it does

- Compute the Cartesian product of all tuning parameters
  - Remove instances that fail any of the restrictions
- For each instance in the parameter space (brute force tuning):
  - Insert preprocessor definitions for each tuning parameter
  - Compile the kernel created for this instance
  - Benchmark the kernel
  - Store the averaged execution time
- Return the full data set with the averaged run time for each instance

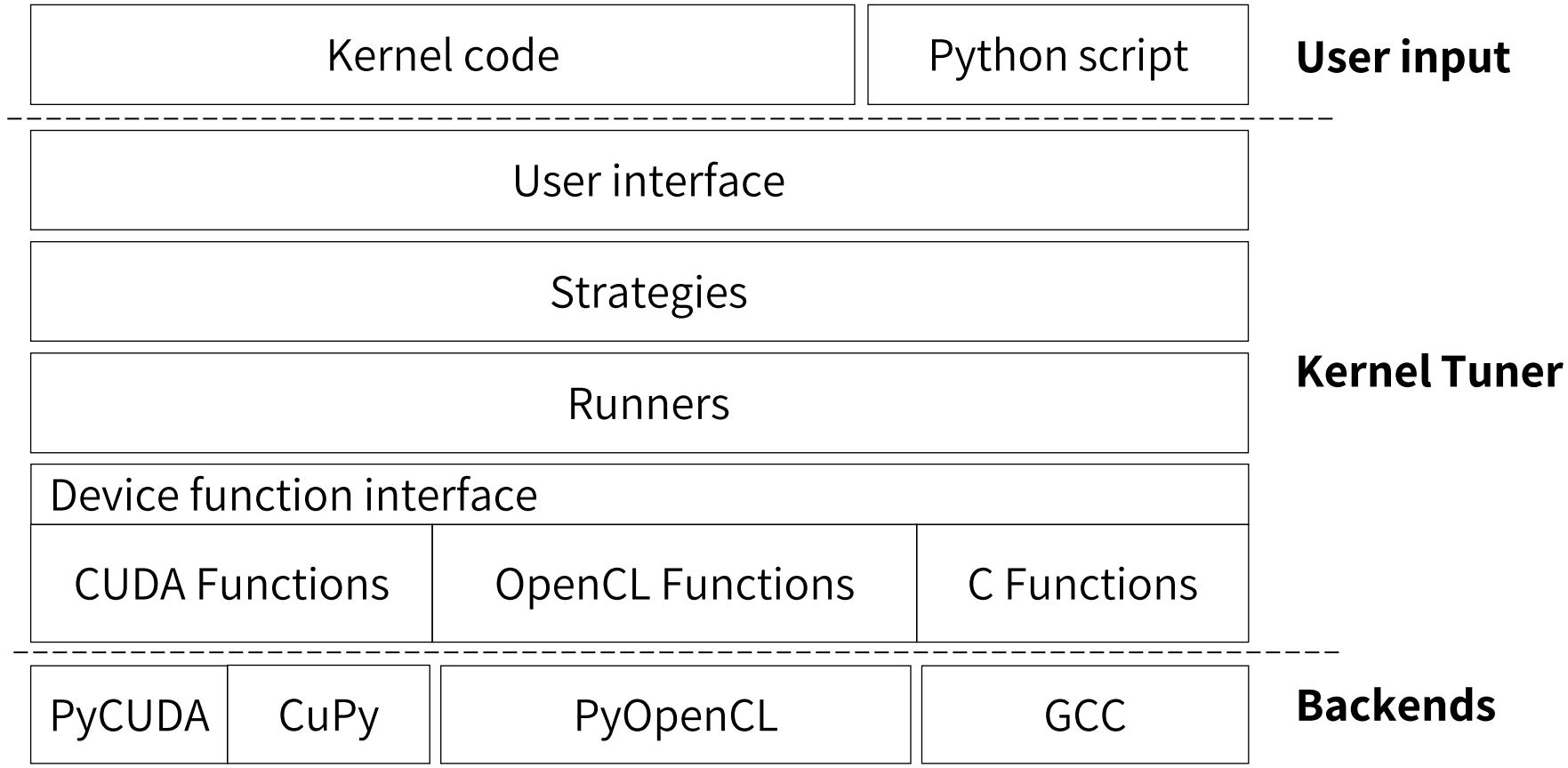
Optimization strategies in Kernel Tuner

- Local optimization
  - Nelder-Mead, Powell, CG, BFGS, L-BFGS-B, TNC, COBYLA, and SLSQP

- Global optimization
  - Basin Hopping
  - Simulated Annealing
  - Differential Evolution
  - Genetic Algorithm
  - Particle Swarm Optimization
  - Firefly Algorithm
  - Bayesian Optimization
  - Multi-start local search
  - …



Kernel Tuner architecture



Installation on your system

- Prerequisites:
  - Python 3.6 or newer
  - CUDA or OpenCL device with necessary drivers and compilers installed
  - PyCUDA or PyOpenCL installed
- To install Kernel Tuner:
  - pip install kernel_tuner
- For more information:
  - https://benvanwerkhoven.github.io/kernel_tuner/install.html
- **Note**: this is not required for the hands-on sessions

netherlands eScience center

First hands-on session

netherlands eScience center

Introduction hands-on

- The first hands-on notebook is:
  - https://github.com/benvanwerkhoven/kernel_tuner_tutorial/blob/master/hands-on/cuda/00_Kernel_Tuner_Introduction.ipynb
- The goal of this hands-on is to install and run Kernel Tuner
  - Copy the notebook to your Google Colab and work there
- Please follow the instructions in the Notebook
- Feel free to ask questions to instructors and mentors

netherlands eScience center

Optional hands-on

- Done with the first hands-on already?
- Play with this other notebook
  - https://github.com/benvanwerkhoven/kernel_tuner_tutorial/blob/master/hands-on/cuda/Kernel_Tuner_Tutorial.ipynb
- Start experimenting with your own code
- Feel free to ask questions to instructors and mentors