Automatically Exploring GPU Program Design Spaces for Increased Productivity and Sustainability

Ben van Werkhoven
Netherlands eScience Center

Towards Improvement of Sustainability and Productivity for Research Software

SIAM CSE 2023, March 1, 2023

Supercomputer application lifetimes

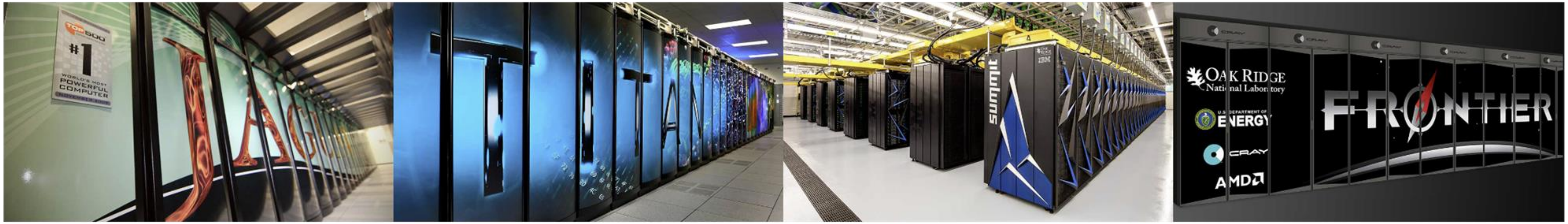
Some widely-used applications[1]:

Application	area	Initial release	Latest release
VASP	Atomic-scale materials	1989	2022
LAMMPS	Atomic/molecular simulation	1995	2022
cp2k	Quantum chemistry & solid-state physics	2000	2023
GROMACS	Molecular dynamics simulations	1991	2023
NEMO	Ocean circulation model	1998 (components dating back to 1980s)	2022

- Average age: 27.8 years

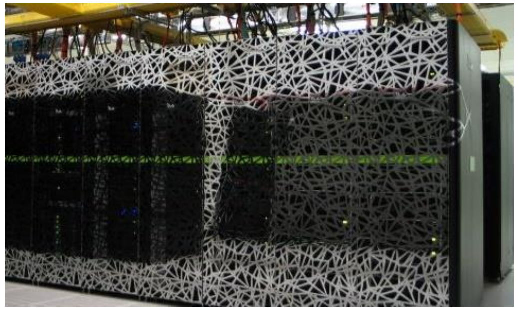
[1]according to Archer2 usage data, release dates from Wikipedia and nemo-ocean.eu

Supercomputer lifetimes



JAGUAR 2009 | **TITAN** 2012 | **SUMMIT** 2018 | **FRONTIER** 2022

Huygens
2007



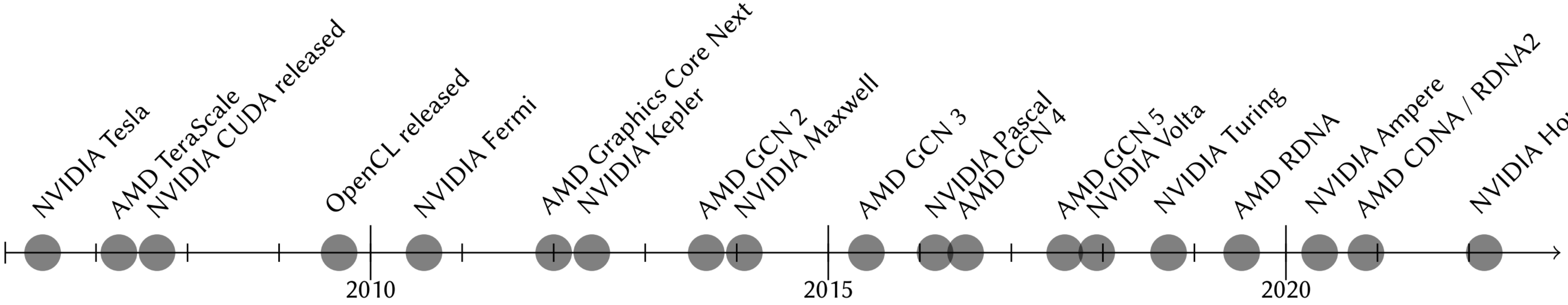
Cartesius
2013

Snellius
2021

Average lifetime: 5.4 years

https://www.olcf.ornl.gov/frontier/, https://www.surf.nl/

GPU architecture lifetimes



Average lifetime: 1.96 years

Optimization Techniques for GPU Programming
Pieter Hijma, Stijn Heldens, Alessio Sclocco, Ben van Werkhoven, and Henri Bal
ACM Computing surveys 2022

Sustainability and productivity problem

Achieving high-performance on GPUs requires optimizing the code to efficiently use the underlying hardware

Problem:

- How can applications adapt to new architectures every two years?

How to not optimize GPU code …

```
#define LOG2_WARP_SIZE 5U
#define WARP_SIZE (1U << LOG2_WARP_SIZE)

// May change on future hardware, so better parametrize the code
#define SHARED_MEMORY_BANKS 16

// Threadblock size: must be a multiple of (4 * SHARED_MEMORY_BANKS)
#define HISTOGRAM64_THREADBLOCK_SIZE (4 * SHARED_MEMORY_BANKS)

// Warps ==subhistograms per threadblock
#define WARP_COUNT 6
```

```
#if __CUDA_ARCH__ == 700 || __CUDA_ARCH__ == 720
  unsigned statY    = firstBlockY + statYoffset + NR_BLOCKS_PER_TCM_Y * y + ((threadIdx.x >> 3) & 2) + (threadIdx.x & 4);
  unsigned statX    = firstBlockX + statXoffset + NR_BLOCKS_PER_TCM_X * x + ((threadIdx.x >> 2) & 2);
  unsigned polY     = threadIdx.x & 1;
  unsigned polX     = (threadIdx.x >> 1) & 1;
#elif __CUDA_ARCH__ == 750 || __CUDA_ARCH__ == 800 || __CUDA_ARCH__ == 860
  unsigned statY    = firstBlockY + statYoffset + NR_BLOCKS_PER_TCM_Y * y + ((threadIdx.x >> 3) & 3);
  unsigned statX    = firstBlockX + statXoffset + NR_BLOCKS_PER_TCM_X * x + ((threadIdx.x >> 1) & 1);
  unsigned polY     = (threadIdx.x >> 2) & 1;
  unsigned polX     = threadIdx.x & 1;
#endif
```

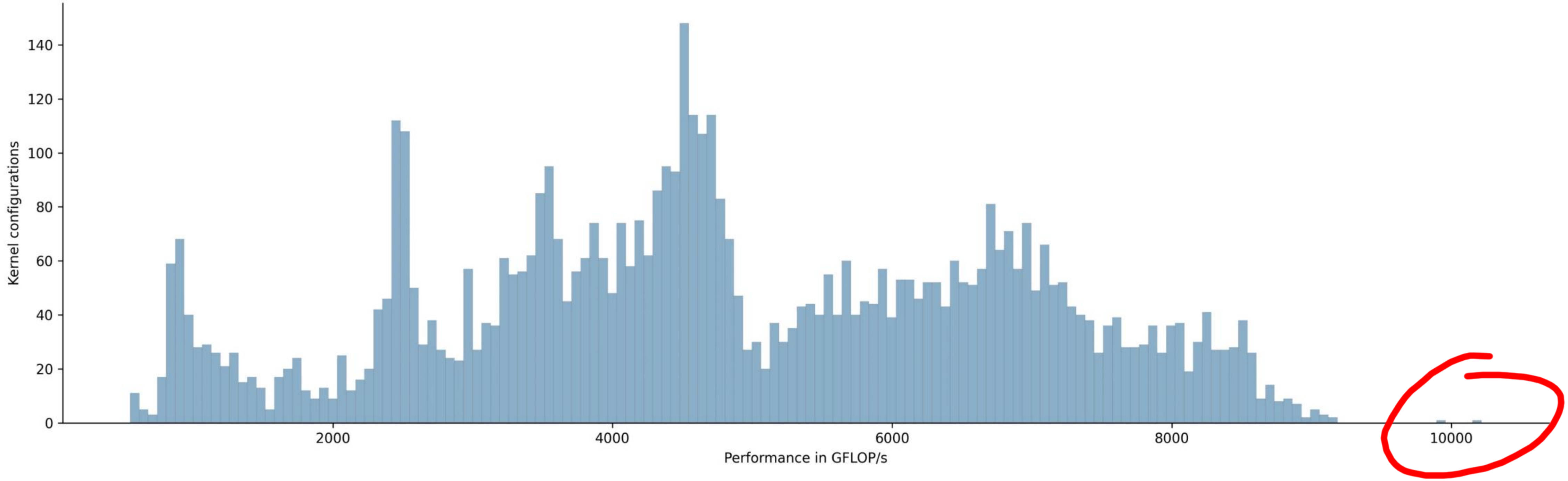
Proposed solution: tunable code

- Parametrize the code:
  - based on implementation choices, not architecture features
  - without hard coding constants in the source code

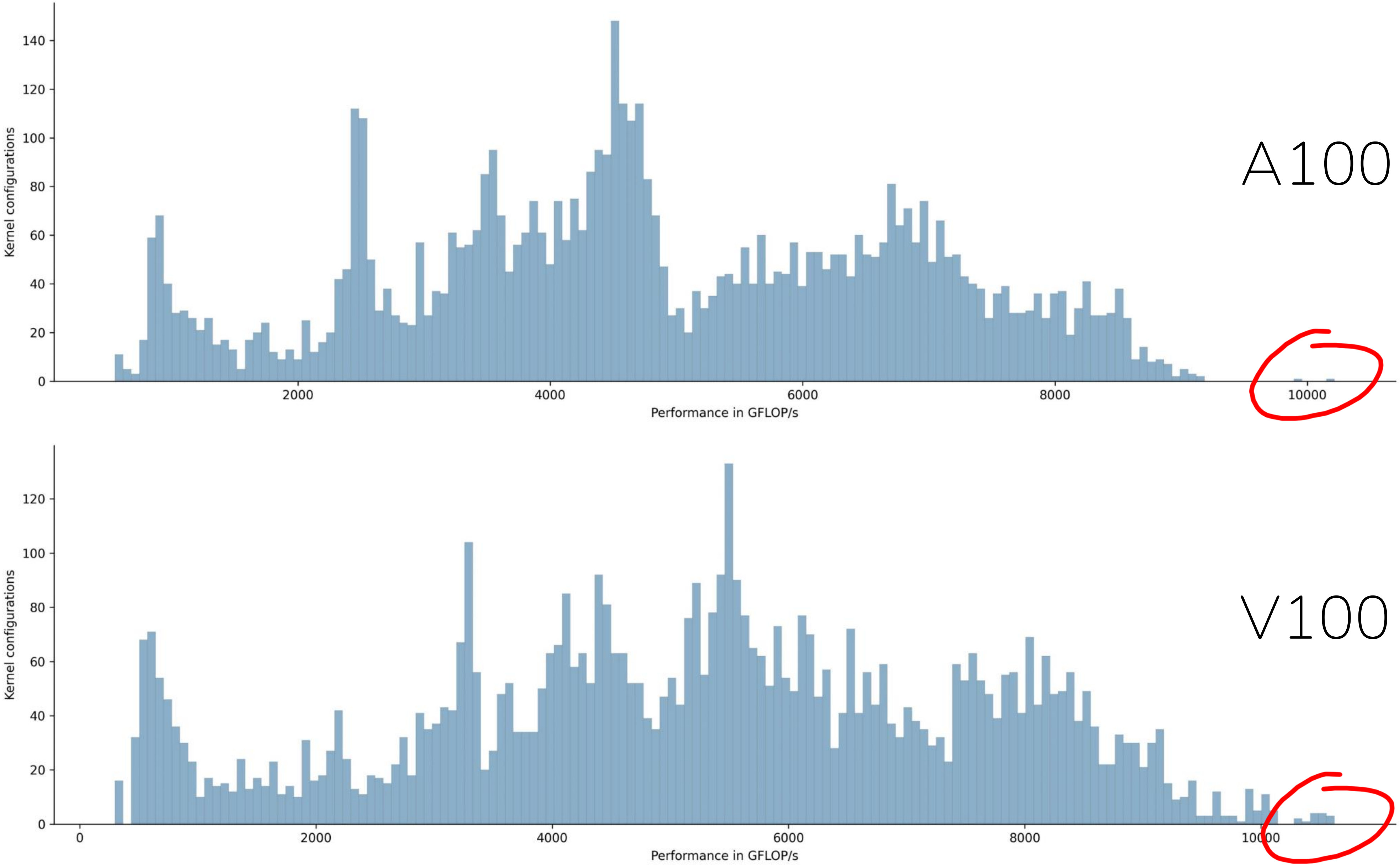
- All parameters in the code combined define the program design space

Large search space of kernel configurations

Exploring different designs of a Convolution kernel on Nvidia A100



On different GPUs ...



top 5% on A100

8,16,2,2,1,0
8,32,2,1,1,0

top 5% on V100

4,32,5,1,1,0
4,32,5,1,1,1
32,4,1,4,0,0
32,4,1,4,0,1
32,4,1,4,1,0
32,4,1,4,1,1
64,2,1,4,0,0
64,2,1,4,1,0
64,4,1,4,0,0
64,4,1,4,0,1
64,4,1,4,1,0
64,4,1,4,1,1
128,2,1,8,0,0
128,2,1,8,1,0

Kernel Tuner – *A Python tool for auto-tuning GPU kernels*

- Started in 2016, now developed by the Kernel Tuner developers team
- Funded by several national and European projects

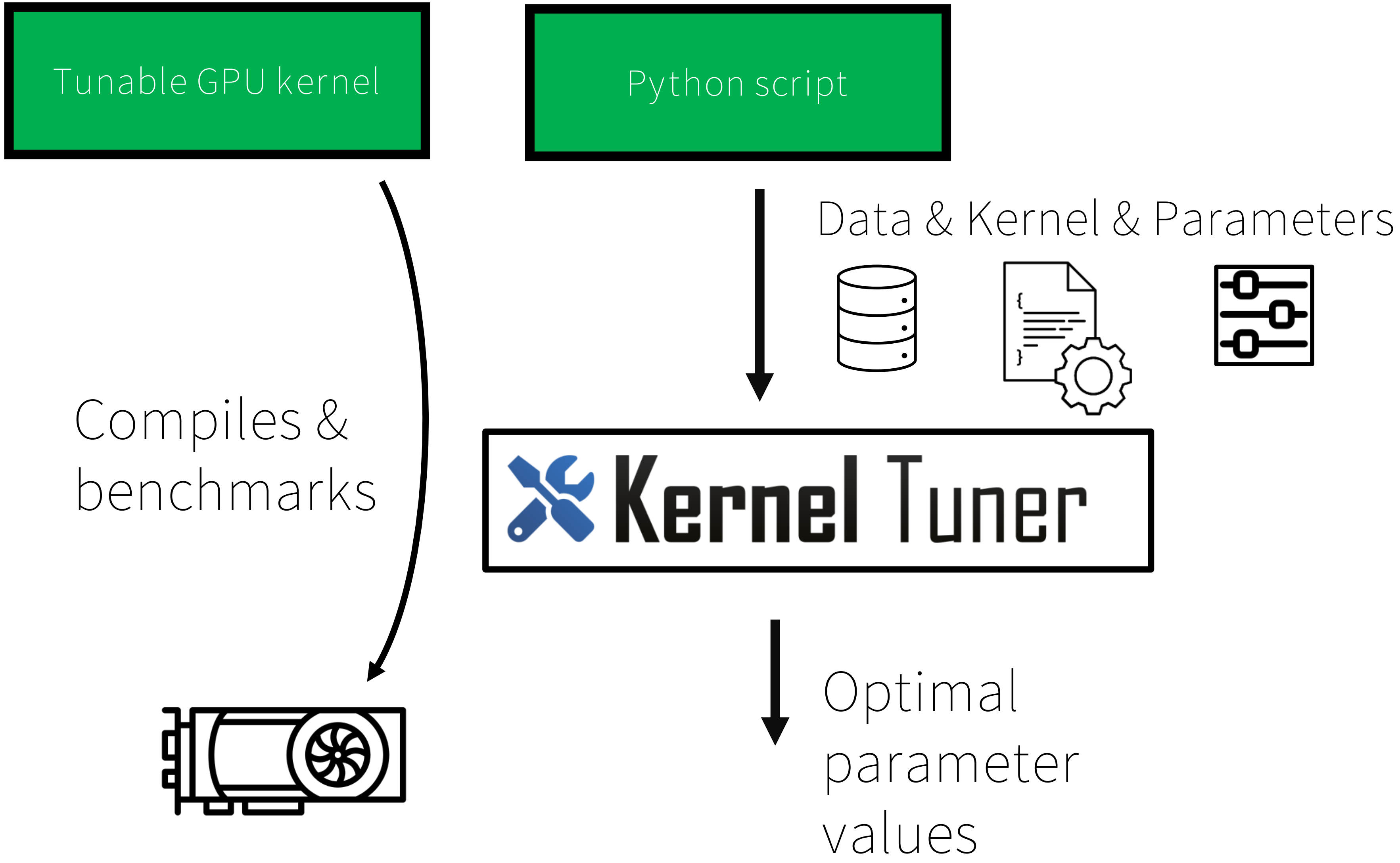
- Supports many different use cases
- Allows testing GPU kernels from Python
- Implements 20+ search optimization algorithms

- Actively used for research software development in many different projects

Kernel Tuner developers team

https://github.com/KernelTuner/kernel_tuner

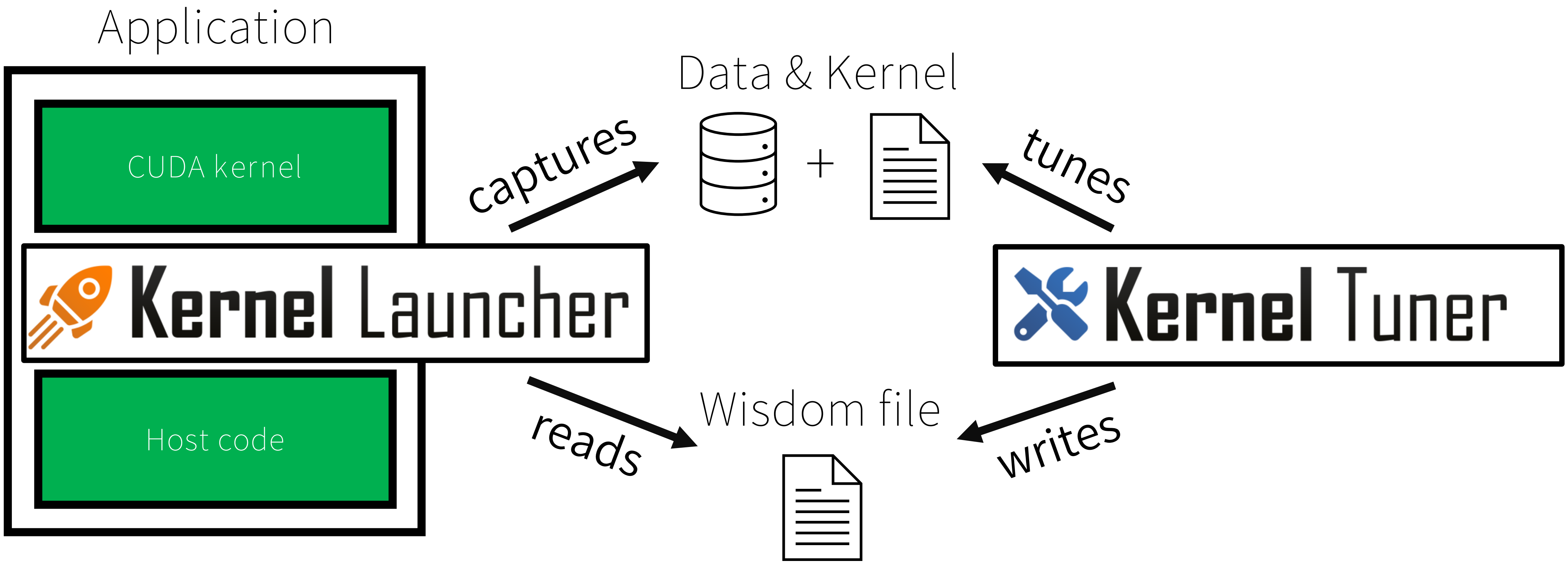
Kernel Tuner workflow



Auto-tuning productivity challenges

- Where are the parameters for the kernels stored/defined?
  - Keeping Python script and the GPU code in sync is not ideal
- How to describe input/output data of all kernels to the tuner?
  - Recreating or dumping & reading in data in Python is extra work
- How to feed the output of the tuner back into the application?
  - Solution depends on the host programming language

Kernel Launcher: C++ library



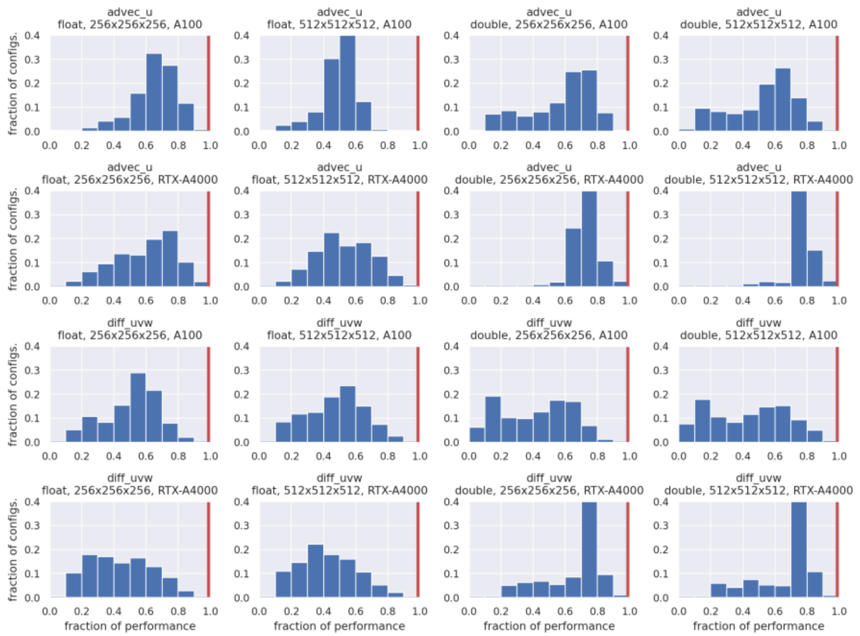
https://github.com/KernelTuner/kernel_launcher

Use case: MicroHH

Computational fluid dynamics code for simulation of turbulent flows in the atmosphere

Tuned kernels for different input domains, precisions, and target GPUs

Used Kernel Launcher to select optimal implementation in each scenario



Wageningen University: Bart van Stratum, Chiel van Heerwaarden
Netherlands eScience Center: Stijn Heldens, Gijs van den Oord, Alessio Sclocco, Ben van Werkhoven

esiwace CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

ESiWACE2 has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 823988

Conclusion

Auto-tuning allows to choose the optimal implementation that is otherwise limited by hard-coded constants in the source code

Improved sustainability: you can automatically adapt the code to new hardware

Improved productivity: optimizing code by hand is a waste of time

Kernel Tuner is open source: https://github.com/KernelTuner/kernel_tuner

For C++, Kernel Launcher: https://github.com/KernelTuner/kernel_launcher

Offline questions: b.vanwerkhoven@esciencecenter.nl

Funding

The CORTEX project has received funding from the Dutch Research Council (NWO) in the framework of the NWA-ORC Call (file number NWA.1160.18.316).

ESiWACE2 has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823988.

ESiWACE3 is funded by EuroHPC JU and national co-funding bodies under grant agreement No 101093054.

The ConFu project is funded by the Netherlands eScience Center (file number 00020223-A).