

CLUEstering: a novel high-performance clustering library for scientific computing

Simone Balducci^{1, 2, 3} Felice Pantaleo³ Marco Rovere³ Wahid Redjeb³
Aurora Perego^{3, 4} Francesco Giacomini²

¹University of Bologna ²INFN-CNAF ³CERN ⁴University of Milano Bicocca

ACAT 2025: 23rd International Workshop on Advanced Computing and Analysis Techniques in Physics Research
8-12 September 2025



The need for a density-based weighted clustering algorithm

- ▶ Density-based clustering finds clusters by indentifying regions with **high density of points**.
 - the robustness to noise of such algorithms makes them particularly good for experimental applications

The need for a density-based weighted clustering algorithm

- ▶ Density-based clustering finds clusters by indentifying regions with **high density of points**.
 - the robustness to noise of such algorithms makes them particularly good for experimental applications
- ▶ Weighted clustering allows to construct the clusters by considering a different weight for each point, representing their respective importance
 - this makes them useful for applications where points are associated with some signal measures or a-priori knowledge

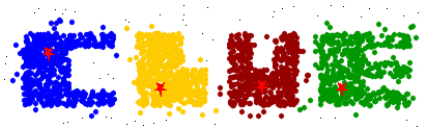
The need for a density-based weighted clustering algorithm

- ▶ Density-based clustering finds clusters by indentifying regions with **high density of points**.
 - the robustness to noise of such algorithms makes them particularly good for experimental applications
- ▶ Weighted clustering allows to construct the clusters by considering a different weight for each point, representing their respective importance
 - this makes them useful for applications where points are associated with some signal measures or a-priori knowledge
- ▶ Most density-based algorithms don't support weighted clustering out-of-the-box
 - They need hand-made modifications to the dataset or to the distance matrix

The need for a density-based weighted clustering algorithm

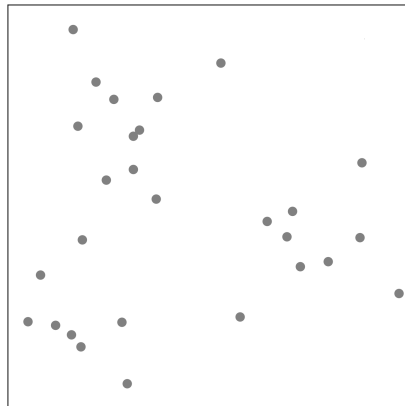
- ▶ Density-based clustering finds clusters by indentifying regions with **high density of points**.
 - the robustness to noise of such algorithms makes them particularly good for experimental applications
- ▶ Weighted clustering allows to construct the clusters by considering a different weight for each point, representing their respective importance
 - this makes them useful for applications where points are associated with some signal measures or a-priori knowledge
- ▶ Most density-based algorithms don't support weighted clustering out-of-the-box
 - They need hand-made modifications to the dataset or to the distance matrix
- ▶ There is a need for an alternative solution that combines the power of **density-based** algorithms with the generality of **weighted** clustering

The CLUE algorithm



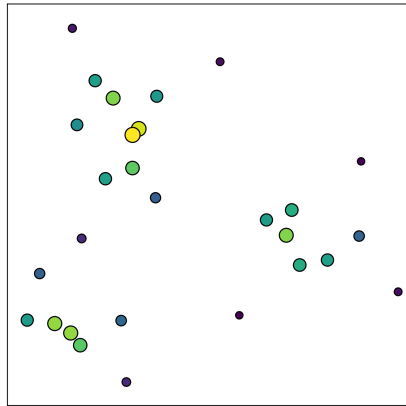
- ▶ CLUE (CLUstering of Energy) [1] is a density-based clustering algorithm used in the CMS experiment at LHC
- ▶ It was originally designed for the clustering of hits in the calorimeters
- ▶ Each point has a weight which is used when calculating the densities
- ▶ The weights are the energy deposit measurements of the detector layer sensors

Description of the algorithm



Description of the algorithm

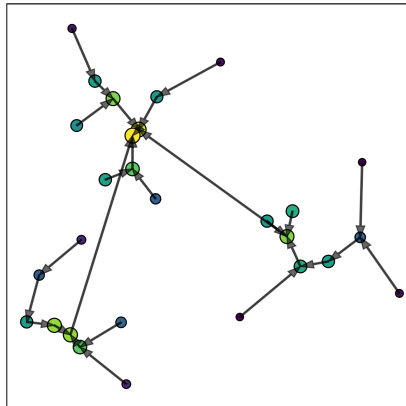
step 1 → Computation of the local density for each point



Description of the algorithm

step 1 → Computation of the local density for each point

step 2 → Selection of the *nearest highers*

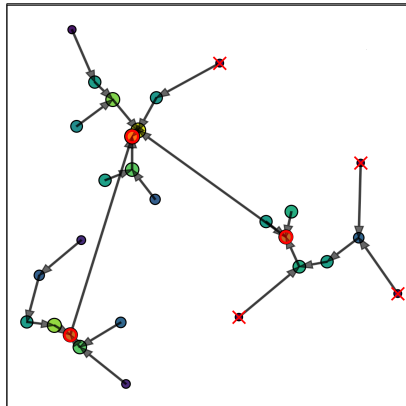


Description of the algorithm

step 1 → Computation of the local density for each point

step 2 → Selection of the *nearest highers*

step 3 → Finding clusters and outliers



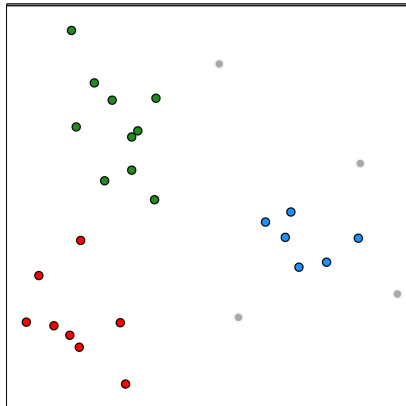
Description of the algorithm

step 1 → Computation of the local density for each point

step 2 → Selection of the *nearest highers*

step 3 → Finding clusters and outliers

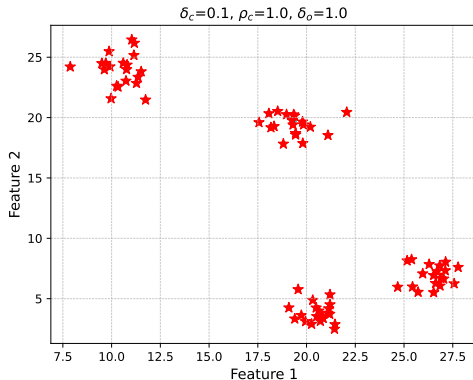
step 4 → Assigning hits to clusters



The parameters of CLUE

3 parameters:

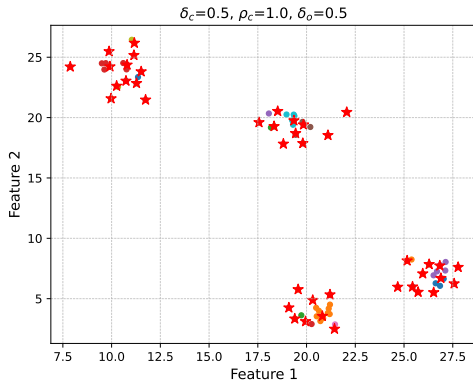
- ▶ δ_c , size of query range for computation of local density
- ▶ δ_o , size of query range for cluster extension
- ▶ ρ_c , density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

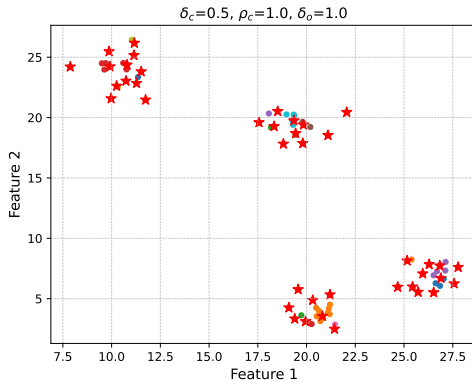
- ▶ δ_c , size of query range for computation of local density
- ▶ δ_o , size of query range for cluster extension
- ▶ ρ_c , density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

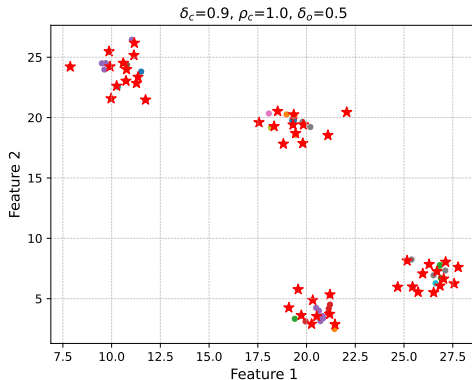
- ▶ δ_c , size of query range for computation of local density
- ▶ δ_o , size of query range for cluster extension
- ▶ ρ_c , density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

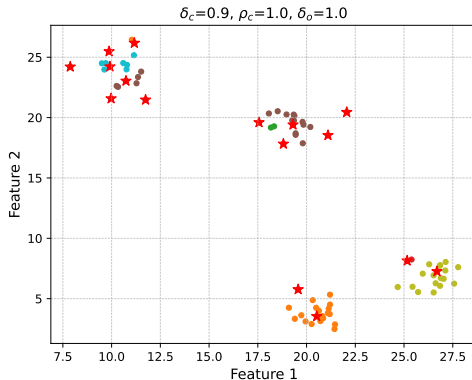
- ▶ δ_c , size of query range for computation of local density
- ▶ δ_o , size of query range for cluster extension
- ▶ ρ_c , density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

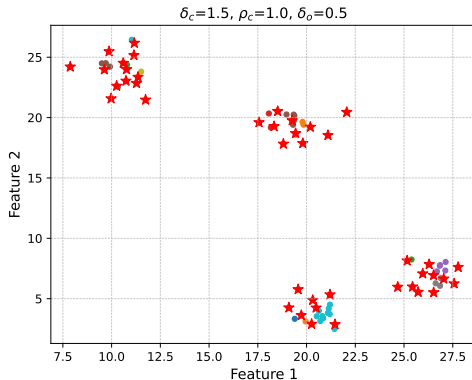
- ▶ δ_c , size of query range for computation of local density
- ▶ δ_o , size of query range for cluster extension
- ▶ ρ_c , density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

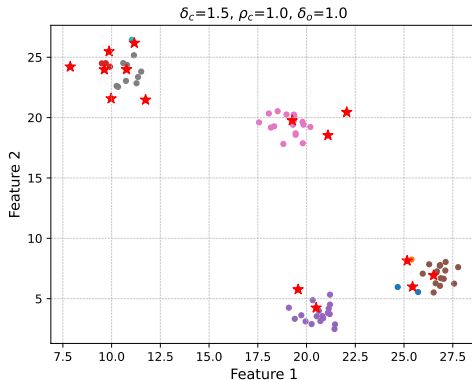
- ▶ δ_c , size of query range for computation of local density
- ▶ δ_o , size of query range for cluster extension
- ▶ ρ_c , density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

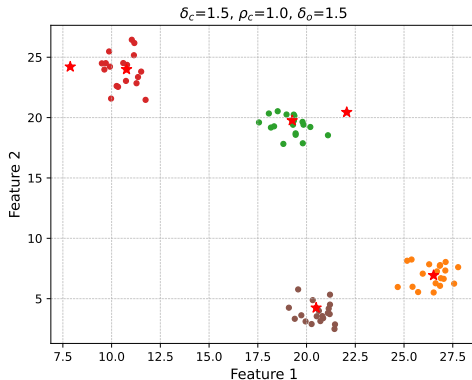
- ▶ δ_c , size of query range for computation of local density
- ▶ δ_o , size of query range for cluster extension
- ▶ ρ_c , density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

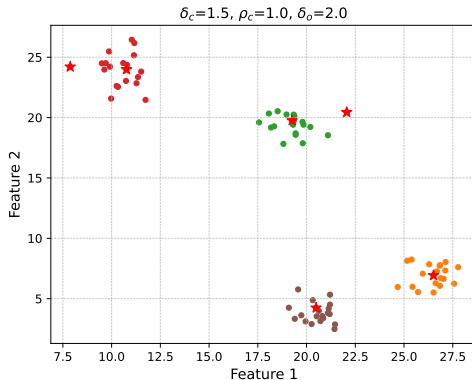
- ▶ δ_c , size of query range for computation of local density
- ▶ δ_o , size of query range for cluster extension
- ▶ ρ_c , density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

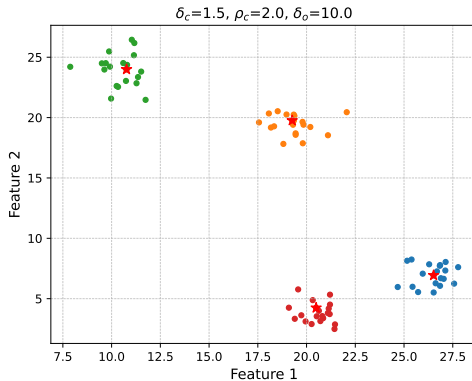
- ▶ δ_c , size of query range for computation of local density
- ▶ δ_o , size of query range for cluster extension
- ▶ ρ_c , density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

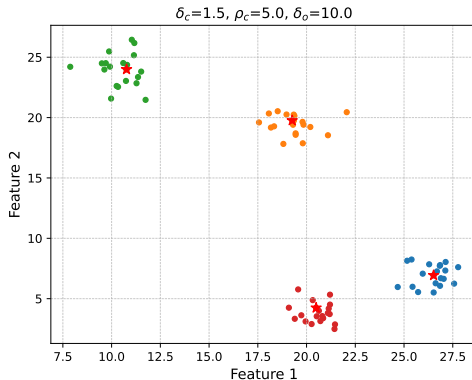
- ▶ δ_c , size of query range for computation of local density
- ▶ δ_o , size of query range for cluster extension
- ▶ ρ_c , density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

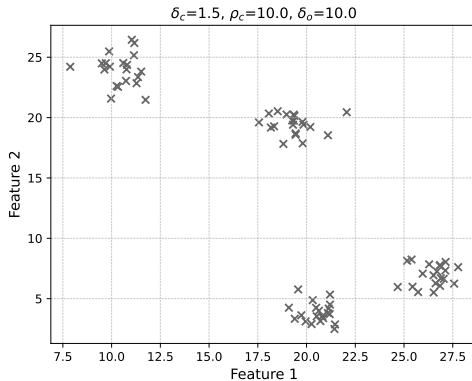
- ▶ δ_c , size of query range for computation of local density
- ▶ δ_o , size of query range for cluster extension
- ▶ ρ_c , density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

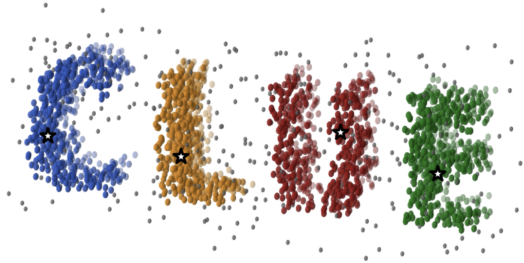
- ▶ δ_c , size of query range for computation of local density
- ▶ δ_o , size of query range for cluster extension
- ▶ ρ_c , density cut-off for promotion to cluster seed



From CLUE ...

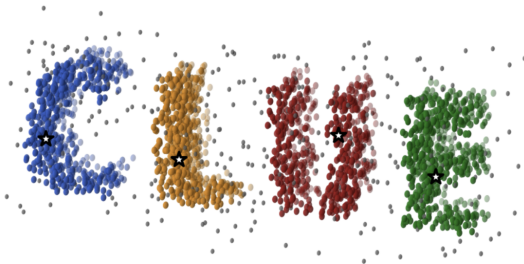
- ▶ CLUE was specifically tailored to work in the CMS detector
- ▶ 2-dimensional clustering for each of the layers
- ▶ Could not be used on a general dataset

From CLUE ... to CLUEstering



- ▶ CLUEstering [2][3][4] provides a generalization of CLUE
 - ➔ It's a general-purpose, header-only library
 - ➔ Applicable to any number of dimensions

From CLUE ... to CLUEstering



- ▶ CLUEstering [2][3][4] provides a generalization of CLUE
 - ➔ It's a general-purpose, header-only library
 - ➔ Applicable to any number of dimensions
- ▶ The backend and main interface are written in C++20
- ▶ It also provides a Python interface, which makes it easily usable by the scientific community

What can CLUEstering be used for?

- ▶ The implementation of CLUEstering makes it very general

What can CLUEstering be used for?

- ▶ The implementation of CLUEstering makes it very general
- ▶ It can be applied to a large variety of problems that use clustering
 - in particular density-based clustering

What can CLUEstering be used for?

- ▶ The implementation of CLUEstering makes it very general
- ▶ It can be applied to a large variety of problems that use clustering
 - in particular density-based clustering
- ▶ The main requirement is that data provides numerical coordinates

The software portability challenge

- ▶ Nowadays there are many different types of processors available
- ▶ Heterogeneous computing platforms are becoming increasingly popular for demanding tasks

The software portability challenge

- ▶ Nowadays there are many different types of processors available
- ▶ Heterogeneous computing platforms are becoming increasingly popular for demanding tasks



- ▶ To support several platforms, often many different code-bases have to be developed and maintained
- ▶ We want to write software that works on all possible platforms while achieving **near-native performance**

Performance portability in CLUEstering

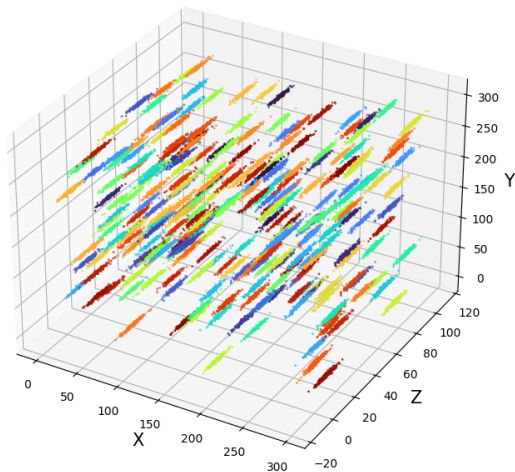
- ▶ CLUE is a highly-parallel algorithm
- ▶ It's designed to work well on heterogeneous platforms
- ▶ The backend of CLUEstering is implemented with Alpaka [5][6]
- ▶ Users can run the clustering on any backend with a single command

alpaka



An example dataset

- ▶ This dataset is representative of a hit reconstruction in an experiment at LHC
- ▶ Multiple datasets are generated increasing the total number of clusters
- ▶ The clusters are generated in a volume of $3 \times 3 \times 1$ meters, which mimics the endcap region of the CMS detector



How to use the library

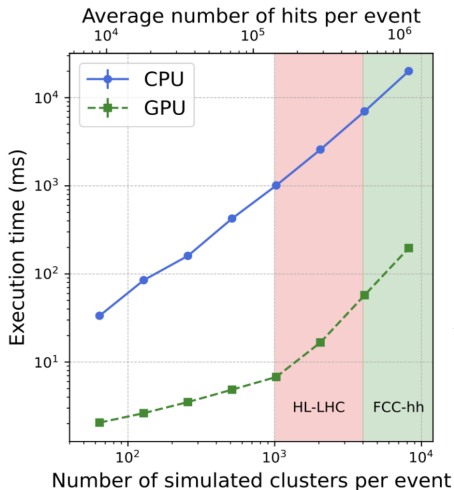
Both the C++ and the Python interfaces of the library are straightforward to use

```
1 #include <CLUEstering/CLUEstering.hpp>
2
3 int main() {
4     auto queue = clue::get_queue(0u);
5
6     clue::PointsHost<Ndim> h_points =
7     ↪ clue::read_csv<Ndim>(queue, "data.csv");
8     clue::PointsDevice<Ndim> d_points(queue,
9     ↪ h_points.size());
10
11     const float dc = 20.f, rhoc = 10.f, outlier = 20.f;
12     clue::Clusterer<Ndim> algo(queue, dc, rhoc,
13     ↪ outlier);
14
15     algo.make_clusters(queue, h_points, d_points);
16     auto cluster_ids = h_points.clusterIndexes();
17     auto seed_map = h_points.isSeed();
18 }
```

```
1 import CLUEstering as clue
2
3 clust = clue.clusterer(1., 5., 1.5)
4 clust.read_data(data)
5 clust.run_clue()
6 clust.to_csv('./output/',
7 ↪ 'data_results.csv')
```

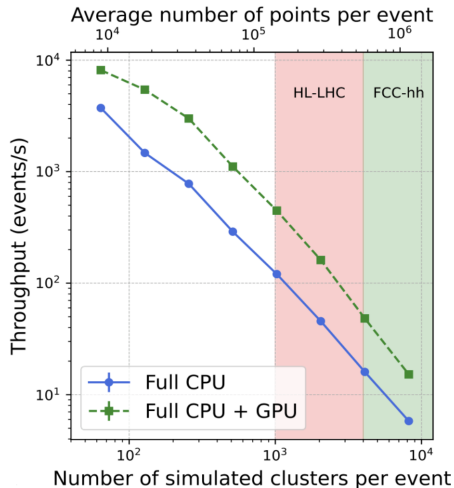
Scaling with dataset size

- ▶ The benchmark on the right shows the scaling of the single-event execution time as a function of the number of simulated clusters, comparing the performance of the CPU and GPU backends
- ▶ The benchmark is performed on a machine with an AMD EPYC 9754 CPU [7] and an NVIDIA Tesla T4 GPU [8].
- ▶ On this machine the GPU backend was 2 orders of magnitude faster than the CPU backend



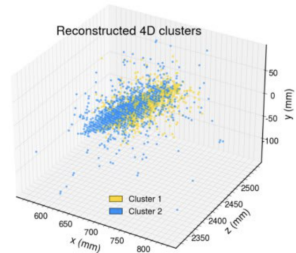
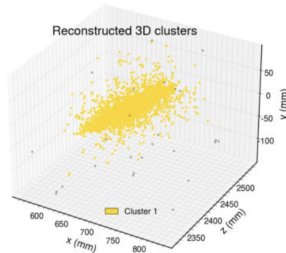
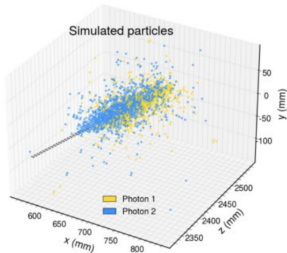
Throughput measures in multithreaded execution

- ▶ Modern HEP software frameworks often are highly multithreaded in order to maximize the number of events processed concurrently
- ▶ The benchmark on the right shows the throughput obtained by fully occupying a machine with an AMD EPYC 9754 CPU and an NVIDIA Tesla T4 GPU, as a function of the number of clusters reconstructed



Ongoing developments

- ▶ CLUE is the default clustering algorithm for HL-LHC reconstruction in the CMS experiment
- ▶ We are constantly trying to improve the library and develop new features
- ▶ We are currently investigating applications in the FCC software in the context of the CERN Experimental Physics R&D program
 - the plot below shows the reconstruction of two simulated particles in a 4D space



Conclusions

- ▶ CLUEstering is a **density-based weighted** clustering library
- ▶ It can be applied to general clustering problems
- ▶ Its use of heterogeneous platforms and its performance portability make it stand out from most other clustering libraries

This work has been funded by the Eric & Wendy Schmidt Fund for Strategic Innovation through the CERN Next Triggers project under grant agreement number SIF-2023-004.

Thanks for the attention!
Questions?



NexTGen
Next Generation Triggers



References

- [1] Marco Rovere et al. “CLUE: A Fast Parallel Clustering Algorithm for High Granularity Calorimeters in High-Energy Physics”. In: *Frontiers in Big Data Volume 3 - 2020* (2020). ISSN: 2624-909X. DOI: 10.3389/fdata.2020.591315.
- [2] URL: <https://gitlab.cern.ch/kalos/CLUEstering>.
- [3] URL: <https://github.com/cms-patatrack/CLUEstering>.
- [4] URL: <https://cms-patatrack.github.io/CLUEstering>.
- [5] Erik Zenker et al. “Alpaka - An Abstraction Library for Parallel Kernel Acceleration”. In: IEEE Computer Society, May 2016. arXiv: 1602.08477. URL: <http://arxiv.org/abs/1602.08477>.
- [6] URL: <https://github.com/alpaka-group/alpaka>.
- [7] URL: <https://www.amd.com/en/products/processors/server/epyc/4th-generation-9004-and-8004-series/amd-epyc-9754.html>.
- [8] URL: <https://www.nvidia.com/en-us/data-center/tesla-t4/>.