International Astronautical Congress, Milan, Italy, 14-18 October 2024.
Copyright ©2024 by Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR). Published by the IAF, with permission and released to the IAF to publish in all forms.

IAC-24-B1-IP-117-x82994

Exploring and processing large data sets in earth observation on HPC-systems with *Heat*

Fabian Hoppe[a]*, **Wadim Koslow**[a], **Kathrin Rack**[a], **Alexander Rüttgers**[a]

[a] *German Aerospace Center (DLR), Institute of Software Technology, High-Performance Computing department, Cologne (Germany)* E-mail: fabian.hoppe@dlr.de, wadim.koslow@dlr.de, kathrin.rack@dlr.de, alexander.ruettgers@dlr.de

* Corresponding author

Abstract

Handling and analyzing massive data sets efficiently is particularly important in the field of earth observation. Nevertheless, this can be challenging, especially for researchers and developers without a background in high-performance computing (HPC). The Python library Heat aims at supporting such researchers and developers by providing general-purpose, memory-distributed and hardware-accelerated array manipulation, data analytics, and machine learning algorithms in Python, targeting the usage by non-experts in HPC. This paper show-cases how Heat can help to facilitate exploration and processing of large amounts of data in earth observation on HPC-systems. This is done on behalf of two ongoing applications of Heat in the context of anomaly detection in remote sensing data of German coastal regions.

Acronyms/Abbreviations

API	Application programming interface
CPU	Central Processing Unit
DMD	Dynamic Mode Decomposition
DLR	German Aerospace Center
EO	Earth observation
GB	Giga Byte (10^9 Bytes)
GPU	Graphics Processing Unit
HPC	High-Performance Computing
LOF	Local Outlier Factor
PCA	Principal Component Analysis
PB	Peta Byte (1000 TB, 10^{15} Bytes)
RAM	Random-Access Memory
SAR	Synthetic Aperture Radar
SVD	Singular Value Decomposition
TB	Tera Byte (1000 GB, 10^{12} Bytes)

Funding: This research was supported by the European Space Agency through the Open Space Innovation Platform (https://ideas.esa.int) as a Early Technology Development Agreement and carried out under the Discovery Program ESA Early Technology Development (Research Agreement No. 4000144045/24/NL/GLC/ov).

1. Introduction

Processing large data sets has become a crucial component in modern science and engineering; this is particularly true for the field of earth observation and remote sensing. The German Satellite Data Archive[1] (D-SDA), e.g., has currently a size of more than 20 PB and is growing by more than 3 PB per year by current and upcoming satellite mission. However, the benefit due to the increasing amount of available data is accompanied by a growing effort required for an adequate analysis. For many scientist, in particular non-experts in HPC, getting the best out of their data may pose a true challenge.

Many scientific data science workflows are built in Python on top of the libraries NumPy, SciPy, and/or scikit-learn, or make use of tools that are based on these libraries. This setup promises fast prototyping and good maintainability due to the simple API, and—due to NumPy's and SciPy's optimized C-kernels in the background—competitive performance on CPU. In the big data regime NumPy's and SciPy's limitation to so-called *shared memory parallelism*, i.e., parallel execution on the cores of a single CPU, can pose a severe restriction: due to the limitation to shared memory parallelism, only a single machine, e.g., a workstation or a single node within a cluster, can be used; this usually limits the available RAM to not much more than 1 TB, even on machines with particularly large

[1] https://www.dlr.de/en/research-and-transfer/research-infrastructure/d-sda-archive-production-oberpfaffenhofen [Accessed August 21, 2024]

Copyright ©2024 by Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR). Published by the IAF, with permission and released to the IAF to publish in all forms.

RAM. Moreover, no advantage can be taken from the huge speed-ups that can be achieved by using modern GPUs, as NumPy and SciPy are bound to CPUs.

Under these premises, the Python library Heat[2] [1, 2] is being developed since 2018 in a collaborative effort of research groups at the German Aerospace Center, Research Center Jülich and Karlsruhe Institute of Technology. The vision behind this library, developed for scientists and by scientists, is to make array computing, data analytics and machine learning as easy on a (GPU-)supercomputer as it is known from NumPy, SciPy, and scikit-learn on a workstation. As related work we mention in particular the libraries Dask[3] [3], which is also well-known in the EO community, and Jax[4] [4]. For an extensive overview on the ecosystem of scalable array computing and/or machine learning in Python we refer to, e.g., [5] or [6].

The remaining part of this paper is structured as follows: in Section 2 we provide the reader with a brief introduction to Heats architecture and main features. In Section 3 we discuss the recent application of Heat in the context of anomaly detection on earth observation data at DLR. A brief summary and outlook in Section 4 concludes the paper.

2. Heat

Heat is a Python library for massively-parallel array computing and machine learning on CPU/GPU-clusters. For a technical description of its architecture and programming model we refer the reader to [2], whereas a more research software engineering-focused exposition can be found in [6]. At this point we will limit ourselves to briefly summarising the most important design principles and features of Heat from [6, Sect. 3]:

- *Multi-node- and GPU-capabilities:* operations can be performed in a multi-node-/multi-GPU-setting (e.g., on several nodes of a GPU-cluster).
- *Simple API and usage:* the simple API mimicks NumPy/SciPy/scikit-learn and allows for rapid prototyping or adaptation of existing NumPy/SciPy/scikit-learn workflows also by HPC-non-experts.
- *Platform independence / Interoperability:* as Heat is mainly based on PyTorch [7] and MPI (via mpi4py [8]) under the hood, it is interoperable, portable, and supports hardware of different vendors (e.g., GPUs by Nvidia *and* AMD).

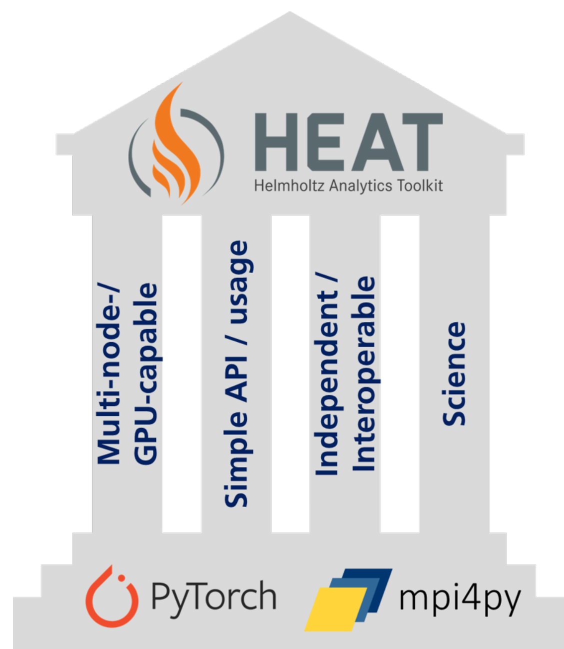


Fig. 1: Heats design principles and main dependencies.

- *Scientific background:* Heat (primarily) targets usage by scientists and offers the opportunity of collaboration/joint publications with users.

The basic data type in Heat is the `DNDarray` class, a distributed-memory- and GPU-capable n-dimensional array, mimicking NumPy's `ndarray` class. For these arrays, several creation, manipulation, and analysis routines, as well as linear algebra operations are available. In addition, a growing amount of classical machine learning algorithms is provided—adapted to a massively parallel setting, of course. Heat allows you to exploit the combined memory of several machines/devices in a compute cluster ("distributed memory parallelism"); consequently, huge data sets can be be handled and processed as a unit, which would not be possible when being restricted to the memory available at a single machine/device ("shared memory parallelism"). Internally, Heat automatically distributes the data as evenly as possible across the available processing resources (CPUs or GPUs). The numerical examples in [2, 6] indicate that this reduces the overall memory footprint and in certain cases also avoids some overhead (w.r.t. runtime) compared to Dask's approach to distribute tasks.

Finally, we note that software quality is enforced by corresponding measures during development (unit tests on different hardware, monitoring of code coverage, code review, etc.).

[2]https://github.com/helmholtz-analytics/heat/
[3]https://www.dask.org [Accessed August 07, 2024]
[4]https://github.com/google/jax [Accessed August 07, 2024]

Copyright ©2024 by Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR). Published by the IAF, with permission and released to the IAF to publish in all forms.

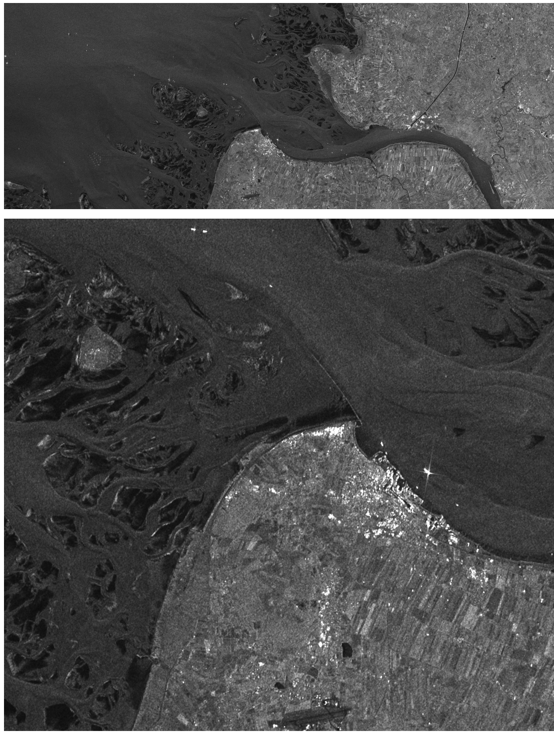


Fig. 2: *Top:* One of the 363 SAR images showing the mouth of the Elbe into the North Sea. *Bottom:* Enlarged section from the centre of the image.

3. Application to use-cases in earth observation

The DLR project RESIKOAST[5] focuses on enhancing the resilience of the North Sea and Baltic Sea coasts against the impacts of climate change. Among the predicted challenges are rising sea levels and an increasing frequency of extreme weather events such as storm surges, storms, and heavy rainfall. To address these threats, the project aims to develop strategies for long-term adaptation and tools for early risk detection. A key aspect hereby is the detection of hotspots, i.e., locations with significant changes or anomalies over time. Detecting these hotspots is crucial for timely interventions and protection of coastal landscapes, populations, and infrastructure.

3.1 Use-case I: massively parallel anomaly detection

In the following we focus on two parts of the entire machine learning pipeline that are both time- and memory-intensive and thus challenging to scale up to the finally intended amount of data. A time series of 363 SAR images[6] (2100×5660 pixels each) is considered as a prototypical example that allows to highlight the challenges associated with scaling up the existing workflows. As an example, one particular image is shown in Figure 2.

Each image of size 2100×5660 pixels contains $11,839,476$ (not necessarily disjoint) patches of shape 7×7 pixels; consequently, given our time series of 363 images, there are $11,839,476$ many time series of 363 such 7×7-patches each. To each of these time series an instance of the so-called local outlier factor (LOF, [9]) algorithm shall be applied, resulting in $11,839,476 \times 363$ LOF-values. Finally, a patch shall be classified as anomalous if, e.g., its LOF-value is more than two standard deviations above the mean over all LOF-values. While the original images have a size of about 17 GB (in single precision float data type), the collection of all patches amounts to roughly 840 GB which makes this a challenging problem, in particular, because sequential execution of $11,839,476$ LOF-instances in a `for`-loop would take prohibitively long.

3.1.1 Implementation of use-case I

Heat allows to implement this in a scalable way with minimal effort. The underlying images can be read from an HDF5-file into a `DNDarray` of shape `(363,2100,5660)`. From this another array of shape `(11839476,363,49)`, containing all the patches, can be easily generated using the `unfold`- and `reshape`-functionality available in Heat. Scaling up an existing PyTorch implementation of LOF is straightforward using Heat's `vmap`-function that works analogous to PyTorch's `vmap`, but in a multi-node, multi-GPU setting. In fact, let

```
def torch_local_outlier_factor(torch_data:
↪  torch.Tensor, n_neighbors: int=10):
    ...
    return lofs
```

be a PyTorch implementation of LOF, i.e., a function that computes the LOF-values `lofs` (a tensor of shape `(n,)`) of the given input tensor `torch_data` of shape `(n,m)`. The number of neighbours to be used in the LOF-algorithm is passed as a keyword argument. Then

```
vmapped_torch_lof =
↪  heat.vmap(torch_local_outlier_factor,
↪  chunk_size=chunksize)
```

yields a callable, `vmapped_torch_lof`, that takes a `DNDarray` of shape `(k,n,m)`, possibly distributed over multiple CPUs or GPUs along axis no. 0, computes the

[5] https://www.dlr.de/en/sc/research-transfer/projects/resikoast [Accessed July 29, 2024]

[6] kindly provided by the DLR Microwaves and Radar Institute

Copyright ©2024 by Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR). Published by the IAF, with permission and released to the IAF to publish in all forms.

LOF-values of each of the k many $n \times m$-data, and returns them as `DNDarray` of shape `(k,n)`. The "vmapped" function has the same keyword argument as the original one, i.e., `n_neighbors`, and the argument `chunk_size` allows to determine how many instances of the function to be vmapped may be performed in parallel on each CPU/GPU. Having computed the LOF-values, one may easily get an overview over them by applying standard statistical functions:

```
mean, std = lofs.mean(), lofs.std()
```

Finally, a boolean array indicating the anomalies can be generated by

```
anomalies = (lofs > mean + 2*std),
```

and the number of anomalies in total, per time, or per patch, can be computed by

```
total = anomalies.sum()
per_time = anomalies.sum(axis=0)
per_location =
↪  anomalies.sum(axis=1).reshape((np1,np2)),
```

where `np1,np2` denote the number of patches in the two images dimensions, respectively. Note that these lines, as well as those for computing statistics and determining the anomalies, are exactly the same as in NumPy. However, since the output of `vmapped_torch_lof` is distributed over several processes and devices, communication between processes and devices is necessary in the background to execute the required operations. Nevertheless, Heat takes care of these aspects internally and hides it behind its simple NumPy-like API.

3.1.2 Results for use-case I

Since our focus is on scalability w.r.t. increasing volumes of data, we performed a so-called weak scaling study on both the GPU- and the CPU-partition of DLRs cluster terrabyte[7], respectively. We increase the the amount of data proportional to the resources used for their processing until the entire data are processed. Consequently, we process roughly $\frac{1}{48}$ of the data (the left 2100×117 range of the images, respectively) on one GPU, $\frac{1}{24}$ of the data (the left 2100×235 range of the images, respectively) on 2

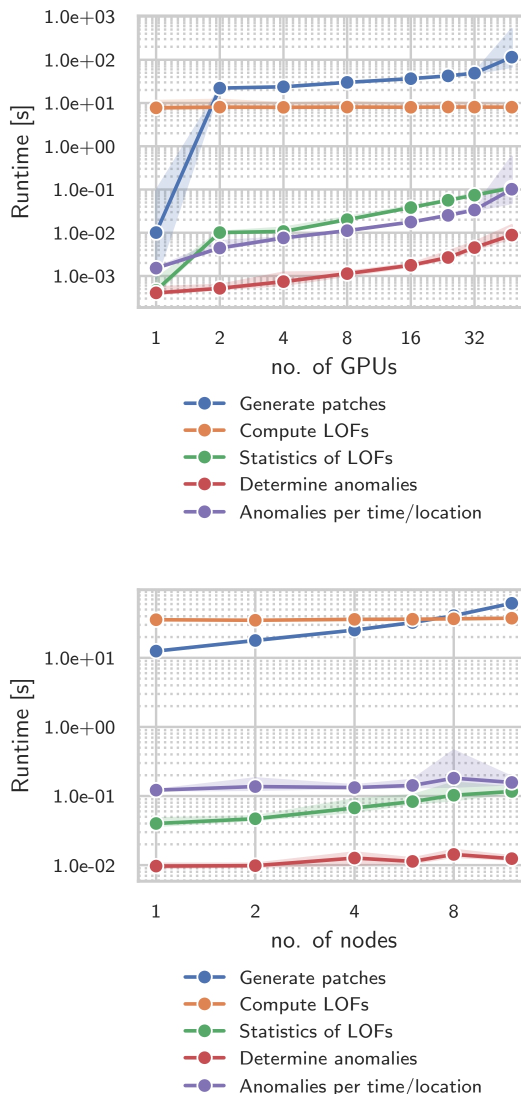


Fig. 3: Run times observed for different functions in use-case I — *Top:* on GPU *Bottom:* on CPU.

[7]The GPU-nodes are equipped with 2 Intel Xeon Gold 6336Y 24 cores 185 W 2.4 GHz, 1024 GB RAM, and 4 Nvidia HGX A100 80 GB 500 W GPUs each, and the CPU-nodes are equipped with 2 Intel Xeon Platinum 8380 40 cores 270 W 2.3 GHz and 1024 GB RAM each. Software loaded as modules: Python 3.10.10, OpenMPI 4.1.5 (Intel compiler 2023.1.0), CUDA 11.8, SLURM 21.08.8-2. Software installed via pip in a virtual environment: Heat 1.5-dev (main branch), NumPy 1.26.3, h5py 3.11.0, PyTorch 2.2.0+cu118, mpi4py 3.1.6, perun 0.6.2.

Copyright ©2024 by Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR). Published by the IAF, with permission and released to the IAF to publish in all forms.

GPUs, and so on, until the full images are processed finally on 48 GPUs (12 GPU-nodes). Similarly, we process $\frac{1}{12}$ of the data, $\frac{1}{6}$ of the data, ... on 1, 2, ... CPU-nodes, until the full data set is processed on 12 CPU-nodes. Runtime and memory consumption are measured using the Python library perun[8] [10].

Figures 3 and 5 (top) show the observations in terms of runtime and memory consumption. Due to the smaller amount of RAM available at GPUs, we required `chunk_size=40000` in `vmap` on GPU, whereas no restriction needed to be imposed on CPU; it can be seen that the maximum RAM consumption per GPU almost touches the available maximum. Nevertheless, computation of the LOFs was significantly faster on GPU than on CPU although on CPU a much larger number of LOFs could be computed simultaneously. LOF computation scales well both on GPUs and CPU; the corresponding run times remain almost constant during our weak scaling study, and memory consumption only increases slightly in the case of GPUs. Statistics and subsequent computations are still faster on GPU than on CPU, but scale much worse on GPU than on CPU; we believe that this is due to the fact that the overhead introduced by communication is (relatively) higher on GPUs because the process-local computations on GPUs are much faster than on CPUs. Finally, one can observe that a—by construction—communication-intensive operation like the generation of patches is comparatively expensive in terms of runtime.

Finally, Figure 6 displays exemplary some of the results obtained during the experiments. The described workflow seems to be able to identify both spatial and temporal "hotspots" of anomalies.

3.2 Use-case II: clustering a huge data set

In this second use-case, the entirety of *all* 7×7-patches over all 363 time steps is considered as a single, huge data set with more than 4 billion elements. To explore the structure of this data set—more precisely: to gain an insight what basic types of patches there are—a clustering algorithm is applied.

3.2.1 Implementation of use-case II

This use-case can be realized with the help of Heat's `BatchParallelKMeans`-class that implements a batch-parallel version of the famous K-Means clustering; see [11] for a detailed description of this algorithm. For simplicity we now load the patches from an HDF5-file in which they

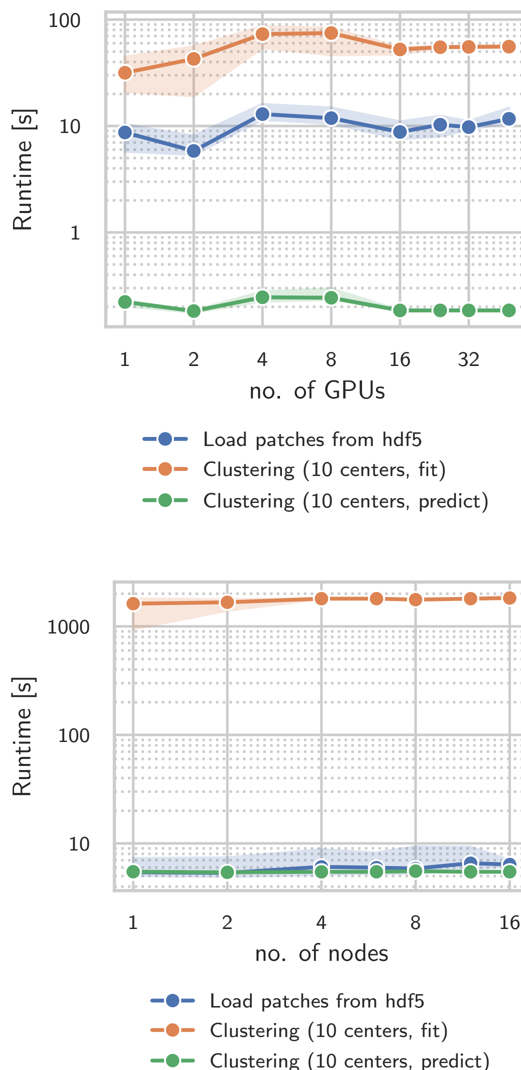


Fig. 4: Run times observed for different functions in use-case II — *Top:* on GPU *Bottom:* on CPU.

[8] https://github.com/Helmholtz-AI-Energy/perun [Accessed August 07, 2024]

Copyright ©2024 by Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR). Published by the IAF, with permission and released to the IAF to publish in all forms.

are stored as an array of shape `(4297729788, 49)`. As pointed out above, this amounts to about 840 GB of data (in single precision); nevertheless, Heat manages the difficulties of parallel I/O for the user. Moreover, Heat's API for machine learning techniques mimicks the one of scikit-learn whenever possible and hence enables rapid prototyping and/or easy transition from serial to scalable code. Hence, the complete code for loading and clustering the data is as follows:

```
import heat as ht
data = ht.load_hdf5("my_file.h5", "patches",
↪  split=0, dtype=ht.float32, device="gpu")
clusterer =
↪  ht.cluster.BatchParallelKMeans(n_clusters=10)
clusterer.fit(data)
labels = clusterer.predict(data)
print(clusterer.functional_value_)
```

Note in particular, that the handling of devices is as simple as known from PyTorch: by specifying `device="gpu"` in an array creation routine, e.g., `load_hdf5`, the respective `DNDarray` will be created on GPUs.

3.2.2 Results for use-case II

Again, we performed a weak scaling study both on CPUs and GPUs. We were able to process the entire set of 4,297,729,788 patches on 16 CPU-nodes in a reasonable amount of time ($\sim$ 30min); judging from the resulting memory consumption of below 200 GB per node, the number of nodes could have been decreased, but then the computing time would have increased in return. We estimate that for the processing of the entire data set about 64 GPUs (16 GPU-nodes), would have been required; however, due to limits on the number of nodes per job, we could only conduct our experiments on up to 48 GPUs (12 GPU-nodes), on which still more than 3 billion patches could be dealt with.

In Figures 4 and 5 (bottom) we show the results for 10 cluster centers, i.e., `n_clusters=10`. It can be seen that all involved operations—loading of the patches from file, determining the cluster centers ("fit"), assigning the data points to their respective clusters ("predict")—scale well, both in terms of memory consumption and runtime. The operations underlying the clustering algorithm allow GPUs to demonstrate their advantages compared to CPUs: on 12 nodes (and, consequently, 75% of the entire data set) we observe a speed-up of roughly factor 40. Only loading from file to GPU is slower than loading to CPU as in the first case an additional data transfer from CPU to GPU is necessary.

Having at hand a scalable clustering routine, an in-

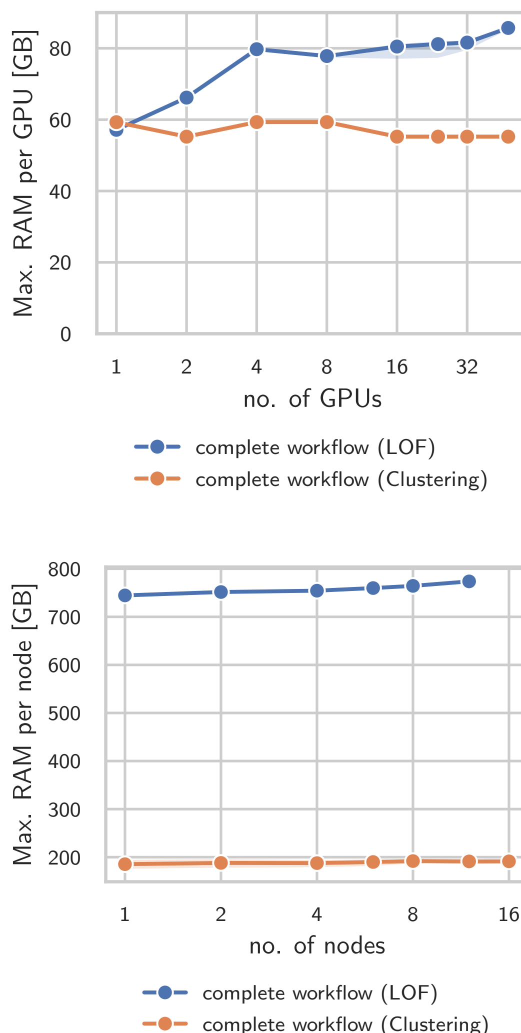
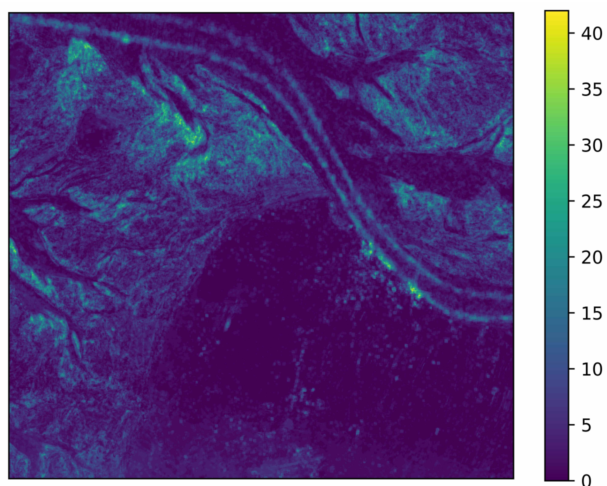


Fig. 5: Memory consumption observed for the entire two use-cases — *Top:* on GPU *Bottom:* on CPU.

Copyright ©2024 by Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR). Published by the IAF, with permission and released to the IAF to publish in all forms.



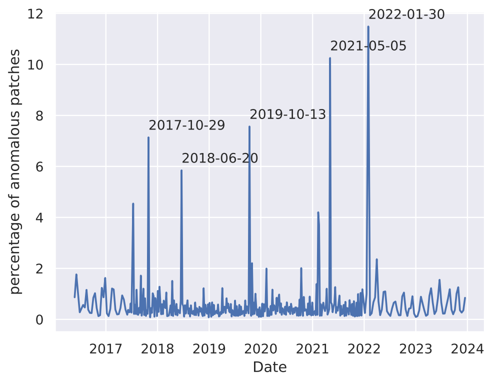
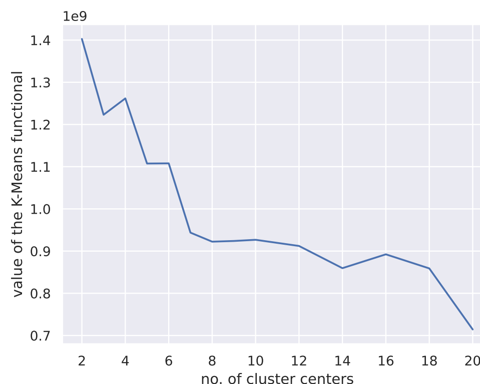


Fig. 6: *Top:* Number of anomalies detected for every patch over time on the enlarged section from Figure 2. *Bottom:* Amount of patches of the entire image classified as anomalous, plotted over time.



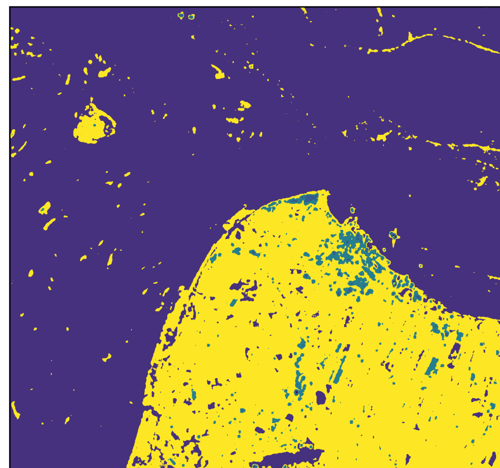


Fig. 7: *Top:* Values of the K-Means functional for different numbers of cluster centers. The "elbow point" is reached at 8 cluster centers. *Bottom:* Cluster labels (8 centers) for every patch on the enlarged section from Figure 2. At this time step and location, only 3 different clusters are visible (colored in blue, yellow, and green).

Copyright ©2024 by Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR). Published by the IAF, with permission and released to the IAF to publish in all forms.

formed choice of the number of cluster centers, becomes feasible, e.g., by repeating clustering for a varying number of cluster centers and choosing the so-called "elbow point"; see Figure 7 (top). An example of how the final assignment to different clusters may look like is provided in Figure 7 (bottom).

4. Summary and Outlook

In this paper we have demonstrated that the Python library Heat allows to scale up data science workflows to large amounts of data in a straightforward and transparent way. We exemplarily considered anomaly detection and clustering on up to 840 GB of data and conducted numerical experiments on up to 16 CPU-nodes or 12 GPU-nodes (48 GPUs) in a compute cluster. The presented example are the first results of the recently initiated efforts to make various analysis pipelines developed for the RESIKOAST project scalable. In the future, our so far prototypical implementations will be further adapted and finally fully integrated in the existing project code base.

At the same time, Heat's capabilities for analysing and predicting time-dependent processes are being expanded as part of the ESA-funded project ESAPCA[9] by implementing a scalable dynamic mode decomposition (DMD, [12]). DMD can roughly be viewed as kind of principal component analysis (PCA) for dynamical systems, and similar as PCA it is based on singular value decomposition (SVD). Thus, the development of a scalable, GPU-accelerated implementation of SVD as a backend for the high-level algorithms like PCA and DMD forms the algorithmic core of the project; this is illustrated in Figure 8.

Another problem related to data originating from satellite observations will serve as prototypical application for scalable DMD: the analysis of long-term thermospheric density data with the goal to enable in data-driven predictions. However, in the course of the project, the context of change detection of coastal areas described here could provide another exciting use case for this upcoming implementation.

Acknowledgements

The SAR images have been kindly provided by Paola Rozzoli and Luca Dell'Amore from the DLR Microwaves and Radar Institute, Satellite-SAR-Systems Department. The authors gratefully acknowledge the computational and data resources provided through the joint high-performance data analytics (HPDA) project *terrabyte* of the German Aerospace Center (DLR) and the Leibniz Supercomputing Center (LRZ). Moreover, the authors thank their student assistant Fynn Osterfeld for implementing the `unfold`-function in Heat.

Development of Heat at DLR is partially carried out under a programme of, and funded by, the European Space Agency. This funding is gratefully acknowledged. *Disclaimer:* The view expressed in this publication can in no way be taken to reflect the official opinion of the European Space Agency.

[9] https://activities.esa.int/index.php/4000144045 [Accessed August 08, 2024]

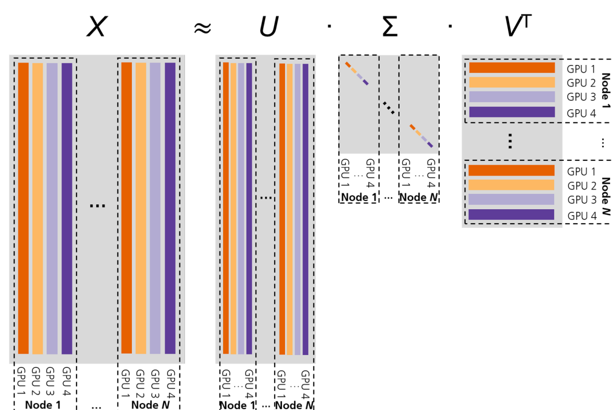


Fig. 8: Schematic representation of SVD in a distributed setting: a matrix $X \in \mathbb{R}^{m\times n}$ is decomposed into the product of two orthogonal matrices $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{n\times r}$ and a diagonal matrix with positive entries $\Sigma \in \mathbb{R}^{r\times r}$. All matrices are split into groups of columns that are distributed onto several GPUs on N nodes of a cluster.

References

[1] Claudia Comito et al. Heat, 2018-2024. `doi:10.5281/zenodo.2531472`.

[2] Markus Götz, Charlotte Debus, Daniel Coquelin, Kai Krajsek, Claudia Comito, Philipp Knechtges, Björn Hagemeier, Michael Tarnawa, Simon Hanselmann, Martin Siggel, Achim Basermann, and Achim Streit. HeAT – a Distributed and GPU-accelerated Tensor Framework for Data Analytics. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 276–287, 2020. `doi:10.1109/BigData50022.2020.9378050`.

[3] Matthew Rocklin. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In Kathryn Huff and James Bergstra, editors, *Proceedings of the 14th Python in*

Copyright ©2024 by Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR). Published by the IAF, with permission and released to the IAF to publish in all forms.

Science Conference (SciPy 2015), pages 130–136, 2015. URL: https://conference.scipy.org/proceedings/scipy2015/pdfs/matthew_rocklin.pdf.

[4] Roy Frostig, Matthew Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. 2018. URL: https://mlsys.org/Conferences/doc/2018/146.pdf.

[5] Oscar Castro, Pierrick Bruneau, Jean-Sébastien Sottet, and Dario Torregrossa. Landscape of High-Performance Python to Develop Data Science and Machine Learning Applications. *ACM Comput. Surv.*, 56(3), oct 2023. doi:10.1145/3617588.

[6] Fabian Hoppe, Juan Pedro Gutiérrez Hermosillo Muriedas, Michael Tarnawa, Philipp Knechtges, Björn Hagemeier, Kai Krajsek, Alexander Rüttgers, Markus Götz, and Claudia Comito. Engineering a large-scale data analytics and array computing library for research: Heat, 2024. Preprint. Submitted to the post-proceedings of the deRSE conference at ECEASST.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[8] Lisandro Dalcin and Yao-Lung L. Fang. mpi4py: Status update after 12 years of development. *Computing in Science & Engineering*, 23(4):47–54, 2021. doi:10.1109/MCSE.2021.3083216.

[9] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, may 2000. doi:10.1145/335191.335388.

[10] Juan Pedro Gutiérrez Hermosillo Muriedas, Katharina Flügel, Charlotte Debus, Holger Obermaier, Achim Streit, and Markus Götz. perun: Benchmarking energy consumption of high-performance computing applications. In José Cano, Marios D. Dikaiakos, George A. Papadopoulos, Miquel Pericàs, and Rizos Sakellariou, editors, *Euro-Par 2023: Parallel Processing*, pages 17–31, Cham, 2023. Springer Nature Switzerland.

[11] Rasim M. Alguliyev, Ramiz M. Aliguliyev, and Lyudmila V. Sukhostat. Parallel batch k-means for Big data clustering. *Computers & Industrial Engineering*, 152:107023, 2021. doi:10.1016/j.cie.2020.107023.

[12] Peter J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010. doi:10.1017/S0022112010001217.