Using HeAT for High-Perfomance Clustering of Remote-Sensing Data

Sebastian Markgraf
Student Computer Science
Karlsruhe Institute for Technology
Karlsruhe, Germany
sebastian.markgraf@student.kit.edu

Charlotte Debus
Institute for Software Technology (SC)
German Aerospace Center (DLR)
Cologne, Germany
charlotte.debus@dlr.de

***Abstract*—Using unsupervised learning for datasets allows to extract information without the tedious task of labeling. Especially, on high-volume datasets this provides a potential time and cost saving benefit.**

For processing high-volume datasets the power of a single machine is not sufficient. By distributing the work across the memory and compute power of multiple machines, it is possible to load bigger datasets and speedup the computation. But due to the difficulty of writing distributed tasks this should be encapsulated into libraries that transparently split the data and work.

In this work we try to use the HPC library HeAT on remote sensing data to find a clustering without using the labels. The result of the clustering is then compared with the labels. Additionally, the scaling behavior of HeAT depending on the number of computing nodes is profiled.

I. Introduction

For quite some time the computing power is growing exponentially [1], but the datasets are growing extremely fast as well. Collecting data is easier nowadays, especially through the vast amount of online services. Although, this allows machine learning algorithms to be more and more precise, there are downsides arising as well with the increased size [2].

To store the datasets the storage capacity needs to expand by the same amount. More importantly, the computing power needs to increase as well. To optimally leverage the increase in computing power efficient algorithms are developed. Especially, when using Graphical Processing Units (GPUs) this is not an easy task. To help with the task many libraries were created e.g. PyTorch. PyTorch provides efficiently implemented mathematical computations and leverages a GPU if available.

Despite these improvements, single machines are reaching their limit when working with huge datasets. The solution lies in using multiple compute nodes. But, while existing libraries already provide excellent single node support, they do not implement distribution of the computation and data on distributed systems. This leaves the developer with the task of managing different nodes and distributing the data correctly on the compute nodes.

The Helmholtz Analytics Toolkit (HeAT) is a library developed by the Helmholtz Analytics Framework, a subset of the Helmholtz Community, which tries to solve this problem. Providing a Numpy like interface HeAT combines PyTorch with the Message-Passing Interface (MPI) to enable the developer to write his script on a single computer and execute it on a distributed compute cluster without worrying about the correct synchronization.

Another problem arising is the tedious work of labelling the dataset. To use datasets for classification or other machine learning purposes instance labels are needed. These labels need to be precise and are often created manually by experts. Due to the sheer amount of data that needs to be labelled, this process is cost and time intensive.

Unsupervised learning is the category of machine learning algorithms that do not use labels. These algorithms try to use the inherent structure of the dataset itself to find useful information and insights. A well-known family of unsupervised algorithms are the clustering algorithms. Clustering is an unsupervised pendant to the supervised classification and splits the dataset in different clusters according to its features. When combined with expert knowledge this could allow to classify datasets without labelling them first.

When combining the benefits of unsupervised clustering with the benefits of using HeAT, it allows to use huge datasets without the tedious work of labelling and with minimal development effort. The goal of this paper is to evaluate the approach of using HeAT on remote sensing data which can only be labelled by experts with a lot of manual effort.

II. Fundamentals

A. Distributed Array Computation

Many scientific fields use Numpy as a basis for their calculation. Due to its implementation in the C language and its well designed Python interface, the computation is fast and easy to write. But, Numpy arrays are only implemented for single node use.

Multiple libraries which allow the use of Numpy computations on multiple nodes have emerged: [4], [5], [6]. Noticeably, [4] adds GPU support to speedup specific operations e.g. big matrix calculations.

The hardest part in distributed array computing lies in the distribution of the data. All mentioned libraries provide either the option to specify how the data should be distributed or even try to estimate the best distribution [5]. All of them have a commonality in encapsulating and hiding the distribution of the data itself from the user. This allows ease of use, but

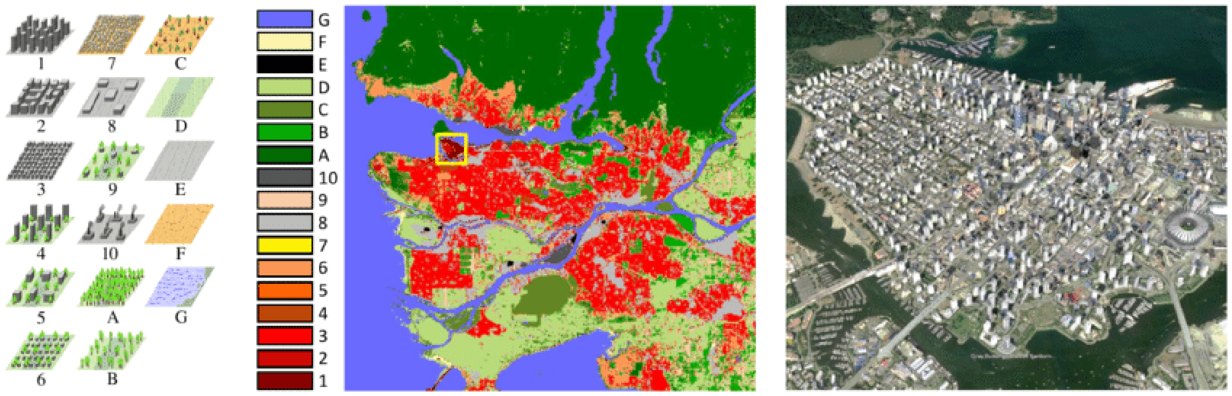

Figure 1: The LCZ classes as schematics on the left and matching of classes for a city in the centre. Image taken from [3].

can lead to drops in performance when the user executes non optimal operations.

B. MPI

MPI is a message-passing standard published from the Message-Passing Interface Forum. It is currently available in version 3.1 [7]. "The goal of the Message-Passing-Interface [...] is to develop a widely used standard for writing message-passing programs"[7]. MPI is today widely used in High-Performance Computing (HPC), especially in Grid Computing. The standard interface allows to pass results or data between different processes or nodes. Due to MPI being a standard multiple implementations exist. Not all of the implementations are available for all systems and support all versions of the standard. OpenMPI [8] and Intel MPI [9] can be cited as examples, which are used in this work. MPI has some limitations that should be remembered when using it with big datasets. The most important limitation is the maximum message size, due to the size being specified as an integer. Therefore, the maximum number of values that can be passed is $2^{31}-1$. Bigger messages need to be split in multiple messages or workarounds need to be implemented.

C. HeAT

HeAT, introduced in [6], is a mathematical library for distributed computations. It conforms to a Numpy [10] interface and makes all distributed calculations transparent. After executing HeAT scripts single threaded during development, executing them parallel on a cluster is done by executing the same script with MPI. When used correctly, no changes to the code itself are necessary. This is due to HeAT wrapping PyTorch as implementation with the correct MPI synchronization to distribute the calculation.

To correctly perform the distributed calculation HeAT adds a new data structure: the Distributed N-Dimensional array (DNDarray). For a user accustomed to other libraries, the HeAT API feels familiar, as it employs the general Numpy and PyTorch tensor syntax but introduces a split parameter. The split parameter allows to control the axis, which the data is distributed across. The developers decided to allow only one split axis to keep the development effort manageable.

Using the split parameter to control the distribution has some pitfalls for developers. When reshaping or splicing the dataset one has to potentially rebalance the dataset. Additionally, this could incur communication effort to redistribute the dataset across the nodes. Other unexpected behavior especially for developers already having distributed computing experience is the need of all processes to call all HeAT methods. While in traditional MPI one discriminates between the processes based on the rank, in HeAT all nodes need to access all functions and DNDarrays or the program could lockup.

Furthermore, HeAT provides the load utility for distributed loading of datasets. Through the usage of all nodes at the same time Input-Output (IO) is maximized and the data is splitted already during loading. Therefore, calculations can start right away without the need to distribute the data [6].

D. Local Climate Zone Classification

Local Climate Zones (LCZs) are a scheme formally proposed by Stewart and Oke in [11]. The scheme classifies areas based on fabric, land cover, structure and metabolism into one of 17 LCZs [12]. The LCZs are decided by 4 components:

1) Height of roughness features
2) Packing of roughness features
3) Surface cover around roughness features
4) Thermal admittance of materials

The different classes can be seen in Fig. 1. Suburban areas are complex and diverse which makes temperature and climate analysis hard to specify, therefore LCZs were introduced to get more explainable and understandable divisions of those areas. According to Stewart and Oke, the system fulfills all criteria of a logical local system as defined in [13]. Their work adds guidelines on how to classify an area correctly into their zones and how to use them afterwards. These guidelines allow to keep a consistent labelling and create datasets with these labels.

E. SO2Sat

Zhu et al. presented the SO2Sat dataset in [3]. They describe SO2Sat as a "valuable benchmark dataset [...], which consists of LCZ labels of about half a million [...] image patches". These images from the 42 different main cities are taken from the European Space Agency (ESA) satellites Sentinel-1 and Sentinel-2. Each image is 32 by 32 pixels and contains 8 channels for Sentinel-1 and 10 channels for Sentinel-2. After preprocessing these images they were given to domain experts for labelling which followed a "carefully designed labelling work flow" [3]. Through the careful work the dataset achieved a "overall confidence of 85%" [3]. The labels consist of integers representing the 17 different LCZs.

F. Spectral Clustering

"Spectral clustering is the process of partitioning data samples into k groups based on graph theory" [6]. Therefore, to derive the formulation of Spectral Clustering we introduce the undirected graph $G = (V, E)$ with nodes V and edges E. We consider the graph as weighted by assigning each edge a weight w_{ij}. As the graph is undirected, the weight is identical for $w_{ij} = w_{ji}$. We define the degree of a vertex $v_i \in V$ by iterating over all vertices $v_j \in V$ that are connected to v_i. Therefore, the weight of the edge is positive:

$$d_i = \sum_{j=1}^{n} w_{ij}$$

Using the preceding definition, we set the degree matrix D as the diagonal matrix with the degrees $d_1, \ldots, d_n$ on the diagonal. [14] For two subsets of indices A, B we define the weights matrix [14]:

$$W(A, B) := \sum_{i \in A, j \in B} w_{ij}$$

The core element of spectral clustering is a similarity graph describing the similarity between different elements of the dataset. There are several possible ways to construct a similarity graph fulfilling the mentioned criteria of an undirected graph. The following three are used often for spectral clustering:

- fully connected
- k-nearest neighbor
- ϵ-neighborhood

The fully connected graph considers the similarity between all samples. The graph should model the local neighbors and is therefore only useful when the distance function models the width of the neighborhood. The k-nearest neighbor graph can be implemented in two ways. If we just use the definition of the k-nearest neighbors we would receive a directed graph. To convert this into an undirected graph we can decide between connecting both vertices if one of them is a k-nearest neighbor of the other or connecting them if both vertices are nearest neighbors of each other. The ϵ-neighborhood graph only connects vertices whose similarity is smaller than ϵ. Since this option already models the neighborhood directly, it usually is considered as an unweighted graph [14]. According to [14] there is no theoretical work on the choosing of one method over another.

```
Data: Similariy matrix $S$, number $k$ of clusters
Result: Clusters $A_1, \ldots, A_k$
1 Construct a similarity graph;
2 $W \leftarrow$ weighted adjacency matrix;
3 $L \leftarrow$ computeLaplacian($W$);
4 $u_1, \ldots, u_k \leftarrow$ computeEigenvectors(L);
5 $U \leftarrow$ Matrix with columns $u_1, \ldots, u_k$;
6 foreach $i = 1, \ldots, n$ do
7     $y_i \leftarrow$ vector corresponding to the $i$-th row of $U$
8 end
9 $C_1, \ldots, C_k \leftarrow$ kMeans($y_1, \ldots, y_n$);
```

Algorithm 1: Basic Spectral Clustering

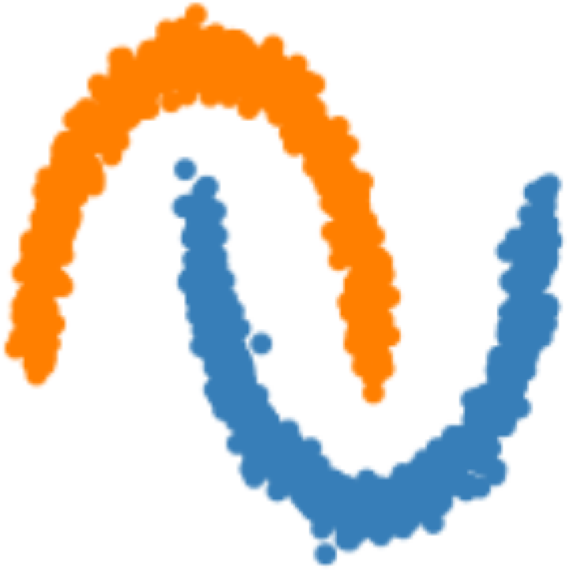

Figure 2: 2 Half-Moon example for clustering performance of spectral clustering. Taken from [16].

Using the constructed similarity graph we need to calculate the spectrum of the similarity matrix. This is done by constructing the normalized symmetric laplacian defined via

$$L_{sym} = I - D^{-1/2} W D^{-1/2}$$

using W, D as defined earlier.

The Lanczos algorithm [15] is used to calculate the eigenvalues and eigenvectors from the laplacian. When using the k smallest eigenvalues and the corresponding eigenvectors $e_1, \ldots, e_k$ we can embed the dataset.

Now a clustering can be performed in the smaller embedding space of the eigenbasis using another algorithm, e.g. KMeans. The overall algorithm can be found in algorithm 1.

One of the most common examples for spectral clustering can be seen in Fig. 2 [16]. Spectral clustering is able to separate the nonlinear datasets. This is often compared with KMeans which fails to correctly recognize the different clusters when using the euclidean distance.

G. Distance Metrics

For spectral clustering the distance between samples needs to be measured. Depending on the data different ways to measure the distance are implemented in HeAT.

1) Euclidean Distance: The euclidean distance is a well known way to measure the distance. The euclidean distance of two points $X = (x_1, \ldots, x_n), X' = (x'_1, \ldots, x'_n)$ is commonly defined as:

$$d\left(X, X'\right) = \sqrt{\left(x'_1 - x_1\right)^2 + \ldots + \left(x'_n - x_n\right)^2}$$

It is fast and easy to calculate, but it is limited to linear separation with most algorithms.

2) Radial Basis Function Kernel: Radial Basis Functions (RBFs) are a kernel that wraps the euclidean distance d [17]:

$$RBF\left(X, X'\right) = \exp(-\gamma d(X, X'))$$

Where γ is a free parameter. An equivalent definition uses $\gamma = \frac{1}{2\sigma^2}$, with a free parameter σ.

The RBFs are a measure of similarity and augment the effects of the euclidean distance due to the exponential function. They allow to separate non-linear data with many algorithms.

H. Clustering Evaluation Metrics

Evaluating clusters is more complicated than evaluating classification or other supervised machine learning tasks. Unsupervised tasks can be evaluated using internal or external metrics. While internal metrics, e.g. cluster purity, are derived from the result of the algorithm itself, external metrics are calculated from the comparison with given labels. Since the in Section II-E explained dataset contains labels, this section is going to focus on external metrics.

1) Adjusted Rand Index: For the here presented metric we define C as the ground truth classes and K as the clustering assignment. On this basis we define the following:
a - the number of pairs of elements that are in the same set in C and in the same set in K
b - the number of pairs of elements that are in different sets in C and in different sets in K
Then the Rand Index (RI) is mathematically given by [16] as

$$RI = \frac{a + b}{C_2^{n_{samples}}}$$

When adjusting this index for random assignments we yield:

$$ARI = \frac{RI - \mathbb{E}[RI]}{\max(RI) - \mathbb{E}[RI]}$$

Both the Adjusted Rand Index (ARI) and the RI measure the similarity of two assignments. While the RI has problems with random assignments, the ARI assigns a score close to 0 to random assignments.

2) V-Measure: The V-Measure was introduced by Rosenberg and Hirschberg in [18] as an external clustering evaluation metric based on entropy. The measurement is computed as a combination of homogenity and completeness. This allows the measurement to be explained more easily by logging homogenity and completeness additionally. Homogenity measures if only data points of a single class are assigned to a single cluster. Completeness is defined symmetrically to homogenity by measuring if all data points of a single class are assigned to a single cluster. When combining these two with a parameter β we get the V-Measure [16]:

$$v = \frac{(1 + \beta) \times homogenity \times completeness}{\beta \times homogenity + completeness}$$

Depending on the value of β we put more weight on homogenity or completeness. The usual value of β is 1.0 which weights both components equally. The measurement is symmetric and bounded between 0 and 1, the later being the perfect achievable score.

III. Implementation

A. Dataset Loading

The dataset is available as two Hierarchical Data Format version 5 (HDF5) files, representing a training - validation split. Both parts of the dataset contain a part of the the Sentinel-1 and Sentinel-2 image patches as well as the labels corresponding to these images. To load them with HeAT the HDF5 support needs to be installed. The training subset contains around 350 000 samples. The implementation can be configured to load the Sentinel-1 or Sentinel-2 dataset and specify a percentage of the dataset which should be loaded. This enables easier testing and profiling of the scaling behavior.

B. Feature Reshaping

When loading the dataset the image patches are available in the shape of

$$(num\_samples, channels, width, height)$$

To use these in the clustering we need a feature vector for each sample instead of the nested information. Therefore, we aim for the shape

$$(num\_samples, feature\_dim)$$

Before reshaping, the data is already split into batches of samples onto different nodes. Therefore, when flattening the channels of a sample, no communication should be incurred. This proved to be more difficult than anticipated. The Numpy reshape function allows to specify -1 for a dimension that should be automatically calculated. HeAT did not have this option at the beginning of this work. Additionally, reshape was not optimized for operations that could occur locally and always performed communication even if it was not necessary in this case. After communicating both of these wishes, the HeAT team provided support for the -1 notation and implemented a fix for the additional communication. For a workaround during the development time, the implementation used a custom version of `ht.flatten` to perform the necessary operation.

C. Normalization

Due to the channels being in different formats, it can be useful to normalize all channels. Therefore, channelwise z-score normalization is implemented [19]:

$$Z = \frac{X - \mu}{\sigma}$$

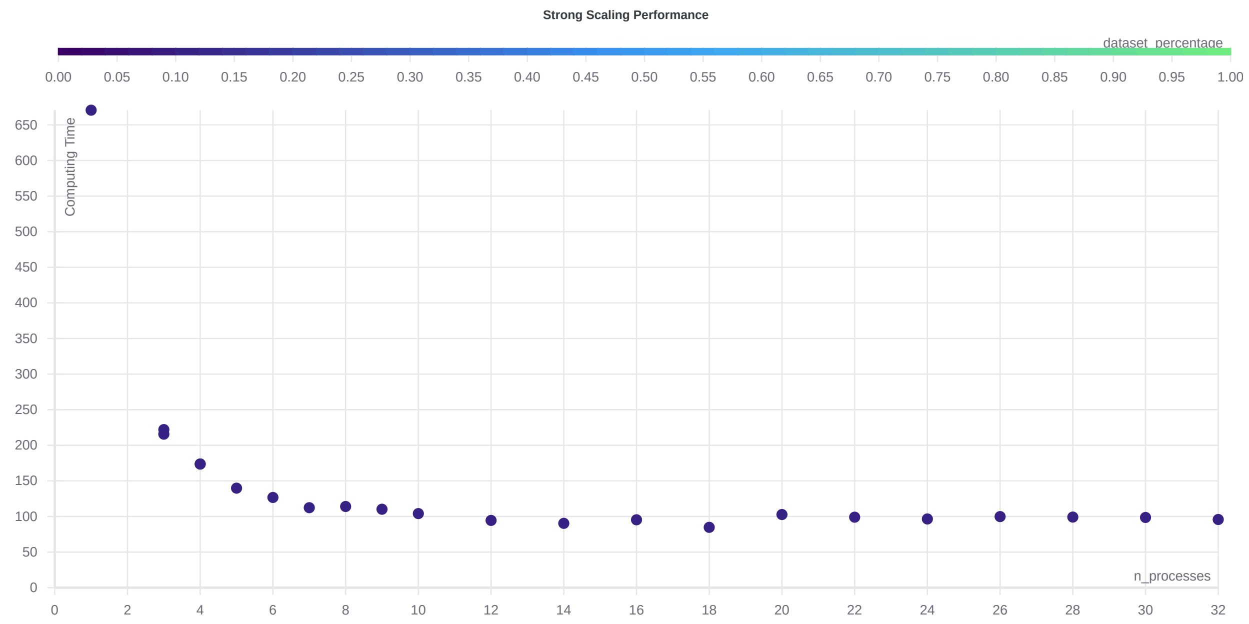

Figure 3: Strong scaling behavior of the spectral clustering algorithm.

with μ, σ denoting the channel mean and standard deviation of the whole dataset. Especially when using small datasets, σ could be potentially 0 which leads to division by zero errors. Therefore, the normalization can be deactivated for time profiling.

D. Logging

For the whole project the default Python logging [20] is used. All important steps print their beginning and their end on INFO level. More specific information, especially about the shapes of the data after transformation is output to DEBUG. Per default a console handler is attached that outputs all logging of all nodes to STDOUT. HeAT does not implement standard library logging and therefore internal steps of the algorithms are not visible on the log files. For more information about the usage of Python logging refer to [20].

E. Experiment Tracking

All runs are tracked with Weights And Biases (WanDB) [21]. Integrating WanDB with the script was difficult due to the fact that the tracking is not built for distributed systems. Therefore, only the root node should perform logging to the endpoint. Thus, only the root node loads the configuration from the disk and broadcasts the current configuration to all other nodes. Additionally, all logging methods are wrapped in a decorator, that only allows the root node to execute the function. Due to the configuration being tracked as well, it is easily possible to sort all runs and filter if they crashed or compare runs with similar configuration.

One has to be careful when the script crashes due to MPI errors. Since MPI errors are thrown below the Python level and the tracking runs on the Python level, there are no signs of an error when looking only at the tracking. This is caused by the design and cannot be circumvented. It is advisable to check runs regularly on the cluster or running machine itself.

F. Timing

Only the fitting of the spectral clustering algorithm is timed. This is done by wrapping the fit call with a start and end time. The prediction of labels for all elements and any plotting is performed after the time measurement. Every run executes the algorithm multiple times to limit the influence of start up and loading effects.

G. bwUniCluster

The bwUniCluster 2.0 was commissioned on the 17.03.2020 by the Steinbuch Centre for Computing (SCC) [22]. It is part of the framework of the Baden-Württemberg Implementation Concept for High Performance Computing (bwHPC). In total the cluster has 848 nodes. The fat and high throughput compute nodes are connected via InifiBand HDR100, which has a throughput of 100 Gbit/s [22]. The connection to the storage is done via InifiBand EDR [22].

Scheduling of jobs on the system is done with Simple Linux Utility for Resource Management (SLURM) [23]. SLURM allows to specify the required resources for a job and schedules it for an available time slot. The specification is mainly done through partitions which group a specific type of node, e.g. fat nodes, multiple nodes, GPU nodes. Depending on the chosen partition the user can configure different requirements regarding the number of nodes, memory and needed time. This work uses the multiple partition for most tasks and the fat partition for running jobs on single nodes.

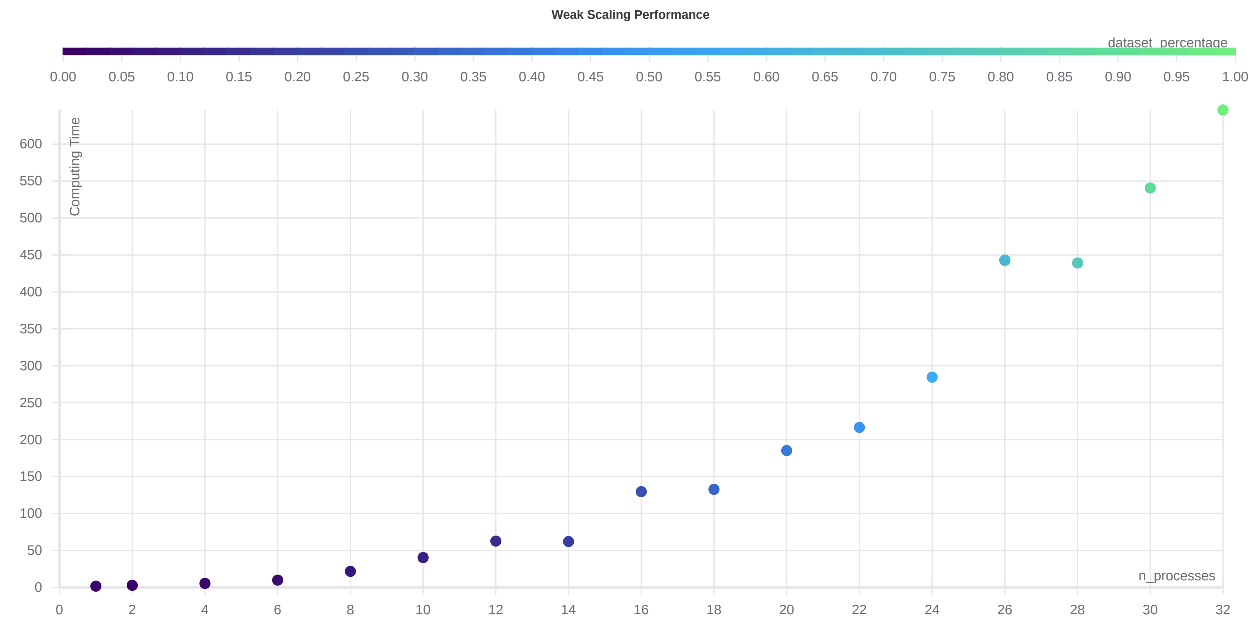

Figure 4: Weak scaling behavior of the spectral clustering algorithm.

IV. Evaluation

A. Strong Scaling

The first evaluation consisted in the strong scaling behavior of the HeAT implementation. For this the same percentage of the dataset was used for all numbers of nodes. The configuration was set to fixed 10% of the Sentinel-1 dataset. For the graph calculation the fully connected graph was chosen. For an explanation of the graph type see Section II-F. Each node is assigned 90GB of Random Access Memory (RAM) and run in exclusive mode to keep the run times clean of the influence of other running programs. All nodes run only one MPI process and use the PyTorch threading.

The results of the measurement can be seen in Fig. 3. As a baseline the same task was performed on one node and yielded a runtime of $680s$. No run for two nodes could be run due to the message size limitation of MPI. This is one of the current weak points of HeAT that the HeAT plans to fix in the future.

Unexpectedly, the first step from one to three nodes performs good, although the communication effort is added. This could be due to HeAT not being optimized for single node performance. The scaling continues until around ten to twelve nodes. From here on the computing power increases but the overhead of communication keeps the runtime consistent. This threshold probably lies higher with bigger datasets but due to the earlier mentioned MPI message limit it is not possible to test bigger datasets with small numbers of nodes.

B. Weak Scaling

For testing of the weak scaling the load per node should stay constant. Therefore, we fixed the maximum number of nodes to 32. The maximum number of nodes work on the full dataset. All smaller number of nodes work on the corresponding percentage of the dataset in relation to the maximum number of nodes. When calculating the percentage, the runtime of the chosen algorithm has to be considered. Since spectral clustering is in the runtime of $O(n^2)$ with n samples, we need to scale the fraction of the dataset squared as well:

$$percentage\_dataset = \left(\frac{nodes}{nodes_{max}}\right)^2$$

All other parameters stayed the same as in the strong scaling case. The results of this profiling can be seen in Fig. 4.

In contrast to the strong scaling in Section IV-A, the weak scaling is not optimal. If the algorithm would scale perfectly, the runtime should be a constant line, since the load stayed constant across all nodes. But the runtime scale could be explained as $O(n \cdot \text{num}_{processes})$. This is probably due to the communication overhead when calculating the similarity matrix. Since both the overall amount of data and the amount of nodes, which need to send the data between them, the scaling of the algorithm could have been expected.

Metric	ARI	V-Measure
Training Validation	-0.00005692	0.0001602

Table I: The calculated metrics of the clustering result.

C. Clustering Results

Although, the main focus of this work is the scaling behavior of the implementation the results will be evaluated as well. To evaluate the results external metrics for evaluating clustering results were presented in Section II-H. They are calculated for the whole dataset and shown in Table I. All metrics are close to 0 which indicates a random distribution of the labels with no correlation to the target labels.

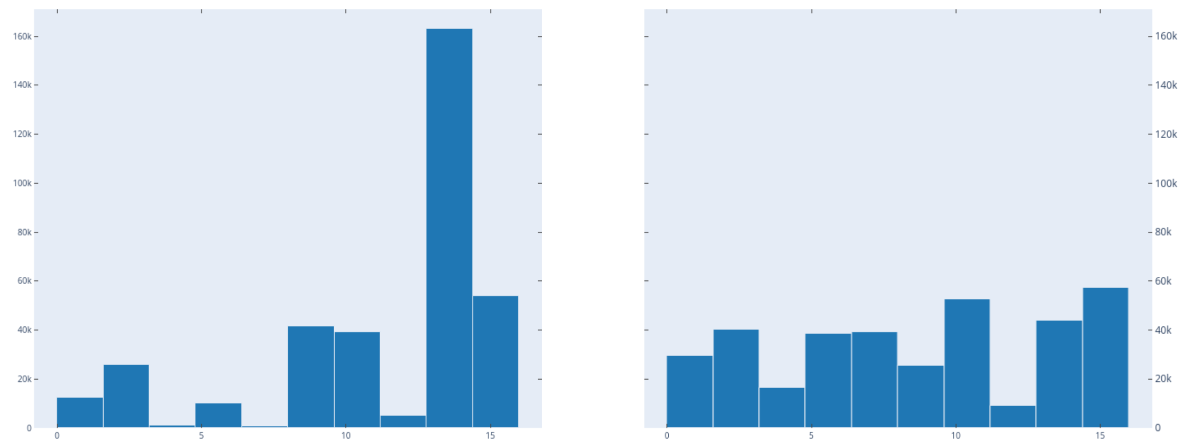

Figure 5: Comparison of the label distribution with the original label distribution.

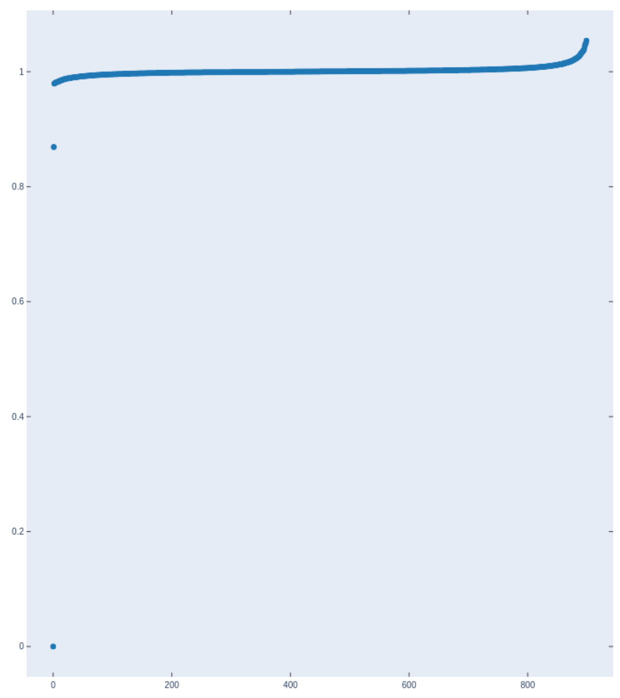

Figure 6: Sorted eigenvalues of a run with 300 lanczos iterations.

To confirm this, a plot of the predicted label distribution can be seen in Fig. 5. The labels do not match in any counts to the original labels. Therefore, they do not line up with the external labels in the pairwise metrics.

The poor performance of the clustering can be due to a variety of reasons. First, the LCZ have been designed on paper and were not extracted from data. Therefore, it is unlikely that the zones would all line up with any structure in the data itself. But, especially the land types could have been expected to line up with the zone specification.

When taking a look at the eigenvalue spectrum in Fig. 6 most of the eigenvalues seem very close together and further explain the poor performance. Overall, the eigenvalue spectrum is highly dependent on the chosen hyperparameters. Especially, when choosing gamma to high, the lanczos iterations can result in numerical instabilities. Therefore, one could observe negative eigenvalues in this case, despite the laplacian being semi-positive definite.

Furthermore, spectral clustering has problems with high-dimensional data. This could be changed by using another approach, a specific version of spectral clustering or do feature extraction before clustering. Possible features could be channel-wise mean, deviation, maximum and minimum.

D. Problems When Using HeAT

During the implementation of this project and the runs many problems arose. One of the problems was in the $reshape$ function of HeAT. When reshaping the data distributed, but keeping the split axis untouched, the implementation still called `MPI_AllToAllv`. If the function is only performed locally on each node operations with huge datasets can be successful, but the call to `MPI_AllToAllv` exceeds the maximum length for messages and therefore crashes the program. This error is already fixed in the experimental build.

Another important factor which was found while testing different configurations are the different implementations of MPI. The first configuration used the Intel implementation of MPI. Here it was possible to use any number of nodes as long as the memory was big enough to fit the distance matrix of the spectral clustering. Explicitly, the strong scaling configuration worked with nodes from 3 to 32 with 40% of the training dataset. But when using nodes that are not a power of 2 the algorithm failed with a unknown communication error in MPI. To solve this problem the MPI implementation was changed to the OpenMPI implementation. Here, nodes that are not a power of two are possible, but therefore nodes below ten are limited by the message size length of $2^{31} - 1$ values. This behavior seems to deviate from the standard message length of $2^{31} - 1$ values. An explanation could lie in different execution of the requested functionality by `mpi4py`. HeAT uses this Python wrapper to call MPI functions. Here, more in depth investigations could be performed in future work.

V. Conclusion

This work used spectral clustering on the remote sensing data of SO2Sat to perform unsupervised learning. All calculations were performed with HeAT on a compute cluster.

Looking at the clustering results, the goal of learning unsupervised was not successful. No LCZs could be recognized from the found clusters. This result was expected as the LCZs are defined manually instead of existing in the structure of the data itself. Another explanation could be the usage of RBF distances for image data. The machine learning world has turned to convolutional neural network for some years when working with image data. Pixelwise RBF or euclidean distances alone are not really able to recognize the similarity or dissimilarity of the different groups and cannot find an adequate clustering.

The performance of HeAT was profiled as well. The strong scaling of the spectral clustering algorithm works for a reasonable amount of nodes. When using a compute cluster this is usually a trade-off between waiting time for more resources and the compute time of the algorithm. A relevant point lies in the amount of memory that is necessary for spectral clustering. Due to calculation of the distance matrix the algorithm uses $O(n^2)$ memory. Therefore, the number of compute nodes to use is limited to the bottom by the needed memory. Still, the strong scaling shows that the compute that comes with more nodes is more useful, than calculating the algorithm on one fat node.

Possible extensions could focus on the result of the clustering itself. With the knowledge of the good scaling behavior, a grid search with a part of the dataset could be performed. Especially, the gamma parameter of the radial basis function kernel and the ϵ of an ϵ-neighborhood graph, allow a lot of optimization. Another approach that could be implemented is the clustering on a subset of the channels. Maybe, clustering only the RGB channels or another combination could be better when keeping the curse of dimensionality in mind.

Acknowledgment

The authors would like to thank the Steinbuch Centre for Computing for providing the necessary compute power for performing the here presented profiling. Furthermore, we would like to thank the HeAT team for providing support regarding any troubles and questions.

References

[1] G. Moore, "Cramming More Components Onto Integrated Circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, Jan. 1998. [Online]. Available: http://ieeexplore.ieee.org/document/658762/

[2] N. Khan, I. Yaqoob, I. A. T. Hashem, Z. Inayat, W. K. Mahmoud Ali, M. Alam, M. Shiraz, and A. Gani, "Big Data: Survey, Technologies, Opportunities, and Challenges," Jul. 2014, iSSN: 2356-6140 Pages: e712826 Publisher: Hindawi Volume: 2014. [Online]. Available: https://www.hindawi.com/journals/tswj/2014/712826/

[3] X. X. Zhu, J. Hu, C. Qiu, Y. Shi, J. Kang, L. Mou, H. Bagheri, M. Häberle, Y. Hua, R. Huang, L. Hughes, H. Li, Y. Sun, G. Zhang, S. Han, M. Schmitt, and Y. Wang, "So2Sat LCZ42: A Benchmark Dataset for Global Local Climate Zones Classification," *arXiv:1912.12171 [cs, eess]*, Dec. 2019, arXiv: 1912.12171. [Online]. Available: http://arxiv.org/abs/1912.12171

[4] M. Bauer and M. Garland, "Legate NumPy: accelerated and distributed array computing," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Denver Colorado: ACM, Nov. 2019, pp. 1–23. [Online]. Available: https://dl.acm.org/doi/10.1145/3295500.3356175

[5] C.-C. Huang, Q. Chen, Z. Wang, R. Power, J. Ortiz, J. Li, and Z. Xiao, "Spartan: A Distributed Array Framework with Smart Tiling," p. 16.

[6] K. Krajsek, C. Comito, M. Gotz, B. Hagemeier, P. Knechtges, and M. Siggel, "The Helmholtz Analytics Toolkit (HeAT) - A Scientific Big Data Library for HPC -," p. 4.

[7] M. P. I. Forum, "MPI: A Message-Passing Interface Standard," p. 868, Jun. 2015.

[8] "Open MPI: Open Source High Performance Computing." [Online]. Available: https://www.open-mpi.org/

[9] "Intel® MPI Library." [Online]. Available: https://www.intel.com/content/www/us/en/develop/tools/oneapi/components/mpi-library.html

[10] "NumPy." [Online]. Available: https://numpy.org/

[11] I. D. Stewart and T. R. Oke, "Local Climate Zones for Urban Temperature Studies," *Bulletin of the American Meteorological Society*, vol. 93, no. 12, pp. 1879–1900, Dec. 2012, publisher: American Meteorological Society Section: Bulletin of the American Meteorological Society. [Online]. Available: https://journals.ametsoc.org/view/journals/bams/93/12/bams-d-11-00019.1.xml

[12] J. Xue, R. You, W. Liu, C. Chen, and D. Lai, "Applications of Local Climate Zone Classification Scheme to Improve Urban Sustainability: A Bibliometric Review," *Sustainability*, vol. 12, no. 19, p. 8083, Sep. 2020. [Online]. Available: https://www.mdpi.com/2071-1050/12/19/8083

[13] D. Grigg, "The Logic of Regional Systems1," *Annals of the Association of American Geographers*, vol. 55, no. 3, pp. 465–491, 1965, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8306.1965.tb00529.x. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8306.1965.tb00529.x

[14] U. von Luxburg, "A Tutorial on Spectral Clustering," *arXiv:0711.0189 [cs]*, Nov. 2007, arXiv: 0711.0189. [Online]. Available: http://arxiv.org/abs/0711.0189

[15] C. Lanczos, "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators," *Journal of Research of the National Bureau of Standards*, vol. 45, no. 4, p. 255, Oct. 1950. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/jres/045/jresv45n4p255_A1b.pdf

[16] "2.3. Clustering — scikit-learn 0.23.2 documentation," Dec. 2020. [Online]. Available: https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation

[17] J. Vert, K. Tsuda, and B. Schölkopf, "A Primer on Kernel Methods," *Kernel Methods in Computational Biology, 35-70 (2004)*, Jan. 2004.

[18] A. Rosenberg and J. Hirschberg, "V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure." Jan. 2007, pp. 410–420.

[19] J. M. Wooldridge, "Introductory Econometrics: A Modern Approach," p. 910, 2012.

[20] "PEP 282 – A Logging System." [Online]. Available: https://www.python.org/dev/peps/pep-0282/

[21] "Weights & Biases." [Online]. Available: https://wandb.ai/

[22] H. S. Haefner, "KIT - SCC - Services - Research and develop - High Performance Computing - High Performance Computing (HPC) and Clustercomputing - bwUniCluster 2.0+GFB-HPC," Sep. 2020, archive Location: KIT Publisher: Haefner, Hartmut (SCC). [Online]. Available: https://www.scc.kit.edu/en/services/bwUniCluster_2.0.php

[23] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer, 2003, pp. 44–60.