6 Earth Science with NextDBSCAN, NextSVM and Deep Learning

Ernir Erlingsson, Helmut Neukirchen, Morris Riedel
University of Iceland, UoI, Iceland

ere29@hi.is

6.1 Introduction

The University of Iceland (UoI) explored the possibilities of combining machine learning methods with the MSA offered by the DEEP-EST system.this aim, UoI selected three machine learning methods and tailored their implementations for the DEEP-EST system. These three applications are:

- NextDBSCAN, a new parallel DBSCAN algorithm used for non-approximate and approximate density-based clustering of arbitrary datasets, such as large three-dimensional point-clouds generated via LiDAR scans. Note that NextDBSCAN supersedes HPDBSCAN, which was used at the start of the project, as HPDBSCAN proved unsuitable for GPU platforms and Exascale systems due to critical inherent scaling issues. Therefore, we started from scratch and developed NextDBSCAN, a DBSCAN algorithm which exhibits good scaling properties irrespective of the input dataset and parameters. We believe that our implementation of NextDBSCAN can be employed by future Exascale HPC systems, especially if it is optimized further for such systems. We substantiate our claim in the following subsections. We have also released the application as a Free- and Open-Source Software (FOSS) to the general public via a Github repository[77].
- NextSVM, a new parallel Support Vector Machine (SVM) for supervised learning classification tasks using labelled datasets (such as remote sensing images with ground-truth). Similar to NextDBSCAN, NextSVM supersedes PiSVM, which was used initially but proved unusable as its performance scaling plateaus after only a few nodes. NextSVM, however, scales much better and supports the usage of GPU accelerators. Through the DEEP-EST project, we have managed to improve especially the model training performance and scalability towards Exascale, which we illustrate and discuss in the following subsections. We also

[77] https://github.com/ernire/nextdbscan-exa

provide NextSVM as a FOSS repository for the public and machine learning communities[78].

- Deep Learning, via TensorFlow and the Keras extension, for computer vision (including remote sensing images), using Convolutional Neural Networks (CNNs). For scalability across multiple nodes, the Horovod framework is used to run TensorFlow/Keras in a distributed fashion.

6.2 Application structure

6.2.1 NextBSCAN

For clustering, the new parallel NextDBSCAN algorithm is used. The core partitions are those related to the pre-processing of the input data, the actual clustering algorithm (i.e. local clustering and global merge) and storing its results. The data selection partition, used for further data analysis and processing, is optional and is only used if there is further data study. The individual partitions of NextDBSCAN are described in the following sub-sections.

6.2.1.1 Data processing

In this phase, the point-cloud dataset is spatially divided using a hyper-grid overlay of different size and/or offsets. The point-cloud dataset is then divided equally among the processes and each point is sorted into its respective cell. Afterwards, the sorted list is stored, and a heuristic is applied to attempt to load-balance the data-grid by dividing it into chunks that fit in RAM, i.e. the total number of executions (and the cell span of each) is determined so that the whole grid can be processed without overwhelming the available hardware resources. The heuristic attempts in particular to divide the dataset into equally sized execution tasks with respect to the number of point comparisons.

After the data has been divided, each chunk is processed further, resulting in smaller chunks, which can then be finally processed by a local DBSCAN implementation.

6.2.1.2 Data chunk pre-processing

Upon execution, the chunk which is being processed is divided into further smaller chunks equal to the number of MPI processes, where a similar load-balancing heuristic to the one described in the section above, i.e. the hyper-grid cells are divided among processes, tries to keep the number of point comparisons for each process as close as possible.

[78] https://github.com/ernire/next-svm

6.2.1.3 Local parallel DBSCAN

Each MPI process performs a local DBSCAN clustering on its assigned cells, using OpenMP for shared memory parallelism. Hence, clusters that span different MPI processes are not yet detected in this step and, as a consequence, a merging approach is performed in the next step.

6.2.1.4 Merge clusters

After clustering, the locally obtained cluster labels are exchanged among the MPI processes to make sure that clusters spanning over multiple cells receive the same unique global cluster label. This is done with selected rules in the algorithm.

6.2.1.5 Results and resiliency

The old HPDBSCAN was not particularly robust as it did not include any measures to increase the application's resiliency. This was not really necessary because the limit on the size of the point-cloud datasets also limited the execution time to such a degree that a system failure would never be very cost intensive. For larger datasets such as those expected in the Exascale era, this must be improved.

We therefore apply a simple but effective measure to add resiliency to the NextDBSCAN application by storing calculated cluster labels to the persistent memory, taking advantage of the inherent compartmentalization of the computation offered by the dataset hyper-grid overlay. This allows the execution to restart using the most recently stored data. In effect, we are adding checkpointing to the application.

6.2.1.6 Data selection

When applying Level of Detail (LoD) or continuous Level of Importance (cLoI) studies, it is possible to modify the point-cloud, e.g. zoom in/out, and perform clustering on sub-set selections of the original dataset, possibly using a new hyper-grid overlay. This in turn may result in various clusters in memory on different datasets that may be often re-read depending on zoom levels. Therefore, it makes sense to store clustered datasets of sub-sets into persistent memory for further iterations of LoD/cLoI studies.

6.2.2 NextSVM

This second machine learning application performs classification of data using the parallel SVM implementation called NextSVM. It can be divided into four partitions described in the following subsections.

6.2.2.1 I/O

For training, the feature engineered labelled HDF5 input dataset is read in parallel by numerous processes. The input dataset has been feature engineered to increase the

likelihood that the model converges, and to reduce the overall computation time by skipping features that have otherwise little or no effect on the training process. Therefore, feature engineering is usually performed with the goal of increasing the accuracy. But as different feature engineering techniques are usually applied, the input datasets may in fact change from time to time even though the raw data is the same.

6.2.2.2 Training

NextSVM training is performed iteratively on the non-linear input data, processing one sample at a time using sequential minimal optimisation (SMO), but using the so-called "kernel-trick" to linearly separate the data in a higher dimension space. The aim is to construct a model which can be used for classification with a high accuracy. This phase is computationally expensive, generally requiring very many samples to be able to achieve good classification accuracy.

6.2.2.3 Validation

Validation is a process in machine learning for model selection that in turn is not only related to the right model (e.g. SVM, neural network, Random Forest, etc.) but also their parameters. We are using a non-linear SVM with RBF (Radial Basis Function) kernels (having a kernel parameter *gamma*) and soft margins (i.e. allowed cost of *error* parameter), therefore an exhaustive search must be made, e.g. using a 10-fold cross-validation, to determine those input training parameters that give the best training results. This is typically performed via a grid search over the parameters and is a process that is embarrassingly parallel, i.e. parallelises nicely: depending on the number of parameters, the overall computing time could be quite significant, but the different runs do not require interaction between them.

6.2.2.4 Inference

Model inferencing in NextSVM is an embarrassingly parallel operation that performs predictions using an otherwise unseen labelled dataset, which can be used to determine a model's accuracy. Furthermore, when a model exhibits good accuracy, it can then finally be used for making classifications on new unseen datasets.

6.2.3 Deep learning

The third machine learning application does classification using deep learning. It uses partly the same dataset as the SVM application for supervised learning to allow for a comparative study of the different classification approaches. For unsupervised learning, however, different multi-spectral datasets are explored. The application uses state-of-the-art deep learning for image pattern recognition, namely Convolutional

Neural Networks (CNNs), that are known for detecting spatial properties in data. The partitions of the deep learning chain are described in the following sub-sections.

6.2.3.1 I/O

As mentioned above, the application can, in principal, process and classify the same input datasets as the SVM application. However, it uses the raw, non-feature-engineered datasets whilst SVM uses a processed, feature-engineered version of it. The reason is that 'feature learning' is an intrinsic part of deep neural networks in general and CNN in particular. In the future, other datasets will be used, e.g. to support Sentinel satellite data provided by the European Copernicus remote sensing programme. This dataset offers enormous volume, with over 23 Terabytes of new data per day, and requires Exascale computing when performing land cover analysis at large scale over time.

6.2.3.2 Training

Training is performed using CNNs since we are mostly handling remote sensing image input data for which CNNs perform best. Additionally, Stochastic Gradient Descent (SDG) and back-propagation are used as standard techniques employed during the training phase. Due to the multi-spectral nature of the input datasets, a 3D CNN is used, which is a special form of regular CNNs that can better take advantage of the multiple input data dimensions (2D spatial data with multiple spectra).

6.2.3.3 Inference

The trained models acquired in the previous partition are evaluated by measuring their prediction accuracy on previously unseen, but labelled input data and thus inferring their suitability for further training. Finally, the trained models can be used for making classifications on new (and even unlabelled) datasets.

6.2.3.4 Transfer learning

After a neural network model has been trained and tested, that model can be re-used, even in parallel by multiple users, as a foundation for additional training on other datasets using transfer learning techniques. The simplest technique involves making an incision in the neural network next to the output layer and adding more layers in-between, prior to fresh training. There are also known existing pre-trained neural networks (e.g. OverFeat) that make sense to have available in the persistent memory for different application use cases. Each network mostly consists of a matrix of weights in multiple dimensions (i.e. for each layer), but memory-footprint can nonetheless be significant when deep learning architectures are used.

6.3 Application mapping

For each of the three applications, we selected multiple mappings to the MSA of the DEEP-EST system, as it was not clear in advance which path would offer each application the greatest benefits.

While the initial assumption was that NextDBSCAN benefits from a hybrid usage of CPU and GPU and therefore would use either CM together with ESB or CM together with DAM, the NextDBSCAN algorithm, data structure, and implementation has since then been improved so that NextDBSCAN runs fastest when using solely CPUs or solely GPUs. Measurements (see Sections 6.5 and 6.6) have shown that it depends on the dataset and the DBSCAN clustering parameters whether using CPU or GPU is preferable (a hybrid approach suffers from a data transmission overhead), although the GPU provides a clear benefit for the vast majority of examined cases. However, NextDBSCAN can run on the CPUs of CM or DAM (benefiting from the huge RAM in the DAM) or on the GPUs of ESB or DAM. Depending on the actual use case, only one MSA module might therefore be used, but to give an idea of a more complex workflow and mapping to the MSA, Figure 6.1 shows an example where a grid parameter search is performed in parallel on all MSA modules, i.e., doing DBSCAN clustering with different values for its two parameters to find out which parameter combination yields best clustering results. By analysing the dataset (e.g. point density), it is first determined which parameter combination is best executed on which MSA module and after that, all available CPUs and GPUs of the MSA modules can be used for the embarrassingly parallel computation using the different parameters on the same dataset (“ensemble scaling”). Optionally (dashed lines in Figure 6.1), it is possible to re-run the clustering with a narrowed down dataset selection or a different parameter range selection.

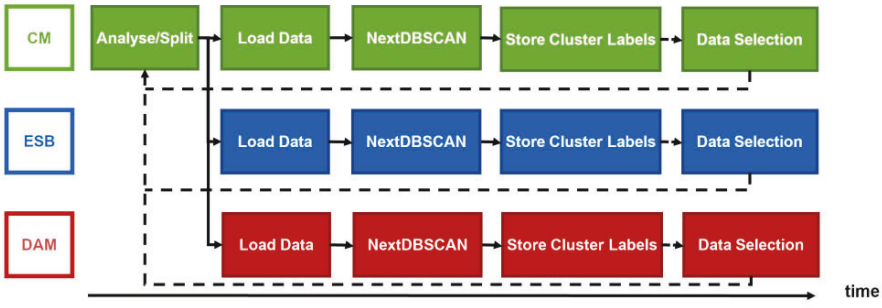

Figure 6.1: Schematic workflow of a grid-search using NextDBSCAN in the MSA

In addition, the MSA usage of NextSVM has been adjusted while implementing NextSVM. For model inference (i.e. testing the trained model), only a few GPUs and

not a lot of RAM are needed; hence, the DAM (i.e. fewer GPUs) or the ESB (i.e. smaller RAM) can be used. Figure 6.2 depicts in the upper part a mapping where model training is performed using the many CPUs of the CM and model inference is then done using the ESB's GPUs and the model is locally stored in the ESB. The lower part depicts model training using the many GPUs of the ESB and then model inference on the DAM using also DAM's DCPMM for model storage. (These stored models can then be re-used for additional training via transfer learning and/or further inference.) An alternative, CPU-based mapping has been described in a research paper[79].

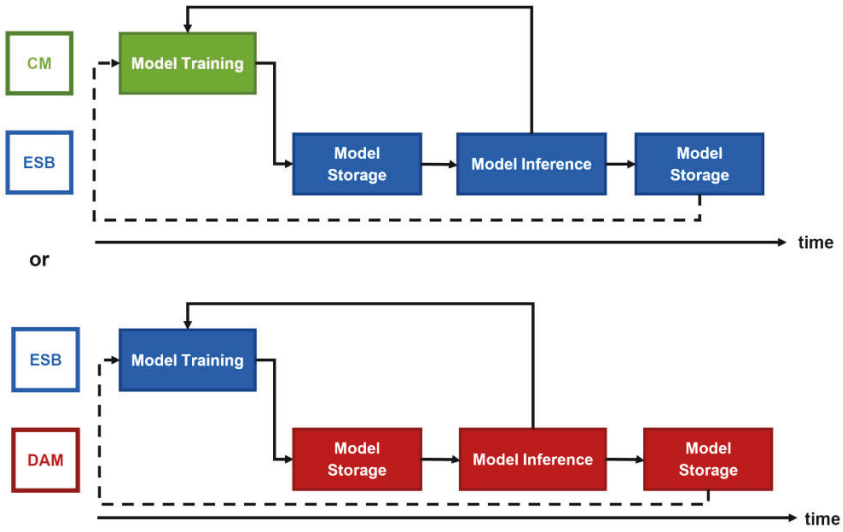

Figure 6.2: Two different schematic workflows of NextSVM in the MSA

UoI's Deep Learning application also explores two different MSA mappings to support performance comparisons: both mappings use the same MSA modules, namely the ESB and DAM (see Figure 6.3). The main difference between these "mirrored" mappings lies in which MSA module trains the neural networks, and which infers their quality. After training, the obtained models are stored at two locations, where one supplies the inference with input data and the other is enhanced by metadata and can be used for any subsequent training, e.g. transfer learning.

[79] Ernir Erlingsson, Gabriele Cavallaro, Morris Riedel, Helmut Neukirchen. Scaling Support Vector Machines Towards Exascale Computing for Classification of Large-Scale High-Resolution Remote Sensing Images. IEEE International Geoscience and Remote Sensing Symposium (IGARSS) 2018. DOI: 10.1109/IGARSS.2018.8517378 IEEE 2018.

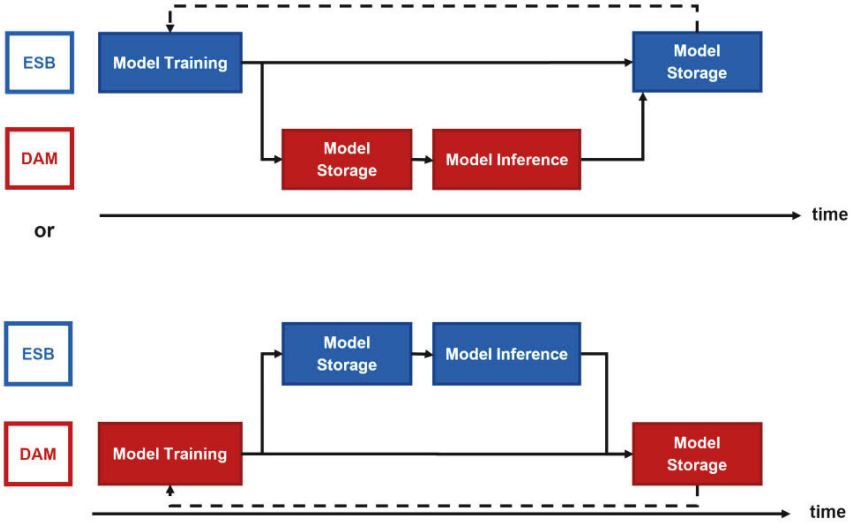

Figure 6.3: Two different schematic workflows of deep learning application in the MSA

Benefits of using of the NAM prototype has been investigated and published in research papers[80, 81].

6.4 Porting experience

At the start of the DEEP-EST project, we expected the porting procedure for our applications to be straightforward. However, this assumption proved false as we discovered a critical scaling problem with our initial Support Vector Machine application (PiSVM), which is explained in Section 6.7. Furthermore, we found that our initial density-based clustering application (HPDBSCAN) was inherently incompatible with the use of GPUs, as it was per-design unable to exploit the parallelism offered by accelerators. Therefore, our porting efforts revolved mostly around re-designing these applications from the ground-up, thereby producing NextSVM and NextDBSCAN, to provide necessary compatibility with distributed memory GPUs and the MSA. Our deep learning application implementation, however, was virtually unaffected by the project's

[80] Ernir Erlingsson, Gabriele Cavallaro, Morris Riedel, Helmut Neukirchen. Scalable Workflows for Remote Sensing Data Processing with the DEEP-EST Modular Supercomputing Architecture. IEEE International Geoscience and Remote Sensing Symposium (IGARSS) 2019, DOI: 10.1109/IGARSS.2019.8898487, IEEE 2019.

[81] Ernir Erlingsson, Gabriele Cavallaro, Andreas Galonska, Morris Riedel, Helmut Neukirchen. Modular Supercomputing Design supporting Machine Learning Applications. International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO 2018), DOI: 10.23919/MIPRO.2018.8400031, IEEE 2018.

focus shift towards GPUs, due to its high-level implementation in TensorFlow and Keras that anyway supports CPUs and accelerators.

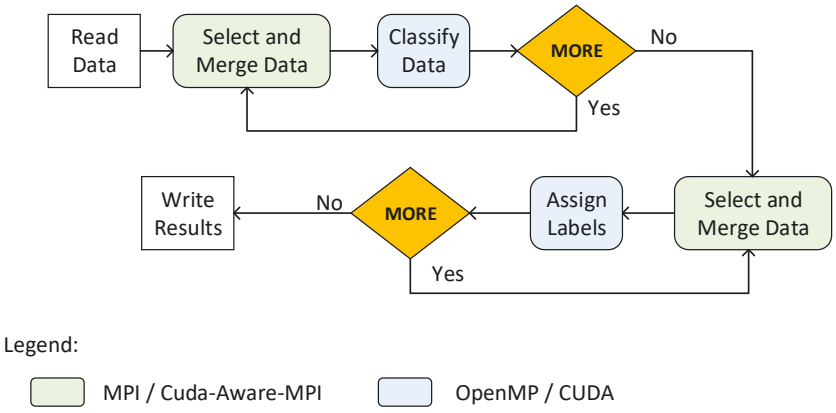

Figure 6.4: An algorithmic flowchart of NextDBSCAN and its numerous versions (OpenMP, CUDA, MPI)

Codebase fragmentation was one of our key concerns as we had to develop multiple application versions to support both CPUs and GPUs. This increases the development time, source code size, and the risk of human errors, which could endanger the quality of the application and its performance results, e.g., when the CPU and GPU versions produce different results due to an error in one of them, or both. To mitigate this problem, we decided to create a single cross-platform version of our application, which can exploit the project's CPUs and GPUs (see Figure 6.4 for a single flowchart of NextDBSCAN supporting MPI, OpenMP, and CUDA). To this effect, we developed a library (called Magma) that mimics the C++ standard library blueprint, and subsequently wrote both NextDBSCAN and NextSVM with it. The main benefit of the library is that it takes care of linking the source code to either the C++ STL (in case of CPU) or CUDA Thrust (in case of GPU) libraries, as depicted in Figure 6.5.

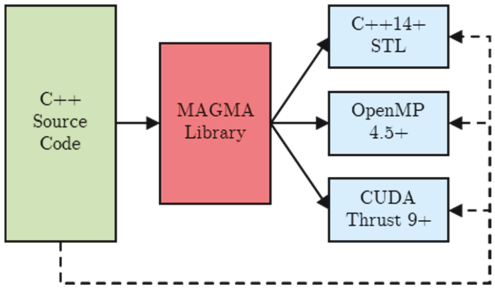

Figure 6.5: Magma Library Schematic Overview

Technically, Magma is a C++ header library that makes extensive use of C++ templates to offer compile-time polymorphism for increased usability at the expense of a small compile-time overhead. Specific compiler flags dictate which header files are used, and therefore which internal libraries are used. Currently our Magma library encapsulates the C++ STL, OpenMP 4.5+, and CUDA Thrust 9+. However, it can be expanded to cover more software libraries as there is nothing in its inherent design that limits the number of supported internal libraries. The Magma library is available to all as FOSS via a public GitHub repository[82].

```
magma::for_each(n_offset_size, v_coord_cell_size.size() - 1, [=]
#ifdef CUDA_ON
    __device__
#endif
    (auto const &i) -> void {
        it_coord_cell_size[i] = it_coord_cell_offset[i + 1] - it_coord_cell_offset[i];
    });
```

Figure 6.6: An example of the usage of a Magma library for each-loop, taken from NextDBSCAN source code using C++ pass-by-value lambdas

By developing NextDBSCAN and NextSVM using the Magma library we could construct a single code-base for each application while still supporting the C++ STL, OpenMP and CUDA (via Thrust). The source code is identical for both the CPU and GPU platform with the exception of the necessary host and/or device annotations which CUDA requires to specify the execution target, as is outlined in Figure 6.6. C++ functors and/or lambdas can be used to specify kernels with or without parameters, which are copied to the kernel (or passed by value). This greatly facilitated the development of our applications by allowing us to target multiple platforms, while simultaneously maintaining a single, compact, code-base accompanied with unit tests. The reusable Magma library is ~2000 lines of code. NextDBSCAN is ~1500 lines, NextSVM is ~1000 lines, and our deep learning scripts ~500 lines of high-level Keras/TensorFlow code (which does not use Magma). The majority of our PMs went into the software development of our applications and the Magma library, following an iterative development process including in each iteration analysis, design, development, and testing. New versions were conceptualised and often scrapped when they proved inadequate. It is difficult to quantify the time spent directly on porting to the DEEP-EST MSA as it was one part of a bigger scope of developing NextDBSCAN and NextSVM from scratch, independently from the MSA. However, we estimate that a quarter of our development process can be attributed to MSA porting, as part of analysis, design and testing. However, developing an application from

[82] https://github.com/ernire/magma

scratch is not the same as porting exiting applications with good scalability onto the MSA. We believe that with the knowledge we now possess, we could retrofit any such an application onto the MSA in a matter of weeks, followed by an arbitrary amount of time for optimisations.

6.5 Scalability

We examined the scalability of our three applications with numerous modular benchmarks and present the highlights of our findings in this section. In Subsection 7.3.1., we also outline the path of our applications towards Exascale. Note that we use the term parallel efficiency according to standard practice, i.e. the speedup of the respective number of nodes when compared to using a single node, divided by the respective number of nodes. For our experiments, we used different dataset suites with multiple input parameters, where applicable, and re-ran each experiment three times, reporting the median of the measured results to remove the effect of outliers. We applied NextDBSCAN on large LiDAR point-cloud datasets, i.e. the Dutch AHN3[83] and Bremen[84] datasets. For NextSVM, we employed the Rome[85] and Indian Pines[86] datasets with their accompanying classification maps. Finally, we trained neural networks with deep learning using Sentinel-2 imagery tiles[87].

Figure 6.7 depicts NextDBSCAN's strong scaling properties when measuring the time-to-solution (TTS). We observe that for an equal number of nodes, the ESB significantly outperforms the CM (GPU vs. CPU), consistently reporting ~50x faster time-to-solution (TTS). These results were consistent for all experiments with big data. However, for smaller datasets the runtime difference between the modules shrunk as a function of its size, as the amount of parallel computations simply are not enough to sustain its scalability with GPUs, and the application's serial processing overhead becomes more and more dominant.

NextDBSCAN's parallel efficiency is depicted in Figure 6.8. On the CM, it remains relatively high for this strong scaling case, but drops more quickly on the ESB. We examined the cause and determined that most of it stems from each ESB node spending much less time doing computations compared with its CM counterpart, but a near equal amount performing MPI communications. Therefore, the MPI communication's latency and inherent scalability affects the parallel efficiency to a

[83] https://downloads.pdok.nl/ahn3-downloadpage/

[84] http://doi.org/10.23728/b2share.7f0c22ba9a5a44ca83cdf4fb304ce44e

[85] https://b2share.eudat.eu/records/daf6c389e54340b4b1416cf874251e77

[86] https://b2share.eudat.eu/records/8d1fbbba69944fc5a5ae01d1c141c37a

[87] https://scihub.copernicus.eu/

higher degree on the ESB. By increasing the problem size, the ESB's parallel efficiency remains higher as the computational load grows then faster than the distributed communication. However, for our comparison, we already selected the largest possible problem size that a single CM node can solve within a reasonable time duration.

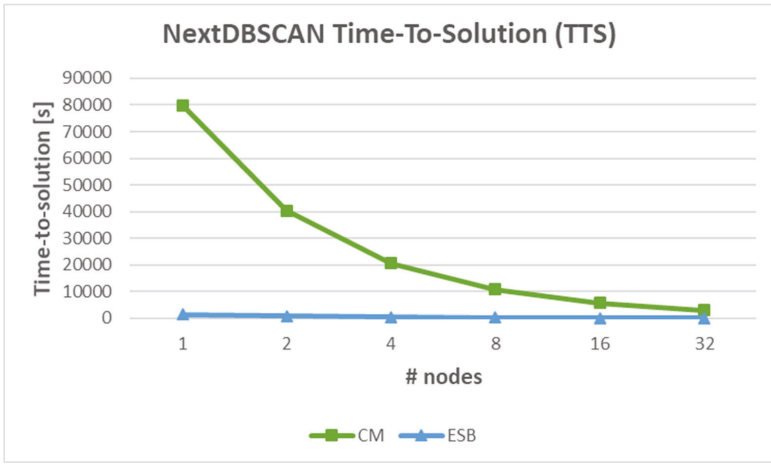

Figure 6.7: NextDBSCAN's time-to-solution, measuring strong scaling on CM (CPU) and ESB (GPU)

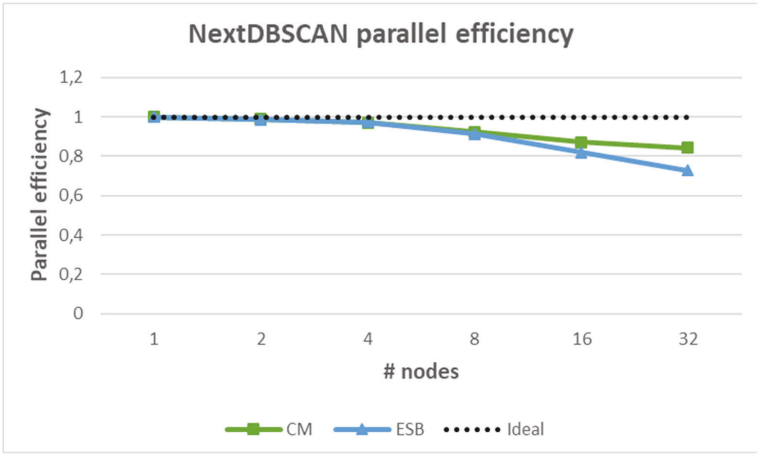

Figure 6.8: NextDBSCAN's parallel efficiency, measuring strong scaling on the CM (CPU) and ESB (GPU)

By using a heterogeneous approach, as is depicted in Figure 6.1, we were able to slightly improve the ensemble TTS performance of a typical grid-search, which executes a shared-memory version of NextDBSCAN concurrently on multiple nodes, using a different pair of parameters. Figure 6.9 shows the aggregated runtime values

using six different epsilon values, where each value represents eight different minPoint values, i.e. each column represent the total runtime of eight different doubling parameter pairs which share the same epsilon. The GPU performed better for most, but not all, parameter pairs, which leads to the optimal performance of the total aggregate being a heterogeneous combination using both CM and ESB.

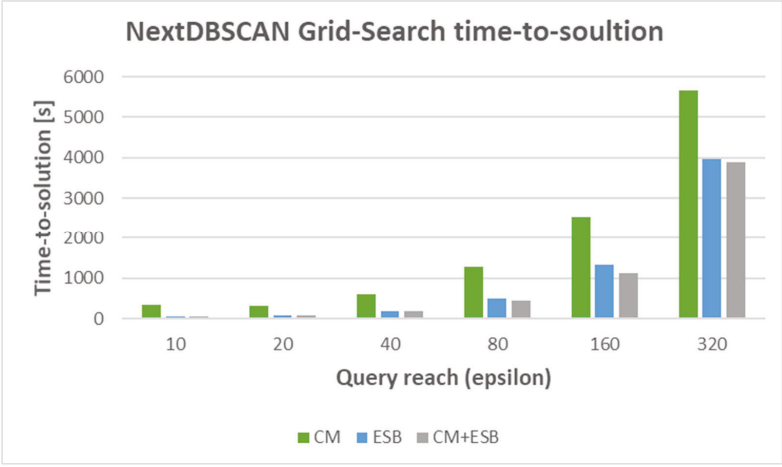

Figure 6.9: Grid-search runtimes aggregated for each epsilon value, which doubles every iteration

Figure 6.10 illustrates our main finding when using NextVSVM, measuring the time-to-solution for model training. We observe that when running NextSVM on a single-module the ESB starts with a faster baseline performance but then drops behind the CM after only 16 nodes. Here, we are implicitly comparing CPUs vs. GPUs, as NextSVM is in this scenario only using the GPU on the ESB. NextSVM, however, uses a Sequential Minimization Optimizer (SMO) solver that enforces a strong serial order of computations for a small part of the iterative algorithm. This part is bound by single core performance, which forms a severe bottleneck for NextSVM running exclusively on the GPU. The best time-to-solution performance was achieved by using both the CPU and GPU on the ESB module, as depicted by the light blue line, despite having higher offloading costs, as data is transferred more frequently between the GPU and CPU memories. Overall, 99% of the execution takes place on the GPU, but the remaining 1% CPU-time is critical to better maintain the speedup curve, as we can observe in the figure. Although the CM offers CPUs with a stronger single-core performance than the ESB, we still attained the best results using only the latter, as the gain with stronger cores did not overcome the cost of transmission at each iteration, mainly due to the very low number of necessary computations.

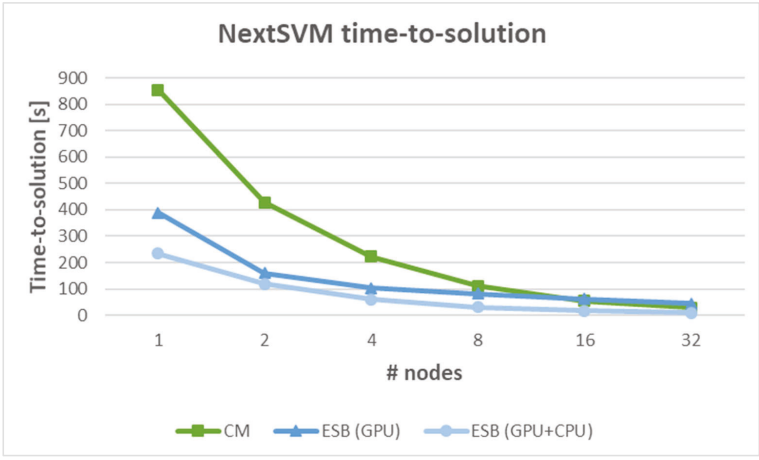

Figure 6.10: NextSVM's strong-scaling time-to-solution, measured on the CM and the ESB, where the former uses CPUs and the latter solely GPUs

Figure 6.11 shows the parallel efficiency of the two NextSVM versions that exhibit the best scaling properties, i.e., running on the CM, or ESB using both CPU and GPU. As the figure illustrates, the application running on the CM maintains its scalability better than the ESB. The main cause of this discrepancy is due to the offloading cost, as data is transferred to and from the GPU, which has a fixed size irrespective of the number of nodes, a consequence of the SMO solver algorithm employed by NextSVM (see also later discussion on fixing limitations).

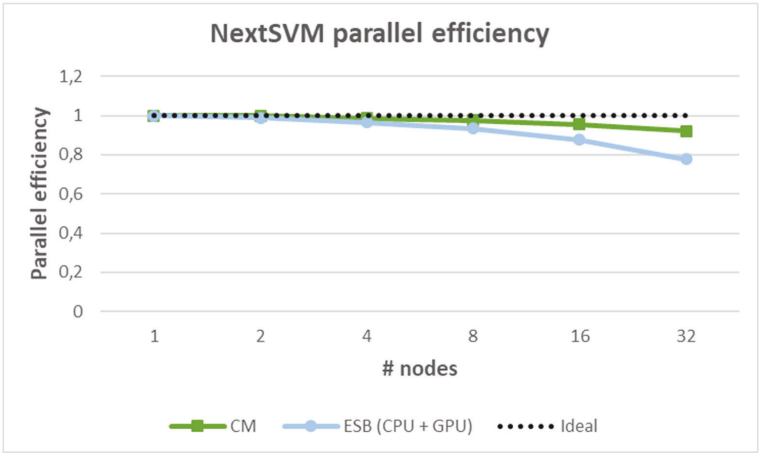

Figure 6.11: NextSVM parallel efficiency while performing strong scaling on the CM and the best ESB scaling with CPU + GPU

Finally, Figure 6.12 and Figure 6.13 depict the strong scalability on the ESB module while training a neural network with satellite imagery, using Keras/TensorFlow and the Horovod framework, for distributed computing. The first figure shows the effect of modifying the image size for each training batch, i.e. it is possible to cause some variations to the scalability curve by loading different batch sizes at once (using HDF5). However, when scaling up the number of nodes this variance will decrease over time, as is illustrated by the parallel efficiency. Overall, our experiments with parameters and I/O configurations had an insignificant effect on the training performance and scalability. Additionally, Horovod's scalability was worse than what we anticipated, making us doubt its suitability as a deep learning vehicle for satellite imagery on Exascale systems (see also later discussion on fixing limitations).

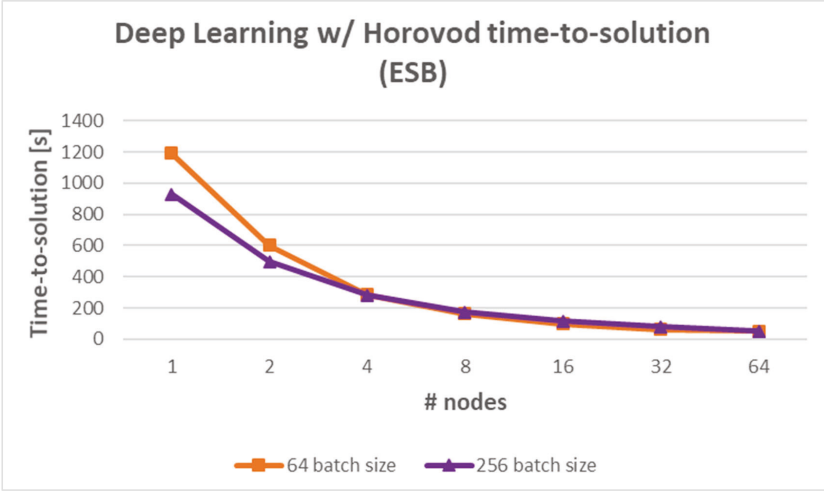

Figure 6.12: Deep Learning w/ Horovod strong-scaling time-to-solution using different training batch sizes

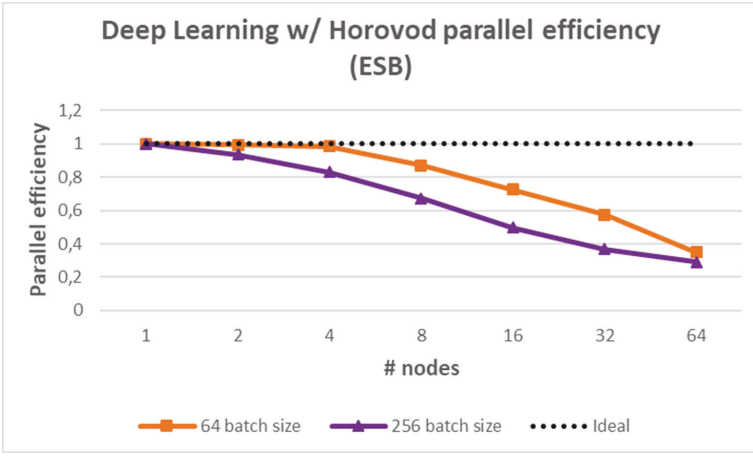

Figure 6.13: Deep Learning w/ Horovod parallel efficiency using different training batch sizes

6.5.1 Our path to Exascale

In summary, we achieved the above results in the following manner:

- We redesigned and reconstructed the MPI communication of our DBSCAN and SVM applications, optimizing them for Exascale HPC systems. We abandoned the master-slave paradigm such as was employed by PiSVM (where a single node controls the execution process and collects all the data) and replaced it with MPI collectives where each node's role is identical. We achieved the best results by limiting our applications to in-place `MPI_Allgather` and `MPI_Allreduce` communication, as much as possible.
    - For NextDBSCAN, we inferred that our initial usage of `MPI_Alltoallv` would scale significantly worse than using in-place `MPI_Allgather` with a fixed buffer size, i.e., using buffer padding where necessary. This was due to internal processing of the send and receive count buffers, which increase linearly with the number of nodes, coupled with the overhead of each node communicating its buffer size.
    - For NextSVM, we completely re-designed the communication strategy towards MPI collectives instead of point-to-point transmissions, using in-place `MPI_Allgather` with a fixed buffer size.
- By minimizing the number of memory allocations, we were able to significantly improve the shared-memory parallel efficiency of our applications. This was especially effective for NextDBSCAN where we managed to replace all

intermediate dynamic buffers with a single fixed buffer allocated at the start of the execution.

- Using the GPU accelerators, we were able to greatly reduce the time-to-solution (TTS) for our applications, with an even lower cost of energy, as depicted in Figure 6.14 and Figure 6.15.

6.5.1.1 What are the limitations? – Can they be fixed?

- NextDBSCAN is ready to be applied to Exascale systems without special limitations. The only requirement is that the aggregated memory is sufficiently large to store the input dataset. To the best of our knowledge, NextDBSCAN is the first non-approximate DBSCAN application that is a viable candidate, and the first to support distributed GPUs.
- NextSVM can also be applied to Exascale systems, but has limited usability as it currently only supports a single linear kernel and does not include the option of cross-validation. Note that our scalability results are kernel agnostic. To fix this, more development time is needed, and it is our hope that our public repository can attract additional open-source developers.
- Deep Learning model training with satellite imagery and Horovod failed to meet our expectations. There is no easy fix, as Horovod would have to be probably partially re-written to meet the demands of Exascale systems. Given the rapid progress of multiple deep learning libraries and frameworks these past years, it is not unlikely that another solution, better suitable for Exascale systems, is nascent.

6.5.1.2 How to use future Exascale systems

- NextDBSCAN can be used as-is for both CPU and GPU clusters. Its source code is compartmentalized to facilitate its usage for heterogeneous systems. However, our evaluation in DEEP-EST indicates that an arbitrarily sized and homogenous GPU cluster should be used for fastest results.
- NextSVM can also be used as-is for both CPU and GPU clusters using standard compilers and software stacks. However, according to the scalability results presented in Subsection 6.5, it is only realistically applied at a large scale to CPUs, not GPUs.
- Our research indicates that deep learning with Horovod is not a good fit for Exascale systems. However, its development continues and future versions, or other uses, could provide a more realistic option.

6.5.1.3 Where did the DEEP-EST project help on the way to Exascale?

The DEEP-EST project was instrumental in enabling us to improve our applications, both their performance and usability.

- The access to state-of-the-art hardware and software resources was critical to the development and optimization of our applications.
- The tools and workshops provided by BSC, combined with expert help from the consortium (in particular JSC, BSC, EPCC, and Intel), gave us the insight we needed to make critical decisions to improve the scalability and usability of our applications. Tracing and profiling with Extrae and Paraver, respectfully, visualized the scalability problems of PiSVM, as outlined in previous deliverables, which prompted the development of NextSVM to supersede it.
- The shift towards GPUs helped us improving scalability of our applications as it revealed bottlenecks which were more difficult to detect with the same number of CPU nodes. As an example, we discovered that we had underestimated the scalability impact of some short sequential code areas in NextDBSCAN. The strong single-core performance of the CPU coupled with the small number of nodes (relative to Exascale) had obscured the fact, but with GPU parallelism the adverse impact of the sequential code areas became apparent and after careful improvements we managed to eliminate them from the source code.
- The DEEP-EST MSA provided us with the modules we needed to study our applications across different CPUs, interconnects, and accelerators, which strengthened our performance claims. Additionally, it allowed us to design novel workflows across different hardware platforms.

6.6 Energy consumption

The total energy consumption was measured using the resources at our disposal in the DEEP-EST project. Figure 6.14 illustrates the aggregate energy consumption of NextDBSCAN, measured alongside strong scaling benchmarks depicted in Figure 6.7. Note that NextDBSCAN on the ESB runs near-exclusively on the GPU, using CUDA-aware MPI, requiring the CPU only for file system I/O. We observe that the difference between the CM and ESB module's energy consumption is similar to the runtime results. However, the difference is slightly larger for the energy consumption, as the ESB uses up to 60× less energy than the CM; i.e. for 32 nodes: 327k vs 18M (Joule), respectively. As the difference increases slightly with the number of nodes, it can be expected that the difference will be even greater for a higher number of nodes, although this hypothesis should be validated on larger systems than the DEEP-EST system.

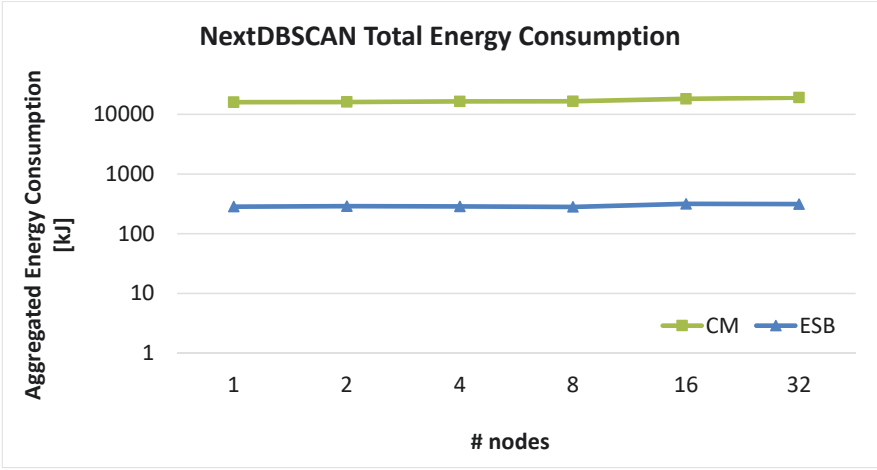

Figure 6.14: NextDBSCAN energy consumption using CM (CPU) and ESB (GPUs)

The total energy consumption of NextSVM is depicted in Figure 6.15. There is some correlation to the runtime of Figure 6.10: we can observe that the energy consumption increases for both the CPU and GPU as its scalability starts to flatten. We also note that the ESB's energy consumption increases faster than the CM, also corresponding to Figure 6.10. Additionally, we can see yet again that the runtime with GPUs requires less energy than using CPUs, however, this should flip when using more nodes, as the GPU runtime will scale progressively worse.

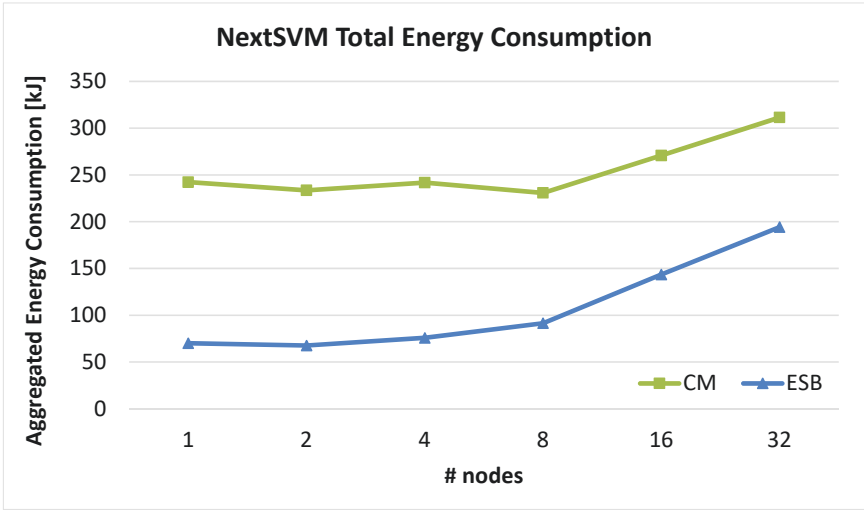

Figure 6.15: NextSVM energy consumption using CM and ESB (GPUs)

6.7 Performance comparison

After over three years of development, this subsection compares our current application status with their status at the start of the DEEP-EST project. Most of our effort has been spent on developing NextDBSCAN and NextSVM, which have already been outlined in previous sections. Their effectiveness is best demonstrated by comparing them to their predecessors, HPDBSCAN and PiSVM respectively. One of the greatest weaknesses of our initial applications was their surprising lack of scalability, as shown in Figure 6.16, which can in turn be compared to Figure 6.8 and Figure 6.11 to see the effect that DEEP-EST has had on our DBSCAN and SVM applications.

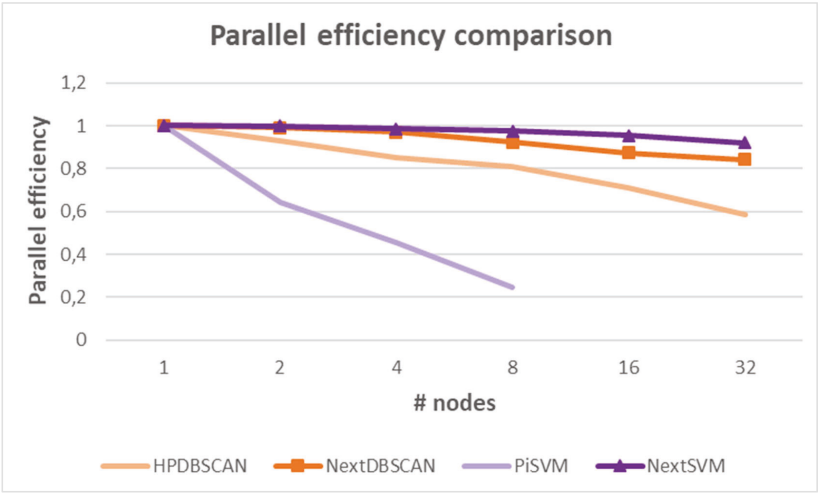

Figure 6.16: Strong scaling parallel efficiency for HPDBSCAN and PiSVM

For PiSVM the parallel efficiency was consistent for model training, mostly irrespective of the dataset used. After studying the application and its underlying algorithms, we found that the problem is caused by a poor distributed communication strategy in combination with too few computations being performed at each iteration. These problems had not yet surfaced, as no large scale performance measurements had been performed on the application for several years, and slower CPUs managed to mask the problem sufficiently by spending a larger portion of the total runtime doing computations, and less doing communication. PiSVM uses the well-known sequential minimal optimisation (SMO) solver, which solves the quadratic programming (QP) problem. The QP problem arises during the training of support vector machines by breaking the problem down to its smallest possible sub-problems which involves optimizing a sequence of pairs of Lagrange multipliers from the problem's dual form expression. This solver is the backbone of most SVM application produced in the last

two decades, where the computation is parallelised by distributing necessary computations to optimize one pair of Lagrange multipliers among the computational nodes being employed. Therefore, the amount of computation which can be achieved from a single pair of Langrangian is limited.

For HPDBSCAN, the parallel efficiency is more mixed, ranging from good to poor, depending on the dataset and input parameters but always degrading fairly rapidly in scalability with an increase in node cardinality. We analysed its code and could determine that the problem occurs in a grid-based data structure which is tightly integrated in the application, using it with heuristics both for data redistribution and thread load-balancing. However, this structure always leads to an imbalance, especially in datasets of high-dimensionality and/or when using a large number of nodes. Additionally, we found that HPDBSCAN's overall usability is limited by design, as it employs a tessellation indexing structure whose range is limited to 64-bit integers, i.e. the total number of possible cells cannot exceed the maximum value of an unsigned 64-bit integer. Although this is a large number, it is easily exceeded for even low dimensional datasets with a low epsilon input parameter.

The Deep Learning application with Horovod is a new application in the form of a high-level Keras/TensorFlow script, and could therefore not be compared to a previously developed application, as we could do with our DBSCAN and SVM applications.

6.8 Conclusion

Work in the DEEP-EST project took a very different path than originally anticipated. Instead of spending most of our effort tailoring already proven applications to the project's MSA platform, their intrinsic scalability and portability limitations led to the necessity to scrap most of them and start from scratch. However, our new applications, NextDBSCAN and NextSVM respectively, are much stronger than the previous applications. We discovered new algorithmic approaches to enhance the scalability of our algorithms and their baseline performance, consistently outperforming the best available algorithms that are freely available to perform the same task.

To our knowledge, NextDBSCAN is the first non-approximate DBSCAN clustering algorithm supporting distributed GPU computing. Furthermore, it also exhibits good scaling properties, as we have shown in previous sections. Our research indicates that NextDBSCAN is a candidate application for Exascale systems, using both CPUs and GPUs. With NextSVM, we managed to refit the old and proven SMO solver to parallel systems, with good scaling using CPUs, and show that there is potential to use GPUs to accelerate even via a heterogeneous approach.

Our results using the Horovod framework with TensorFlow on DEEP-EST are underwhelming and demonstrate that more work must be done in order for it to reach Exascale system potential. Additionally, the landscape of deep learning is still changing rapidly and it is difficult to predict which deep learning library will be utilized on future Exascale systems.

Overall, the GPU accelerator was a key component on our path towards Exascale: all our applications gained a significant performance benefit by employing it, also in terms of less energy used. Last but not least, we also published numerous research articles to further research in HPC and machine learning, with some of our greatest algorithmic findings, namely NextDBSCAN and NextSVM, still pending publication.