H2020-FETHPC-01-2016

DEEP-EST

DEEP Extreme Scale Technologies

Grant Agreement Number: 754304

D1.3

Application distribution strategy

Final

Version: 1.0

Author(s): P. Martínez (BSC)

Contributor(s): H. E. Plesser (NMBU), P. Petkov (NCSA), V. Pavlov (NCSA), S. Markov (NCSA), J. Romein (ASTRON), J. Amaya (KU Leuven), D. Gonzalez (KU Leuven), E. Erlingsson (UoI), H. Neukirchen (UoI), G. Cavallaro (UoI), S. Barakat (UoI) M. Riedel (UoI), M. Girone (CERN), V. Khristenko (CERN)

Date: 29.06.2018

Project and Deliverable Information Sheet

<table>
<tr><td rowspan="7">DEEP-EST Project</td><td colspan="2">Project Ref. №: 754304</td></tr>
<tr><td colspan="2">Project Title: DEEP Extreme Scale Technologies</td></tr>
<tr><td colspan="2">Project Web Site: http://www.deep-projects.eu</td></tr>
<tr><td colspan="2">Deliverable ID: D1.3</td></tr>
<tr><td colspan="2">Deliverable Nature: Report</td></tr>
<tr><td>Deliverable Level:
PU *</td><td>Contractual Date of Delivery:
30 / June / 2018

Actual Date of Delivery:
29 / June / 2018</td></tr>
<tr><td colspan="2">EC Project Officer: Juan Pelegrín</td></tr>
</table>

* - The dissemination levels are indicated as follows: PU = Public, fully open, e.g. web; CO = Confidential, restricted under conditions set out in Model Grant Agreement; CI = Classified, information as referred to in Commission Decision 2001/844/EC.

Document Control Sheet

<table>
<tr><td rowspan="5">Document</td><td colspan="2">Title: Application distribution strategy</td></tr>
<tr><td colspan="2">ID: D1.3</td></tr>
<tr><td>Version: 1.0</td><td>Status: Final</td></tr>
<tr><td colspan="2">Available at: http://www.deep-projects.eu</td></tr>
<tr><td colspan="2">Software Tool: Microsoft Word</td></tr>
<tr><td colspan="3">File(s):DEEP-EST_D1.3_Application_distribution_strategy_v01.0</td></tr>
<tr><td rowspan="4">Authorship</td><td>Written by:</td><td>P. Martínez (BSC)</td></tr>
<tr><td>Contributors:</td><td>H. E. Plesser (NMBU), P. Petkov (NCSA), V. Pavlov (NCSA), S. Markov (NCSA), J. Romein (ASTRON), J. Amaya (KU Leuven), D. Gonzalez (KU Leuven), E. Erlingsson (UoI), H. Neukirchen (UoI), G. Cavallaro (UoI), S. Barakat (UoI) M. Riedel (UoI), M. Girone (CERN), V. Khristenko (CERN)</td></tr>
<tr><td>Reviewed by:</td><td>Niels Burkhardt (EXTOLL), I.Schmitz (ParTec)</td></tr>
<tr><td>Approved by:</td><td>BoP/PMT</td></tr>
</table>

Document Status Sheet

Version	Date	Status	Comments
1.0	29/June/2018	Final version	EC submission

Document Keywords

Keywords:	DEEP-EST, HPC, Exascale, Applications, Co-design

Table of Contents

List of Figures

Executive Summary

The main goal of the applications work package, namely WP1, in the DEEP extreme scale technologies (DEEP-EST) project is to assess the modular supercomputing architecture (MSA) developed in the project and to evaluate the DEEP-EST prototype. For this purpose, six applications from a wide range of scientific fields are chosen. These will show that the new architecture is beneficial for not only one specific kind of application, but for several ones and in different ways.

This third deliverable gives a detailed partition of each application as well as its mapping into the MSA. It is worth noting that the different mappings presented in this work are determined, firstly, by each application's specific requirements and, secondly, by the design and development group (DDG) choice of architecture for the different modules that will integrate the MSA, therefore resulting in a truly co-design effort. The changes undergone by the applications and their performance assessment (based on the benchmarking and analysis tools introduced in early stages of the project) will highlight the importance of the DEEP-EST co-design project.

1 Introduction

The first deliverable D1.1 reported on each DEEP-EST's application structure and its requirements, both in terms of hardware and software, for the co-design of the MSA to be built within the DEEP-EST project[1]. The second deliverable D1.2 presented the benchmarks that will track the performance progress that each application will undergo throughout the lifespan of the project[2].

The aim of this third document, entitled "Application distribution strategy", is to detail the application distribution within each module of the MSA. For this purpose, the present document is structured as follows:

1. Firstly, each section begins with a short introduction to the corresponding application or group of applications.
2. Secondly, each application is partitioned, that is, dissected into its logical parts; these logical parts make sense on their own and, more importantly, vis-à-vis the MSA.
3. Finally, each application is mapped into different modules of the MSA. This is illustrated via workflows, which effectively show how applications intend to use each specific module of the MSA, during the timeframe of their execution, and depending on their particular requirements.

[1]A. Kreuzer, P. Martínez, H. E. Plesser, P. Petkov, V. Pavlov, J. Romein, J. Amaya, D. Gonzalez, M. Riedel, M. Girone, V. Khristenko. "Application co-design input", Deliverable D1.1, DEEP Extreme Scale Technologies (2017).
[2]P. Martínez, H. E. Plesser, P. Petkov, V. Pavlov, J. Romein, J. Amaya, D. Gonzalez, E. Erlingsson, H. Neukirchen, G. Cavallaro, S. Barakat, M. Riedel, M. Girone, V. Khristenko. "Application use cases and traces", Deliverable D1.2, DEEP Extreme Scale Technologies (2017).

2 Task 1.2: Neuroscience (NMBU)

2.1 Introduction

The long-term goal of the neuroscience task in DEEP-EST is to provide an optimised setup for the integrated simulation and analysis of large-scale brain activity. Such in situ analysis is essential to facilitate the interactive investigations of brain dynamics, where scientists can observe network activity while a simulation is running and interact with the simulation to ensure that dynamics stay within relevant regimes. In DEEP-EST, our focus will be on simulations of functional models of brain structure simulated using the NEST simulator[3] combined with two types of in situ analysis: computation of electrical local field potentials using the Arbor[4] and HybridLFPy packages[5] on the one side, and statistical analysis of spike activity using the Elephant package[6] on the other. NEST will be executed on the CM, Arbor/HybridLFPy on the ESB and Elephant on the DAM. NEST output will be communicated to the analysis packages using the MUSIC library[7].

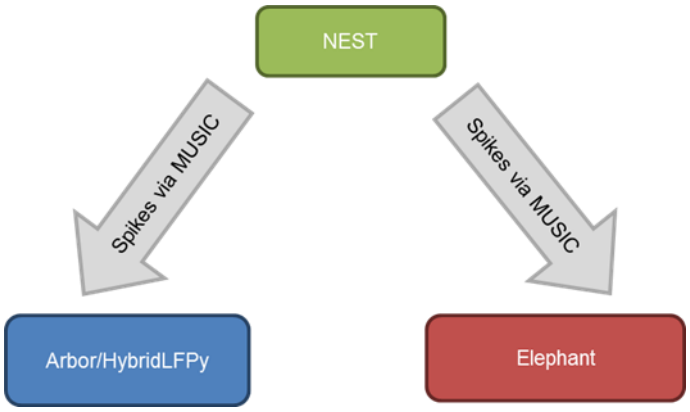

Figure 1: Neuroscience simulation and analytics workflow for NEST with in situ computation of local field potentials using Arbor and HybridLFPy (left path) and in situ statistical analysis using Elephant (right path).

2.1.1 NEST

NEST is a simulation code for the investigation of the dynamics of brain-scale neuronal network models, as for example the recently published multi-area model[8]. NEST operates on the level of resolution of neurons and synapses, where neurons are brain cells which are connected to each other by synapses.

The simulator considers brain tissue as an abstract assembly of nodes (neurons) and connections (synapses) or, in other words, a directed graph. The neurons in these simulations are point neurons, i.e. the state of a node changes according to a set of ordinary differential equations (ODE), without taking into account the complete morphology of the cell. The interaction between nodes is mediated by stereotyped events in the form of delayed delta pulses. These so-called action potentials (or spikes) are emitted by the nodes (neuronal activity) and propagated along the connections. The interaction strength (synaptic weight) can

3 http://www.nest-simulator.org/
4 https://github.com/eth-cscs/arbor
5 Hagen E et al. (2016) *Cerebral Cortex*, 26(12) pp. 4461–4496. doi: 10.1093/cercor/bhw237
6 http://elephant.readthedocs.io
7 Djurfeldt M et al. (2010) Neuroinform 8: 43. doi: 10.1007/s12021-010-9064-z
8 Schmidt M et al. (2018) Brain Struct Funct 223: 1409. doi: 10.1007/s00429-017-1554-4

either be static or dynamic (synaptic plasticity) and depends on the activity of the two neurons joined by the connection.

NEST does not implement a specific network model but provides the user with a range of neuron and synapse models and efficient routines to connect them to complex networks with on the order of ten thousand incoming and outgoing connections for each neuron. Concrete network models and the corresponding simulation experiments are specified by model description scripts. These scripts are written either in NEST's built-in simulation language SLI (based on PostScript) or using the Cython-based Python interface PyNEST[9,10], with PyNEST being the default interface.

A recently published example of a large-scale network model is the multi-area model, which is relevant also in the context of the DEEP-EST project. It is the first multi-scale model of vision related brain areas and comprises approximately 4 million neurons and 6000 incoming synapses per neuron, where neurons emit on average 14.6 spikes/s. Each individual area is represented by a modified version of the Potjans-Diesmann model[11], a microcircuit model corresponding to a cortical network under a surface of 1 mm^2. The microcircuits representing the areas differ in neuron numbers and connection probabilities. The minimal synaptic transmission delay in the network is 0.1 ms biological time, i.e., the time simulated in the biological system. This requires frequent MPI communication of spikes (every 0.1 ms biological time). In terms of wallclock time, MPI communication occurs at approximately 10–30 ms intervals, depending on the activity level in the neuronal network. Due to long transients in the network dynamics the model needs to be simulated for 100 s biological time.

The NEST code base is open source and under continuous development in order to enable the investigation of novel models and theories in Computational Neuroscience on the one hand and to meet the requirements of new computer hardware on the other hand. The NEST release 2.16 includes the NEST 5th generation simulation kernel (5g)[12], which achieves excellent scaling with respect to memory usage and good scaling with respect to runtime on the largest supercomputers currently available for academic research. The key step from the previous kernel used in NEST releases 2.6.0–2.14.0 to the 5g kernel is a new connectivity representation and spike exchange scheme using directed communication based on MPI_Alltoall().

2.1.2 Arbor/HybridLFPy

Arbor simulates compartmental neuron models. This means that the spatial structure of each neuron is represented as a spherical cell body (soma), to which an arbitrary number of dendritic trees are attached. Each dendritic tree consists of segments, i.e. tubes or cables, of a given length and radius; in the simulation, each segment is represented by a configurable number of compartments. Each segment is either connected to one other segment at each of its ends (linear cable) or to several segments at its far end (branching point; far end: end pointing away from the soma). Electric currents flow along the cables formed by the dendritic tree. This current flow is described by ordinary differential equations, with one set of equations for each compartment, coupled to neighbouring compartments. The main task of Arbor is to solve the resulting system of ODEs; this task is highly amenable to vectorisation. In addition, Arbor also

9 Eppler, JM et al. (2008) Front. Neuroinform. 2:12. doi: 10.3389/neuro.11.012.2008
10 Zaytsev YV and Morrison A (2014) Front. Neuroinform. 8:23. doi: 10.3389/fninf.2014.00023
11 Potjans TC and Diesmann M (2014) Cereb. Cortex 24, 785–806. doi: 10.1093/cercor/bhs358
12 Jordan J et al. (2018) Front. Neuroinform. 12:2. doi: 10.3389/fninf.2018.00002

transmits spikes between neurons via synapses; this mechanism is of lesser importance for our purposes because HybridLFPy is based on simulating the dynamics of disconnected compartmental neurons based on spike input generated by NEST.

HybridLFPy computes mesoscopic electrical brain signals, called local field potentials (LFPs) based on the network dynamics simulated using NEST. Specifically, spike trains generated by neurons in a NEST simulation, using highly connected point neurons are fed into detailed models of unconnected neurons simulated using Arbor to compute the electrical currents passing through the cell membrane at different locations. From these currents, HybridLFPy then computes the LFP at different locations in a piece of brain tissue using electrostatic principles. Currently, HybridLFPy is run only after a NEST simulation is complete. In the DEEP-EST project, our aim is to compute LFP signals in situ, while the network simulation is running.

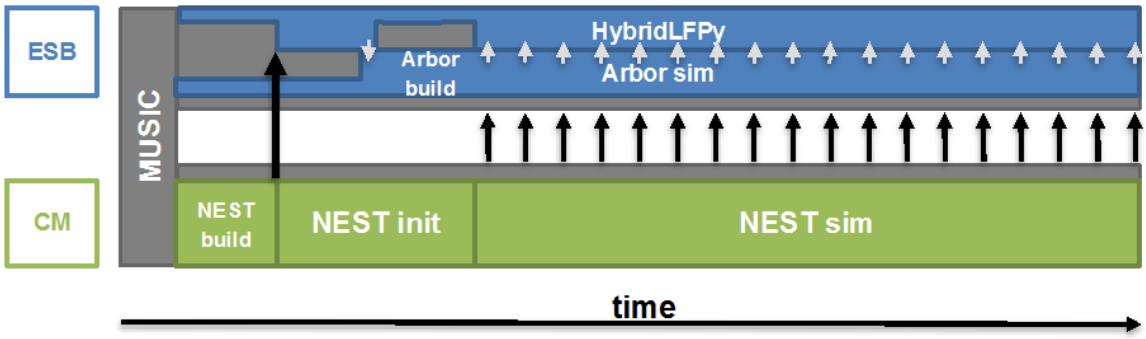

Figure 2: NMBU schematic workflow of NEST and Arbor/HybridLFPY in the DEEP-EST Modular Supercomputing Architecture (MSA). NEST runs on the CM and Arbor/HybridLFPy on the ESB. When the NEST build phase is completed, connectivity data is transferred from CM to ESB, where it is used by HybridLFPy to create the mapping between NEST spiking neurons and Arbor compartmental neurons. During the NEST simulation phase recent spikes recorded from part of the network need to be transferred from CM to ESB in short intervals, where they are fed to Arbor compartmental neurons. MPI communication is coordinated by the MUSIC library.

2.1.3 Elephant (ASSET)

Elephant is a pure Python library for the statistical analysis of spike activity of neurons. It can be installed using standard Python distribution tools. Elephant implements a wide and growing range of analysis methods. We focus mainly on the calculation of cross-correlations between spike trains and the detection of repeated patterns of spike activity across groups of neurons, so-called synfire chains.

Cross-correlations are detected using standard approaches, either implemented directly in Python or using NumPy convolution algorithms. Except for possible thread-parallelisation provided by the NumPy convolution implementation, cross-correlation algorithms are purely serial at present.

Detection of synfire chains uses the ASSET algorithm[13] in a recently optimised version[14], replacing the non-optimised version currently included in the release version of Elephant. The optimised algorithm uses MPI4Py for parallelisation.

13 Torre E et al. (2016) PLoS Comput Biol 12(7): e1004939. doi: 10.1371/journal.pcbi.1004939
14 Canova C et al. (2017) ASSET for JULIA: executing massive parallel spike correlation analysis on a KNL cluster. Poster presented at HBP Summit 2017.

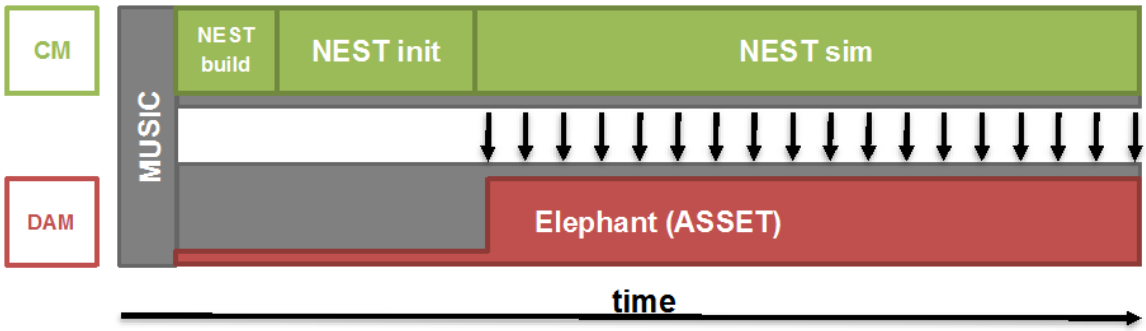

Figure 3: NMBU schematic workflow of NEST and Elephant (ASSET) in the DEEP-EST Modular Supercomputing Architecture (MSA). NEST runs on the CM and Elephant runs on the DAM. During the NEST simulation phase the latest spikes recorded from selected neuronal populations need to be transferred from CM to DAM in short intervals, where they are fed to Elephant. MPI communication is coordinated by the MUSIC library.

2.2 Application partitioning

2.2.1 NEST

Traditionally, NEST simulations have two distinct phases: a network construction (build) phase and a simulation phase. The key part of the build phase is the construction of network connectivity, i.e., building in largely random order a hierarchical data structure representing connections between neurons; each connection is represented only on the thread managing the connection's target neuron.

During the simulation phase, differential equations for the individual neurons are updated and spikes emitted according to a threshold criterion. Information on emitted spikes is exchanged between MPI processes and threads in steps of the minimal synaptic delay in the network, which is the maximum interval permitted by causality. Spikes are delivered to target neurons in parallel, each virtual process being responsible for delivery to the set of neurons it manages. This delivery process entails essentially random accesses to the connectivity data structure.

For the fifth generation (5G) kernel, we distinguish a third phase, called initialization phase, which comprises all necessary initialization processes at the beginning of a NEST simulation before the actual simulation takes place. In the NEST 5G kernel (NEST release 2.16), connectivity information, which is available only on the postsynaptic side after the build phase, needs to be transferred to the presynaptic side in order to enable directed communication of spikes during simulation. The transfer of connectivity data involves at least one round of MPI_Alltoall communication, which makes the initialization phase a non-negligible component.

In the benchmarks hpc_benchmark.sli and hpc_mam_benchmark.sli, build phase and initialization phase take up a significant amount of the total runtime as the neuronal networks are simulated only for one second of biological time. In simulations of the multi-area model, build phase and initialization phase require only a small fraction of the total runtime as the network is simulated for 100 s of biological time.

To enable the interaction of NEST with Arbor/HybridLFPy (see Figure 2), a small fraction of the connectivity details of the multi-area network, which is available after the build phase of NEST, needs to be communicated, where HybridLFPy maps the connectivity to the detailed neuron models.

During the simulation phase NEST needs to communicate spikes from a fraction of the neurons of the multi-area model to Arbor or Elephant. Communication takes place frequently and is coordinated by the MUSIC library (see Figure 2 and Figure 3). We estimate that the total amount of data that needs to be communicated from CM to ESB or DAM in each communication round is negligible (about 1kB if we assume communication every 0.1 ms of simulated time).

2.2.2 Arbor/HybridLFPy

NEST (on CM) and Arbor/HybridLFPy (on ESB) start to run at the same time (see Figure 2). While NEST constructs neurons and connections, Arbor instantiates neuron models. After the build phase of NEST, detailed connectivity information about the multi-area network is available. HybridLFPy requires part of this connectivity data in order to map the incoming connections of selected point-neurons simulated in NEST to their compartmental counterparts simulated in Arbor. Based on that, Arbor can build connections to the neuronal compartments.

After the communication of connectivity data from CM to ESB, NEST enters the initialization phase, which does not necessarily end at the same time as the Arbor build phase (for simplicity this detail is not shown in Figure 2). The simulation phases of both NEST and Arbor follow, where Arbor relies on frequent spike input from NEST.

During the simultaneous simulation phases of NEST and Arbor, full network activity of the multi-area model is simulated in NEST and spikes from the previously selected fraction of the network are frequently communicated to Arbor running on the ESB using the MPI-based MUSIC library. The spatially detailed (compartmental) neuron models simulated in Arbor consume the spikes according to the mapping created by HybridLFPy.

Locally on the ESB HybridLFPy requires frequent information about ionic currents into and out of the neuronal compartments simulated in Arbor in order to predict the LFP signals and their development over time.

2.2.3 Elephant (ASSET)

Elephant is fed with spikes from selected populations of the multi-area model using the MUSIC library to coordinate MPI communication (see Figure 3). Therefore, NEST (on CM) and the Python script that applies the necessary Elephant functions to the incoming spike trains (on DAM) start to run at the same time but the Python script needs to wait with the analysis until NEST reaches the simulation phase and produces spikes.

We expect that in simulations of the multi-area model this initial idle time of Elephant will be irrelevant as neither build nor initialization time, but the actual simulation time dominates the total runtime of NEST.

2.3 Application mapping

The simulation of the multi-area model with NEST is run on the CM using a hybrid parallelisation scheme combining MPI and OpenMP threads. CM is optimal for NEST, because NEST's irregular memory access patterns perform optimally on CPUs with large, low-latency RAM and because NEST does not benefit from vectorisation.

Selected neurons of the multi-area network are simulated in greater detail with Arbor running on the ESB, because Arbor requires considerably more compute power relative to memory,

since Arbor simulation does not require full network connectivity information. Arbor benefits significantly from vectorisation using AVX2, AVX512, and GPGPUs; it uses hybrid parallelisation combining MPI and C++11 threads or Intel TBB.

Analysis of spike trains recorded from selected populations of the multi-area model is carried out by Elephant, which runs on the DAM.

The changes in hardware design necessitated by Intel's discontinuation of the Knight's Landing processor architecture makes the application mapping for NEST, Arbor and Elephant less unequivocal than anticipated. We will review it systematically once test systems become available to ensure that each application runs on the module providing maximal overall performance.

3 Task 1.3: Molecular dynamics (NCSA)

3.1 Introduction

GROMACS is one of the fastest molecular dynamics simulators in the world. It is used mainly for soft matter molecular dynamics (MD) simulations with implementation in life sciences. GROMACS tracks the trajectories of a system of particles (atoms) that evolves in time by solving differential equations of motion at each time step. The coordinates and velocities of the particles are calculated by using the coordinates and velocities from the previous time frame. In each time step, one needs to calculate the forces acting on each atom, which is indeed the most time-consuming operation. Usually, pairs of atoms are defined in a predefined cut-off radius calculating short-range interactions, while the long-range interactions are calculated using Fast Fourier Transform (FFT) based algorithms.

3.2 Application partitioning

The particle-mesh Ewald (PME) algorithm uses FFT to solve long-range electrostatic contribution to real-space direct Coulomb sums. The reader should consider the fact that the specific implementation involving All-to-All MPI communications causes the performance scalability to drop. The latter can be solved by overlapping real-space calculations and Fourier-space calculations. In GROMACS the MPI ranks divided into two groups: one for real-space calculations (PP nodes) and the rest being dedicated to PME calculations (PME nodes). As shown in Figure 4, the resulting flowchart in an MD step can be described in the following manner. Each PP node has a corresponding PME node. At the beginning of the time step, each PP node sends coordinates and charges to its corresponding PME node and once the PME calculations are completed, each PME node sends the resulting forces back to the corresponding PP node. Meanwhile, all collective communications proceed only between PME nodes as well as only between PP nodes and overlapping the FFT All-to-All communications (exchanged between PME nodes only) with real-space calculations. Consequently, one must optimise the number of PP and PME nodes in such a way that PME nodes need to send the forces they have calculated in the exact moment when PP node need Fourier-space forces, energies, etc. The GROMACS tool called *tune_pme* provides the ability for the users to scan different combinations and start the simulation with an optimal PP/PME nodes ratio whilst the *mdrun* simulator further tunes the PME mesh and cut-off radius at the beginning of the MD simulation run.

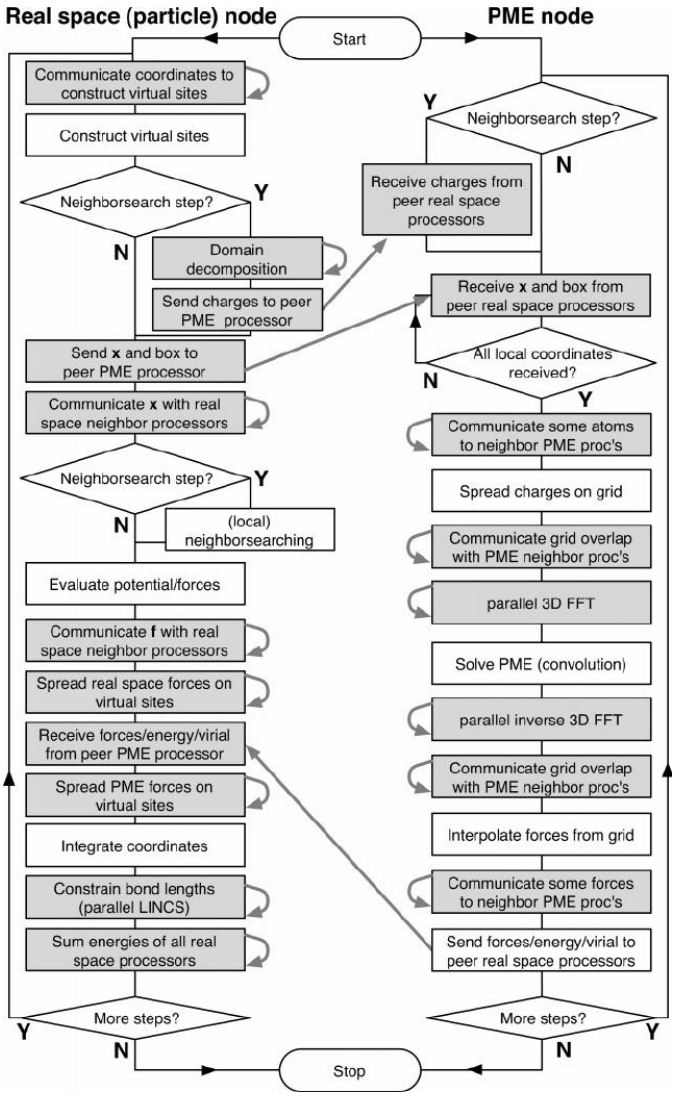

Figure 4: Flowchart for a typical simulation step for both particle and PME nodes. [Berk Hess*, Carsten Kutzner, David van der Spoel, and Erik Lindahl, GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation, Journal of Chemical Theory and Computation 2008 4 (3), 435-447]

3.3 Application mapping

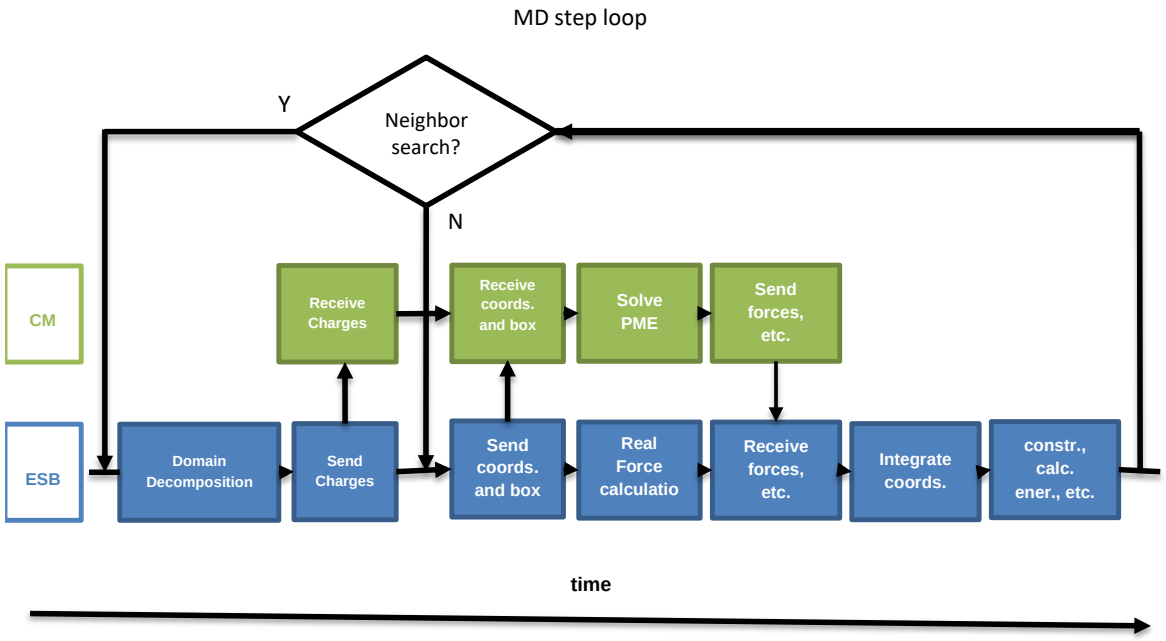

Figure 5: NCSA schematic workflow in the DEEP-EST Modular Supercomputing Architecture (MSA).

The two groups of nodes introduced above could be mapped to two modules of MSA type architecture:

- The PP nodes, which calculate pair forces over neighbour particles. GROMACS has SIMD kernels for efficient utilisation of vector operations offered by the DEEP-EST ESB module. MPI parallelisation would be used between processes running on CPU cores and groups of them. Finally, OpenMP parallelisation could also be used.
- A small number of PME nodes is preferable in order to reduce All-to-All MPI communications. Therefore, this part of the application should be run on a module which offers relatively higher performance per CPU core, i.e. CM or DAM modules. An optimal number of PME ranks would balance the calculation time of both PP and PME nodes. For this, hybrid MPI/OpenMP parallelisation is envisaged.

The resulting application mapping, which uses both the CM and ESB, is depicted in Figure 5 (see also Figure 4 for most important parts of a typical simulation step). The amount of hardware resources required to run a specific simulation should guarantee similar computation and communication times within groups of PP and PME nodes as well as between PP and PME nodes.

From a computer architecture point of view, the CM should have good node interconnect (for low latency all-to-all MPI communications) and high performance cores (e.g. frequency higher than 3 GHz). On the other hand, the CPUs integrating the ESB should have many cores with vector instructions, high memory bandwidth (suitable for loops over big arrays), and a node interconnect with good performance for point-to-point MPI communications. The interconnecting networks of the ESB and CM, as well as the ESB-CM interconnect, must have 100GB/s or even higher bandwidth.

3.4 Preliminary benchmark results

The idea of application MSA mapping was tested on DEEP-ER EXTOLL partition, which consists of DEEP-ER SDV nodes (Haswell) and KNL nodes with EXTOLL interconnect. A 325k atom system was used to measure GROMACS 2018 performance: it scales well on the SDV partition and poorly on the KNL partition.

In Figure 6 blue bars represent the performance of GROMACS on KNL nodes, orange bars on DEEP-ER SDV nodes and red bars on SDV nodes. In these real space calculations, i.e. PP nodes, used the KNL partition while reciprocal space calculations, i.e. PME nodes, used the SDV nodes. Each SDV node used 24 MPI ranks with 2 OpenMP threads per rank. Each KNL node used 64 MPI ranks and 2 OpenMP threads per rank. Two executables compiled for Haswell and KNL architectures were used respectively. The executable for KNL was compiled with MIC-AVX512 Intel compiler option to enable AVX-512 instruction set optimised kernels. The results of Figure 6 show a clear advantage when using the Cluster-Booster mapping.

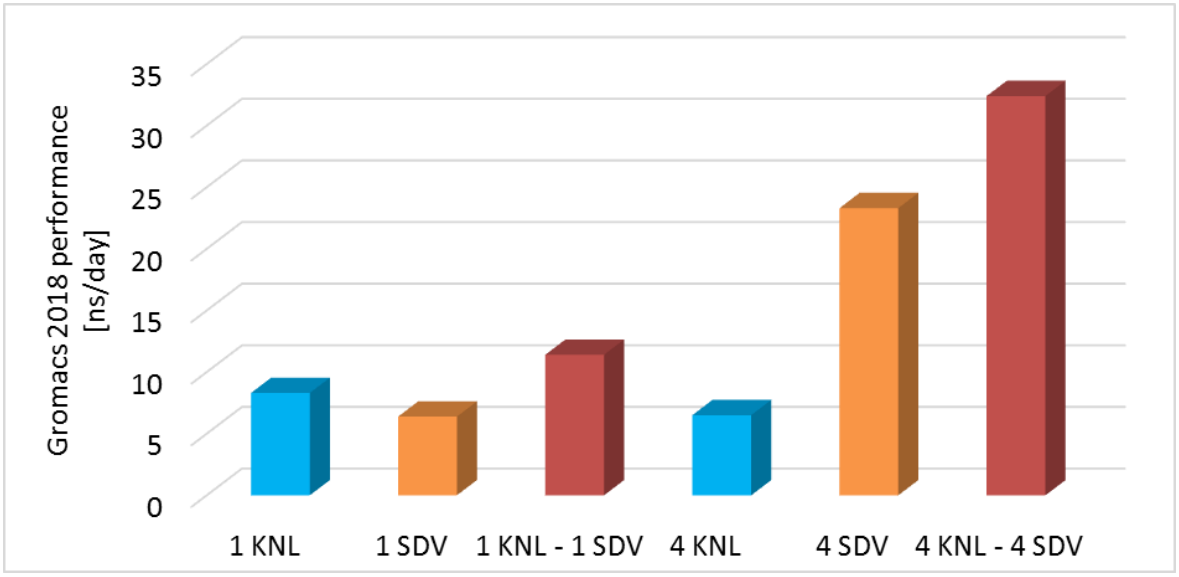

Figure 6: Gromacs 2018 performance of 325k atom test system on KNL partition (blue) only, DEEP-ER SDV partition (orange) only, Cluster-Booster mapping on KNL and DEEP-ER SDV partitions (red).

4 Task 1.4: Radio astronomy (ASTRON)

4.1 Introduction

ASTRON will explore two applications on the DEEP-EST Modular Supercomputer Architecture: a correlator and an imager. The correlator combines telescope data; the imager creates sky images.

Both applications are highly optimised for GPUs and CPUs. Yet, they run much faster on GPUs than on CPUs, albeit for different reasons. The imager performs a large number of sine/cosine operations, for which there is efficient hardware support on GPUs but not on CPUs. The newly developed GPU correlator takes advantage of the *tensor cores* of the latest GPU generation. Tensor cores are special-purpose mixed-precision matrix multiplication hardware with an exceptionally high performance: 71 TFLOP/s on correlations. Hence, both applications are tens of times faster on GPUs than on CPUs, and this affects the way we will use them on the DEEP-EST MSA.

Both applications are also being ported to FPGAs. FPGAs used to be programmed in Hardware Description Languages like VHDL and Verilog, which is difficult, time consuming and error prone, and not feasible for complex applications like the imager. New FPGA technologies (the OpenCL high-level programming language, hard Floating-Point Units, and tight integration with CPU cores) have changed this: they should reduce the programming effort of “simple” tasks like a correlator, and should allow complex applications like the imager. We explore Intel's OpenCL/FPGA technology to allow comparisons with GPUs (with respect to performance, energy efficiency, and programming effort), and to bring these technologies into radio astronomy.

Apart from the correlator and imager applications, there is a third task for which the computational resources of the DEEP-EST MSA are needed: creating FPGA designs. Compiling an OpenCL program for FPGAs takes long: typically 8-12 hours for the currently available Arria 10 FPGA, and probably even longer for the bigger Stratix 10 FPGAs that are planned for the DAM. We need to perform many of these compilations when doing performance optimisations for FPGAs, for parameter tuning and for compiling an application with different seeds that have a large and random impact on the clock frequency with which the application will eventually run. FPGA compilations do not only take long, they also require much memory (typically 50 GBytes for the Arria 10, for the Stratix 10 possibly more). In addition, using many CPU cores does not help, as only one to four threads are used throughout the compilation process.

Neither the correlator nor the imager run as distributed (MPI) applications, but rather as independent tasks on multiple nodes. They are trivially parallel, operating on different frequency bands or different observations.

4.2 Application partitioning

Both the correlator and the imager perform a series of operations that are described below. More details can be found in the application description document[15].

[15] https://bscw.zam.kfa-juelich.de/bscw/bscw.cgi/d2411360/application-description-ASTRON.pdf

4.2.1 Correlator

Figure 7: Correlator pipeline.

The correlator operates on streaming telescope data, normally in real time. As illustrated by Figure 7: Correlator pipeline. Figure 7, it first performs some filtering, then some corrections, and finally correlates (combines) the data from multiple receivers. The filter splits a wide frequency band into narrower frequency channels, like a digital prism that splits white light into colours. The filter consists of a Finite Impulse Response (FIR) filter and an FFT (the FIR filter reduces leaking into adjacent frequency channels). The correlator also performs a phase correction to the incoming signals (delay compensation, to follow a sky source) and an amplitude correction (bandpass correction, to correct errors introduced by another filter near the receivers). Finally, the correlator computes all correlations, which is the most compute-intensive step.

4.2.2 Imager

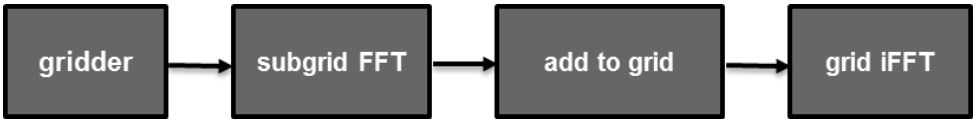

Figure 8: Imager pipeline.

The imager also performs a series of operations (see Figure 8). Incoming data is partitioned in small blocks that are convolved and gridded onto small subgrids. These subgrids are fast Fourier transformed (FFTed). The FFTed subgrids are added to a large grid (the Fourier transformed sky image that is being created). The grid is finally inversely FFTed to a sky image.

Note that the imager is also able to perform all steps in the backward direction; this is not shown in the figure.

4.3 Application mapping

Below, we will elaborate on the application mappings for FPGA compilations, the correlator, and the imager onto the DEEP-EST Modular Supercomputer Architecture (MSA).

4.3.1 FPGA compilations

The Cluster Module nodes are most suitable for creating FPGA designs, due to the high clock speed of their CPUs. Compiling for a FPGA does not require the FPGA to be present in the same machine. The DAM machines are also suitable to perform many concurrent compilations (due to their large amounts of memory), but we do not want to allocate DAM machines for extended periods of time without using their GPUs or FPGAs. The ESB has many cores but cannot perform many concurrent compilations because of the high memory usage of each compilation.

4.3.2 Correlator

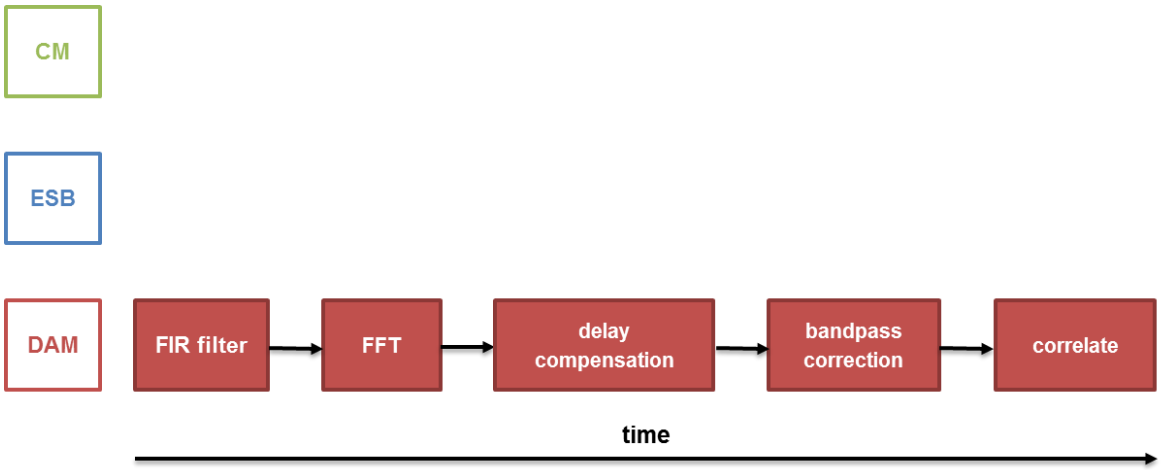

Figure 9: GPU correlator schematic workflow in the DEEP-EST Modular Supercomputer Architecture (MSA).

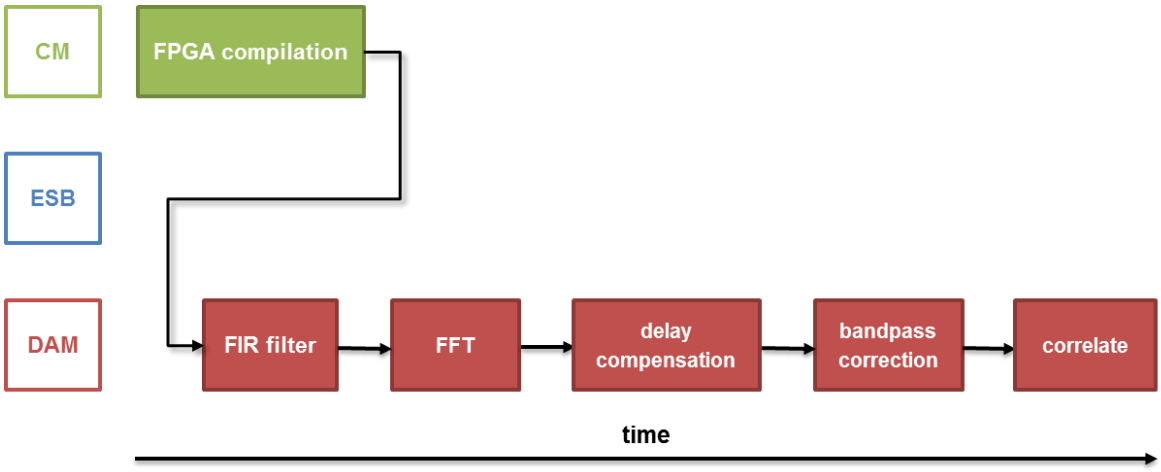

Figure 10: FPGA correlator schematic workflow in the DEEP-EST Modular Supercomputer Architecture (MSA).

Due to the high data rates between the pipeline components of the correlator (in excess of PCIe bandwidth limits), it is not logical to separate the operations and run them on different DEEP-EST module types. All operations will be performed consecutively on the GPUs or the FPGAs of the DAM modules (see Figure 9 and Figure 10, respectively).

4.3.3 Imager

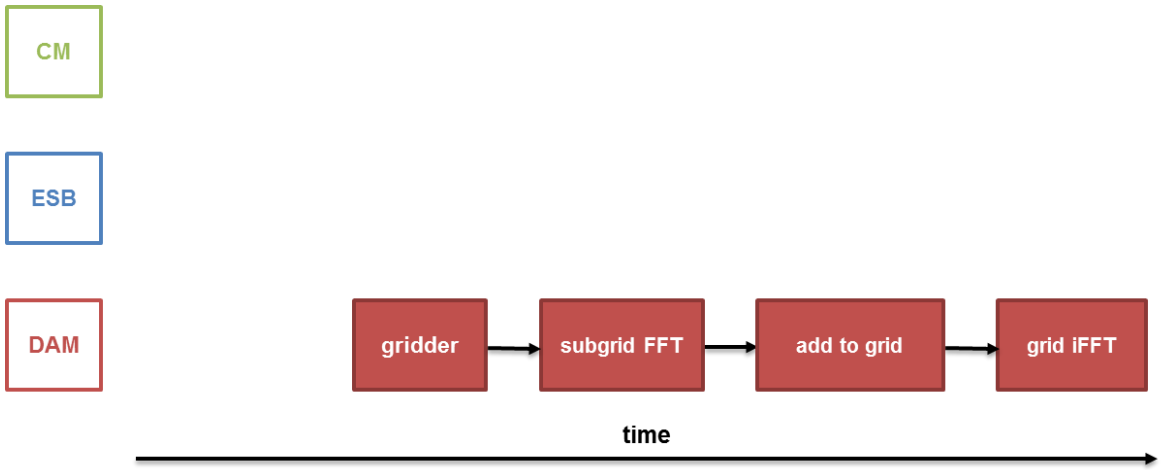

Figure 11: GPU imager schematic workflow in the DEEP-EST Modular Supercomputer Architecture (MSA).

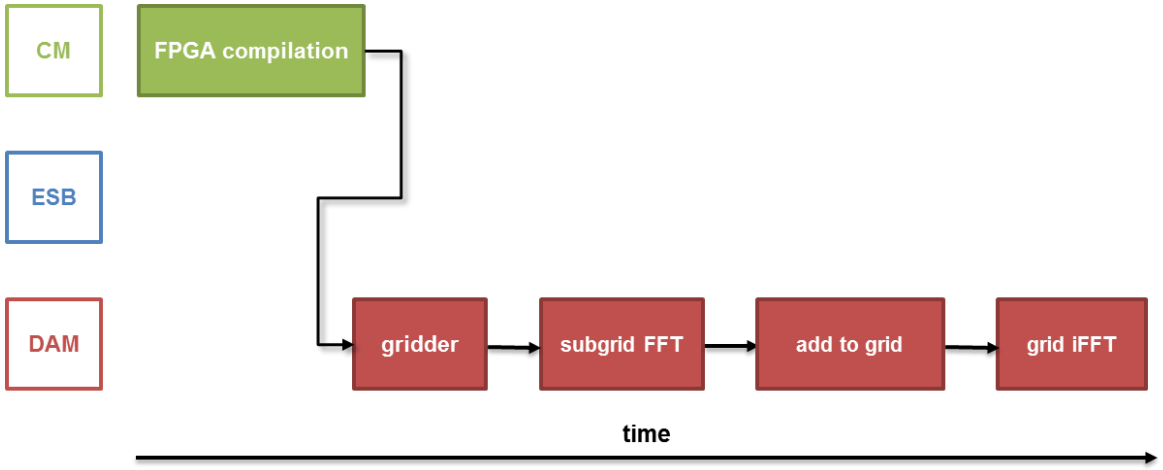

Figure 12: FPGA imager schematic workflow in the DEEP-EST Modular Supercomputer Architecture (MSA).

The gridder should definitely run on GPUs (or FPGAs) due to the large amounts of sine/cosine operations that it performs, hence it should run on the DAM (see Figure 11 and Figure 12, respectively). The subgrids can be Fourier transformed efficiently on the GPU as well. Addition of the FFTed subgrids to the grid can be done either on the GPU or host CPU of the DAM; in our experience, it is somewhat more efficient to perform it on the GPU. The best place to perform the final inverse FFT of the grid is not yet determined: indeed it seems to to depend on the image size and on the efficiency of the FFT library for a particular architecture. The best place can be the GPU, FPGA, or CPU of the DAM, or even one of the other DEEP-EST modules.

Another reason to run the imager on the DAM is the presence of 3D XPoint DIMM modules, which may be used to save huge sky images that do not entirely fit in DRAM. In this case, the DRAM transparently caches the “hot” parts of the grid that are stored in the larger-capacity 3D XPoint DIMMs. On top of that, the even smaller GPU device memory will be used to transparently cache the “really hot” parts of the grid, using NVIDIA's Unified Memory technology. The imager is a particularly interesting application to demonstrate the usefulness of transparent caching (both 3D XPoint and Unified Memory) as the access pattern to the grid becomes complex (but with sufficient locality).

5 Task 1.5: Space Weather (KU Leuven)

The Space Weather application from KU Leuven couples HPDA with HPC. The goal of the application is to perform more realistic simulations of the plasma environment of the Earth, given the current activity of the Sun.

In comparison with weather forecasting, our goal is to predict, days in advance, possible disrupting events on the Earth to avoid damage to our technology or to the health of exposed humans (in particular people aboard airplanes crossing the polar caps and astronauts). However, contrary to regular weather, in Space Weather we do not have “weather stations” situated between the Sun and the Earth. Satellites collecting in-situ data are mostly located around our planet. Information on solar activity is only available with the use of remote sensing equipment, including imagers and radiation detectors.

Machine Learning (ML) techniques are used in the present application to detect correlations between the images of the Sun, and plasma conditions measured near the Earth. Spacecraft data and solar images are freely available from multiple sources, including the OMNIweb data center, the Virtual Solar Observatory, the SDO and SOHO missions, the CACTus CME catalog, among many others. The full application can be divided in three main jobs:

1) The KU Leuven application downloads parts of this data to the SSSM. This data is then pre-processed to create a clean set of input and output files used for the training of the ML algorithm, using the DLMOS-DPP python package (where DPP refers to data pre-processing.). This pre-processed data is also stored in the SSSM for future utilization.
2) The input/output datasets are then used to train the ML model with the DLMOS-training python package. This training requires moving the datasets from the disks to the processors. The result is the generation of the coefficients of the corresponding trained Neural Network (NN). These coefficients are stored in a disk space accessible to multiple systems; we suggest using the NAM device for this task.
3) Using the latest images of the Sun and the latest trained NN model, the DLMOS-inference python tool creates the initial and boundary conditions for the xPic code. xPic is then launched in a Cluster-Booster mode in the CN and the ESB, to perform simulations of the Earth’s environment.

Figure 13 presents a global schematic view of the full Space Weather application. Arrows represent the data movement among the different elements and the colors identify the different modules of the DEEP-EST modular supercomputer prototype.

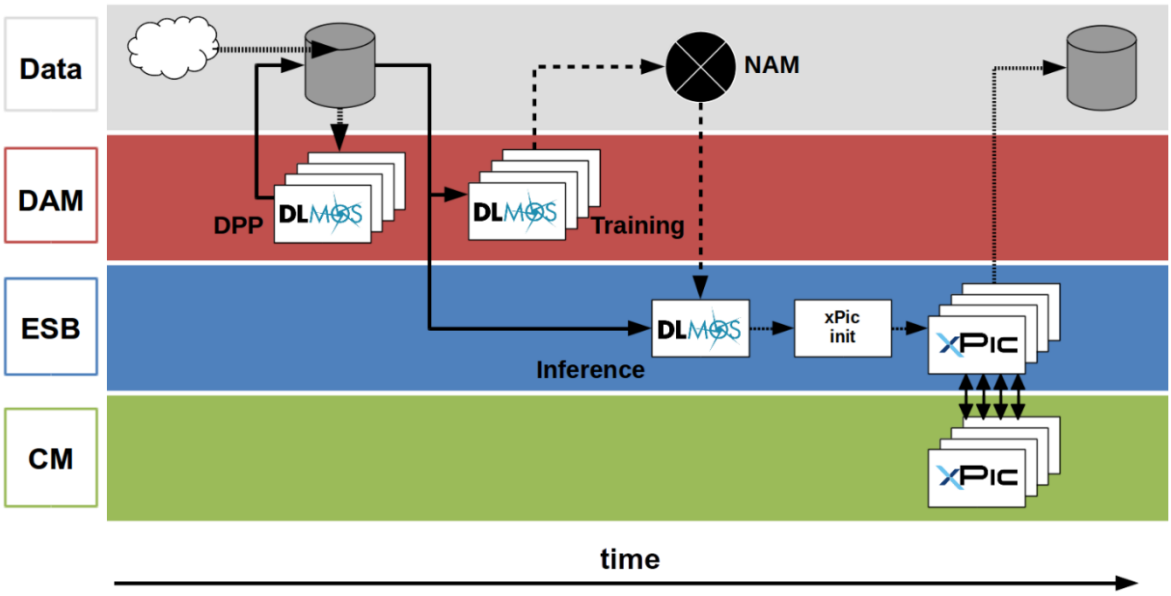

Figure 13: KU Leuven schematic workflow in the DEEP-EST Modular Supercomputing Architecture (MSA).

Most of the requirements of the two applications, DLMOS and xPic, have been described and included in deliverable D1.1, in particular the memory (D1.1-5.2.9) and I/O requirements (D1.1-5.2.10).

5.1 Application partitioning

5.1.1 DLMOS

The DLMOS software is a python package that contains multiple sub-packages. The three main objectives of the software are: 1) download and pre-process the data, 2) use pre-processed data to train a ML (NN) model, and 3) perform single forecasts given new solar images.

The code is then partitioned in three sub-packages:

a) DLMOS-DPP,
b) DLMOS-Training,
c) DLMOS-Inference.

The sub-packages are independent, but use data processed from the preceding package.

5.1.2 xPic

This Particle-in-Cell code has been partitioned into two solvers:

1) Field solver: a numerical algorithm that solves Maxwell's equations for electromagnetism in a 3D Cartesian grid. The solver uses an iterative Krylov subspace method to solve a large system of linear equations. This method requires the computation of a residual every Krylov iteration. Global communications are required to keep track of such residual and neighbour communications are required to compute the derivatives of the source terms of the linear system. Parallelisation is obtained by domain decomposition.

2) Particle solver: uses Newton's equations of motion to compute the movement of the billions of charged particles in the system. Collisions and other particle-particle interactions are not computed (their interaction is mediated through the electromagnetic fields). All particles are independent of each other. The particle solver is also parallelised by domain decomposition, so communications are required to move particles from one domain to the neighbouring domain when they cross boundaries. In addition to the movement of particles, the particle solver also performs the calculation of particle statistics called moment gathering. In this last step particle properties are projected on the 3D grid of the code and transmitted to the field solver.

The field solver and the particle solver are inter-dependent and require constant exchange of information. However, they show different numerical strategies that can be mapped to different hardware architectures.

5.2 Application mapping

5.2.1 DLMOS

The DLMOS application from KU Leuven is characterised by a heavy and continuous movement of data from the disk to the processor memory. It requires a good management of data movement between the different levels of memory.

This application also targets the use of such large amounts of data to train a deep neural network. This process is based on the constant use of tensor operations (matrix and vector multiplications) that can benefit from low compute power with large number of threads, and vectorisation. These operations are also relatively fast, so the memory bandwidth needs to be high. A more quantitative description of the requirements is not yet available.

These are the reasons why the DLMOS application from KU Leuven would benefit from the large number of cores and higher memory bandwidth proposed for the DAM.

5.2.1.1 DLMOS-DPP

The data pre-processing procedure is not based on highly parallel and multi-threaded, vectorisable, tensor operations. It requires high per core performance and high memory capacity. It also requires the full I/O infrastructure to move the data from disk to the processor. Data pre-processing can be implemented on accelerator cards and can also take advantage of the host processors of accelerator nodes (like the DAM nodes).

It would be possible to deploy the DLMOS-DPP sub-package to the CM where memory and sequential performances are higher. The final decision on the correct mapping of the DLMOS-DPP (DAM or CM) will need to be done on the prototype modules.

As it is today, it is more convenient to maintain the multiple elements of the KU Leuven application as close as possible in hardware, using the same module for the DLMOS-DPP and the DLMOS-Training sub-packages.

DLMOS-DPP will take advantage of the different levels of data storage of the DEEP-EST system, moving data from the internet, to the FS, then to the SSSM, the local disk, the local memory and finally to the processor. This package will generate new enhanced data that need to move in the opposite direction, up to the SSSM.

DLMOS-DPP can run continuously, processing new data, for multiple applications of multiple datasets. It does not depend on the results of any other components of the DLMOS package. Parallelism can be achieved by processing multiple inputs at the same time. No data communications are required in this package.

The pre-processing also involves the downloading of data to a local storage space, the detection of anomalies, the normalisation of the data, and the selection and building of the training datasets for the DLMOS-Training sub-package.

5.2.1.2 DLMOS-Training

Training is based on highly parallel multi-threaded vectorisable tensor operations that can be deployed on accelerator cards. These operations also require a constant stream of data, thus taking advantage of high bandwidth memory. For these reasons the DLMOS-Training sub-package will be mapped to the DAM.

This python code takes the processed data from the DLMOS-DPP, stored in the SSSM, and performs the training process of the DLMOS Neural Network Models (DLMOS-NNMs). The architecture of the DLMOS-NNMs is not yet defined and will be tested during this project.

Parallelism of the DLMOS-Training will be achieved in three different ways:

1) Model parallelism: Different accelerators will train different NNMs for the same training data, allowing to select and cross-breed the best performing models and achieve good convergence to a satisfactory result.
2) Data parallelism: A single dataset can be divided in multiple smaller datasets that can be used to train a model in independent accelerators.
3) Model parallelism: The NN model is divided in multiple subtasks that can be mapped to different accelerators.

The DLMOS-Training sub-package implements python algorithms for the parallelism methods 1 and 2, but the method 3 is implemented in the ML framework used (e.g. TensorFlow, Keras, PyTorch). For methods 1 and 2 network latency and bandwidth are not a constraint. The amount of data and the frequency of communications are very low.

Method 3 is very restrictive and requires constant data movement of small amounts of data. Parallel efficiency is achieved by the package communications algorithm which is not based on MPI communications. TensorFlow has shown in recent tests low parallel efficiency using multiple GPU node systems. We will study the use of other frameworks as PyTorch and MXNet as potential replacements of TensorFlow, in case parallel performances are unsatisfactory.

The output of the DLMOS-Training sub-package are the coefficients of all the matrices of the different NNMs. These coefficients will be stored in the NAM for easy and frequent access from multiple different nodes.

5.2.1.3 DLMOS-Inference

This procedure is similar to the DLMOS-Training sub-package but requires only one input (instead of hundreds of thousands) and produces only one output. The full procedure is extremely fast and does not require special hardware components. The sub-package is mapped to the ESB. The code takes inputs from two different sources: the SSSM, where recent input data has been stored by the DLMOS-DPP, and from the NAM, where the most recent NNMs, and their coefficients, are stored.

The DLMOS-Inference code is mapped to the ESB, because it is also the launching point of the xPic code. The inference process is very fast and requires minimum resources, so it is not necessary to use the DAM for this procedure.

DLMOS-Inference will generate the output that will be stored in local disk and used by the xPic initialisation tool to create the initial and boundary conditions of the code.

5.2.2 xPic

The code xPic will be run in the Cluster-Booster mode using the CM and the ESB. All I/O is performed from the ESB.

5.2.2.1 xPic initialisation

Data from the DLMOS-Inference will be read and interpreted by this python script that belongs to the xPic code. The script will create the initialization files for the code. It will write one field file (up to 1 GB of data for the largest cases) and a particle file (up to half a TB for the largest cases). The files will be written in the file system of the ESB. The files will be read from all the allocated ESB nodes.

5.2.2.2 xPic particle solver (booster)

The particle solver of xPic performs very fast calculations on a very large number of independent particles. The data of each particle is stored in memory aligned vectors in memory. Such vectors contain information about the particle location, velocity and charge. Memory aligned temporal vectors are used to store the projected values of the electric and magnetic fields on the particle.

All vectors are fitted in cache memory, aligned and ready for SIMD vector operations. The operations are basic multiplications and additions. All the conditions in this problem point to a system which is highly parallel and independent. This code will benefit from a system with a large number of cores with access to very high memory bandwidth. Following our past developments in the DEEP and DEEP-ER projects we decided to map the particle solver to the ESB.

The SLURM batch script calls the particle solver of xPic, pinning this executable to the allocated processors of the ESB. The particle solver can run using multiple OpenMP threads. Each MPI process in the particle solver spawns a child on the CM containing the executable of the field solver and creating an MPI interconnector between the parent and the child processes.

5.2.2.3 xPic field solver (cluster)

The field solver requires the resolution of an iterative linear system. It also requires performing finite-element differential operations on a Cartesian grid. The differential operations require the communication of data between neighbouring processors and the iterative method requires the global gathering of a residual value (the difference of the result between two iterations). These procedures are complex and require high performance in a single thread. Memory access is not necessarily cache optimised. Two communication patters stress the system in different ways. But the amount of data transferred between processors in not very high. The field solver of xPic is mapped to the CM to take advantage of the per-thread performance.

The field solver is spawned, from the particle solver, in the CM and receives information from the particle solver via MPI communications. The field solver does not perform any type of I/O. Communications between the field and the particle solver are performed using a point-to-point MPI intercommunication less frequently than communications inside each one of the solvers. The message size is about ten times the size of the Cartesian grid in each MPI process. For a typical run of 10x10x10 cells per MPI process, the message size between the CM and ESB modules is around 80 KB.

6 Task 1.6: Data analytics in Earth Science (UoI)

6.1 Introduction

The following subsections contain a brief introduction of each of the three data analytics approaches for selected machine learning applications driven by the University of Iceland that take advantage of the DEEP-EST modular supercomputing architecture.

6.1.1 HPDBSCAN

Highly Parallel Density-Based Spatial Clustering of Applications with Noise (HPDBSCAN) is an unsupervised clustering algorithm implementation tailored for high performance computing to cluster points of data. It is specifically used to cluster point-cloud datasets acquired using airborne and hand-held laser scanners. The algorithm is highly optimised, with fast HDF5 I/O, exhibiting both strong and weak scaling properties, but it has so far been limited to datasets which do not exceed a few GBs. In DEEP-EST, we are expanding the algorithm to support the clustering of arbitrarily large input datasets that do not fit in RAM. We will demonstrate the improved algorithm´s effectiveness by performing clustering on a point-cloud dataset in the range of several terabytes. This in turn will lay the foundation for Level of Detail (LoD) and Continuous Level of Importance (cLoI) studies on the dataset. These methods give researchers the opportunity to "zoom" into the big data, i.e. to study and analyse specific areas of the datasets using different resolutions, enabling the clustering of different attributes for the same area or object, e.g. form clusters of individual houses, or cluster the windows of a specific house.

6.1.2 PiSVM

PiSVM is a parallel Support Vector Machine (SVM) implementation using kernel methods and MPI for inter-process communications. It is a supervised learning algorithm we use to perform land-cover classifications on both hyper-spectral and multi-spectral remote sensing input datasets obtained by satellites. The datasets are labelled and have been feature engineered to get better classification accuracy and to speed-up the training process. The dataset sizes are of a wide range, ranging from several hundred MBs up to the terabyte scale, for example when considering time series of remote sensing images (e.g. change of land cover over time). The algorithm is currently undergoing improvement in terms of a fast-parallel HDF5 support and a Cascade SVM version to better scale for massive datasets. The prototype implementations are ready and are expected to be tested, integrated, and fully operational before Q4 this year.

6.1.3 Deep Neural Networks

We employ Deep Neural Networks (DNNs) for both supervised and unsupervised learning on multi-spectral remote sensing datasets collected by satellites such as those use by the PiSVM described above. Convolutional Neural Networks (CNNs), a special form of DNN, are trained to perform classification tasks or other unsupervised learning tasks on remote sensing image datasets ranging from several hundred megabytes to several terabytes. Model inference with an exclusive test dataset is used and an additional approach of transfer learning is one application goal as well.

6.2 Application partitioning

The work of each of the three applications can be broken down into partitions as described in the following subsections in order to take advantage of the modular supercomputing architecture by distributing the loads on several modules.

6.2.1 HPDBSCAN

The core partitions of HPDBSCAN 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.

6.2.1.1 Data pre-processing

In this phase, the point-cloud dataset is first spatially divided by a hyper-grid overlay, with equally sized cells of the same dimension as the points in the dataset. 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 but only being able to select any cell in whole, i.e. no partial cell selections are possible.

After the data has been divided, each chunk, with the addition of boundary halo cells from adjacent chunks, is processed sequentially by the HPDBSCAN algorithm.

6.2.1.2 Partition 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 as described in the section above, i.e. the hyper-grid cells are divided among processes, trying to keep the number of point comparisons for each process as close as possible.

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 able of being 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 passed 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

Previous versions of HPDBSCAN were not particularly robust as they 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 needs to be changed. We thus apply a simple but effective

measure to add resiliency to the HPDBSCAN 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, and allow the execution to restart after the last stored data. In effect, we are adding check pointing 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 data sets 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 detailed studies by scientists.

6.2.2 PiSVM

The application is 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 convergences, and to reduce the overall computation time by skipping features that have otherwise little or no effect on the training process. Therefore, feature engineering usually is performed with the goal of increasing the accuracy. But as different feature engineering techniques are usually applied, the input datasets change from time to time.

6.2.2.2 Training

PiSVM 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. A non-linear SVM is used in conjunction with RBF kernels (i.e. with 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. nicely parallel. 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 PiSVM is an embarrassingly parallel, i.e. nicely parallel, operation that performs predictions using an otherwise unseen labelled dataset which can be used to determine 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 deep learning application uses partly the same dataset as PiSVM for supervised learning to allow for a comparative study of the different machine learning 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.

6.2.3.1 I/O

The application uses principally the same input datasets as the PiSVM application does. However, it uses the raw, non-feature engineered datasets whilst PiSVM uses a processed version of it because feature engineering. The reason is that 'feature learning' is an intrinsic part of deep neural networks in general and CNN in particular. In future, also 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. 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 subsection are evaluated by measuring their prediction accuracy on previously unseen 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 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

The distribution to the different modular supercomputing architecture (MSA) modules described below represents specifically selected mappings. In fact, further mappings to the MSA modules are thinkable: machine learning techniques with training, testing, and validation offer many possibilities per application to use the characteristic of the MSA. In this sense, this deliverable can only capture a couple of possible application mappings. Note that the figures

abstract some important concepts used by our applications (e.g. NAM, persistent DIMMs, etc.) for the sake of providing a better overview, but these details are described in the text.

6.3.1 HPDBSCAN

We present two different HPDBSCAN workflow mappings, A and B, onto the DEEP-EST MSA, see Figure 14 and Figure 15 and their step-by-step description in the subsections below. We do not know yet, which workflow will offer HPDBSCAN more benefits, therefore we intend to implement and evaluate both workflows in order to be able perform a comparative study. Other mappings are possible but not described in detail here for the sake of conciseness, e.g. it could make sense to have the indexed data sets in persistent memory as well once indexed.

The main difference of the workflows A and B is that workflow A uses for data storage the Network Attached Memory (NAM) located in the Extreme Scale Booster (ESB) module, but workflow B uses the non-volatile memory (NVRAM) locally available in each node of the Data Analytics Module (DAM). This selection affects the rest of the workflow due to an intrinsic cost-factor associated with the gateway interconnects between the modules.

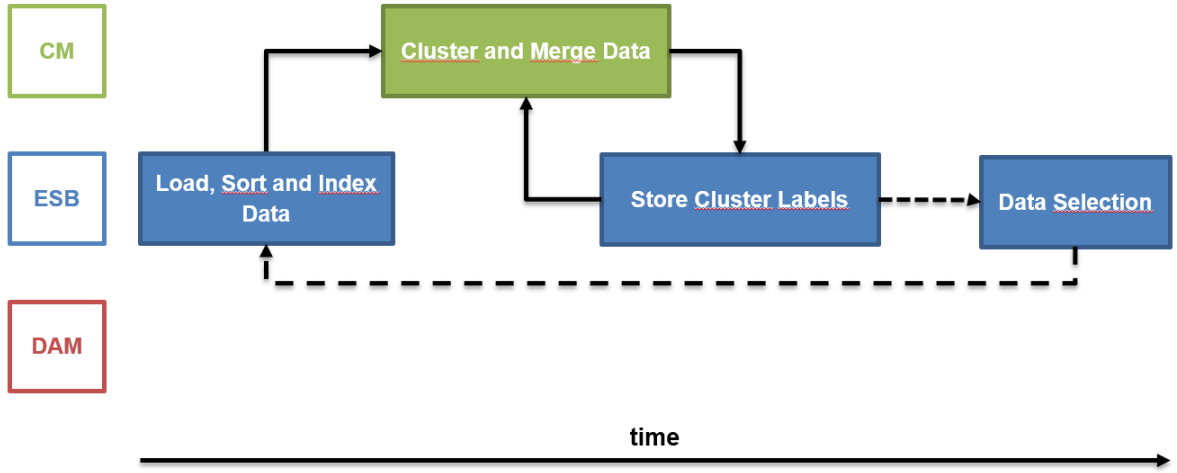

Figure 14: HPDBSCAN schematic workflow A in the DEEP-EST MSA. Note that the dashed lines represent optional flows.

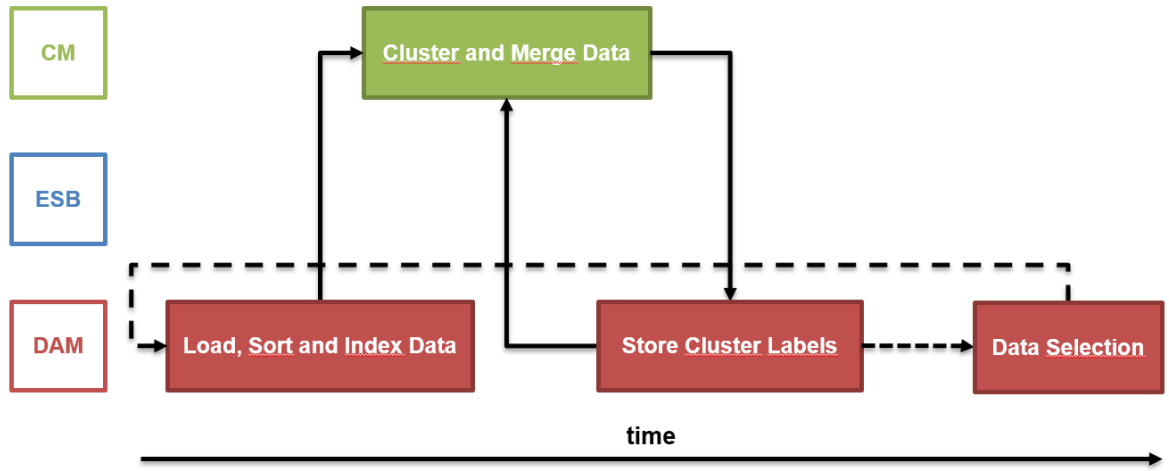

Figure 15: HPDBSCAN schematic workflow B in the DEEP-EST MSA. Note that the dashed lines represent optional flows.

6.3.1.1 Load, sort and index data

The dataset is loaded from the file-system in parallel by numerous processes where each process is assigned an equal sized chunk of data. Each data-point is sorted into its respective grid cell in the overlaying hyper-grid. Subsequently, the sorted data and relevant metadata is written to the persistent storage.

- Workflow A uses the NAM as a persistent memory target. Furthermore, it uses the Global Collective Engine (GCE) to speed-up MPI communication between the many-core processes in the ESB and the NAM.
- Workflow B uses the NVRAM in the DAM as a fast storage target, and its CPUs or optionally its FPGAs to sort the data.

6.3.1.2 Cluster and Merge Data

The Cluster Module (CM) reads a partition of the stored, sorted dataset in parallel using a load-balancing heuristic that assigns cells to processes (i.e. ranks) via MPI. Each process clusters its data locally, using as well OpenMP to speed-up the clustering in a hybrid parallel processing style. Finally, the clusters are merged together in parallel, using MPI collectives for inter-process communication. The CM module is best suited for clustering and merging, a computationally expensive and non-embarrassingly parallel task, as it contains the fastest CPUs and for non-embarrassingly parallel tasks a good interconnection too.

6.3.1.3 Store Cluster Labels

The obtained merged cluster labels are stored in the persistent memory, and if applicable, the clustering process is repeated in the CM with a new data chunk.

- Workflow A stores the computed cluster labels in the NAM, and as mentioned previously, uses the GCE to speed-up MPI communication as much as possible.
- Workflow B stores the labels in the NVRAM of the DAM.

6.3.1.4 Data Selection

In the case of Level of Detail (LoD) or continuous Level of Importance (cLoI), a subset of the dataset can be selected for further study.

- Workflow A performs the selection in the ESB module on the data stored in the NAM, possibly utilizing the NAM´s FPGA to further speed-up the selection process.
- Workflow B performs the selection in the DAM on the data stored in the NVRAM, possibly utilizing the FPGAs to facilitate the selection process.

6.3.2 PiSVM

As with HPDBSCAN, we will carry out a comparative study on two different possible mappings of the PiSVM application onto the DEEP-EST MSA, see Figure 16 and Figure 17 and their step-by-step description in the following subsections. The main difference between the two is that workflow C uses the NAM in the ESB module as a storage target and the CM for training the machine learning model, whereas workflow D uses the NVRAM in the DAM and uses the ESB module only for inference.

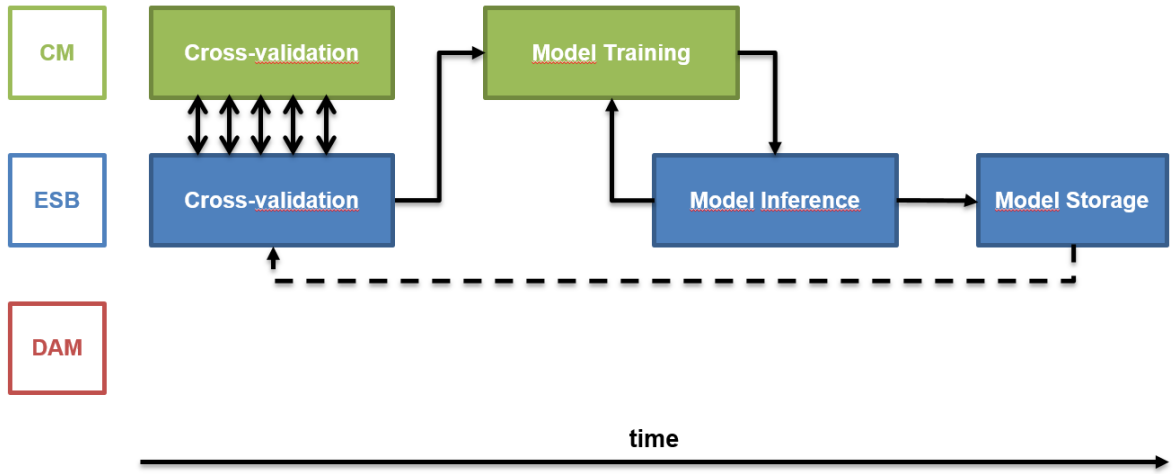

Figure 16: PiSVM schematic workflow C in the DEEP-EST MSA. Note that the dashed line represents an optional flow extension.

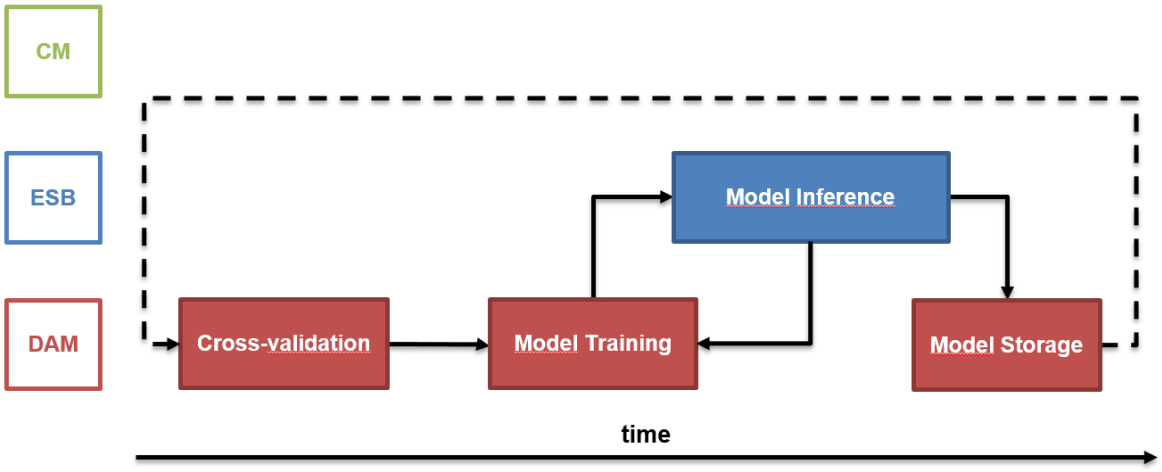

Figure 17: PiSVM schematic workflow D in the DEEP-EST MSA. Note that the dashed line represents an optional flow extension.

6.3.2.1 Cross-validation

Cross-validation determines the best parameters with regards to the maximum accuracy in all the folds across all parameter spaces.

- Workflow C performs the cross-validation using both the CM and the ESB module. Cross-validation is made up of computationally expensive mutually exclusive parallel operations, which therefore naturally leads us to select the CM with its powerful CPUs. However, the NAM in the ESB module will be used as a fast storage target, also including the possibility of taking advantage of the FPGA to select the best parameters values calculated by the CM.
- Workflow D performs the validation in the DAM with data stored in the NVRAM. It will explore the usage of the DAM´s FPGAs to enhance this phase.

6.3.2.2 Model Training

In this part, the two workflows differ substantially: our goal is to include a comparative study of powerful CPUs with gateway interconnects between different MSA modules versus slower

CPUs within the same module to determine limiting factors in the DEEP-EST heterogeneous MSA.

- Workflow C uses the CM for model training, putting its powerful CPUs to good use, and fetching data from the NAM in the ESB module.
- Workflow D continues using the DAM, which has slower CPUs but keeps the data stored in the local NVRAM, as well for training.

6.3.2.3 Model Inference

Testing with PiSVM is an embarrassingly parallel operation, i.e. nicely parallel. It does not require powerful CPUs, nor powerful interconnects, and thus the ESB module is well suited to fit the task, with its many-core CPUs.

6.3.2.4 Model Storage

Storing the models obtained in the earlier steps, differs for the two workflows mappings as follows:

- Workflow C stores both the input datasets and output models in the NAM, which resides in the ESB module.
- Workflow D stores both the input datasets and output models in the NVRAM, which resides in the DAM.

6.3.3 Deep learning

The deep learning application has two alternative workflows planned (see Figure 18 and Figure 19 and their subsequent description) where both use the DAM and ESB modules, but with different mapping to these modules. These two mappings will be compared and studied to determine the pros and cons with each approach, with the goal of creating an efficient execution pipeline to train and test neural networks on the DEEP-EST system.

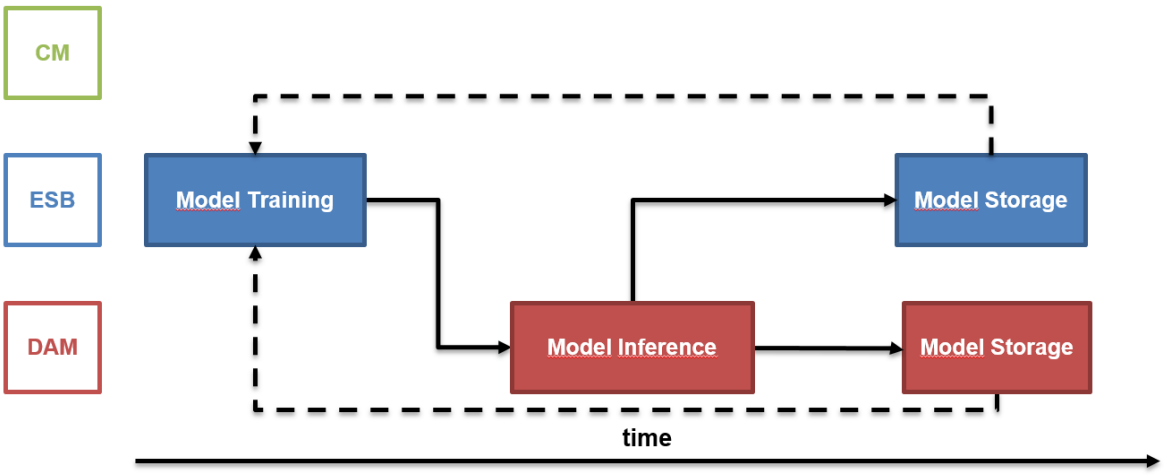

Figure 18: Deep neural network schematic workflow E in the DEEP-EST MSA. Note that the dashed lines represent optional flow extension.

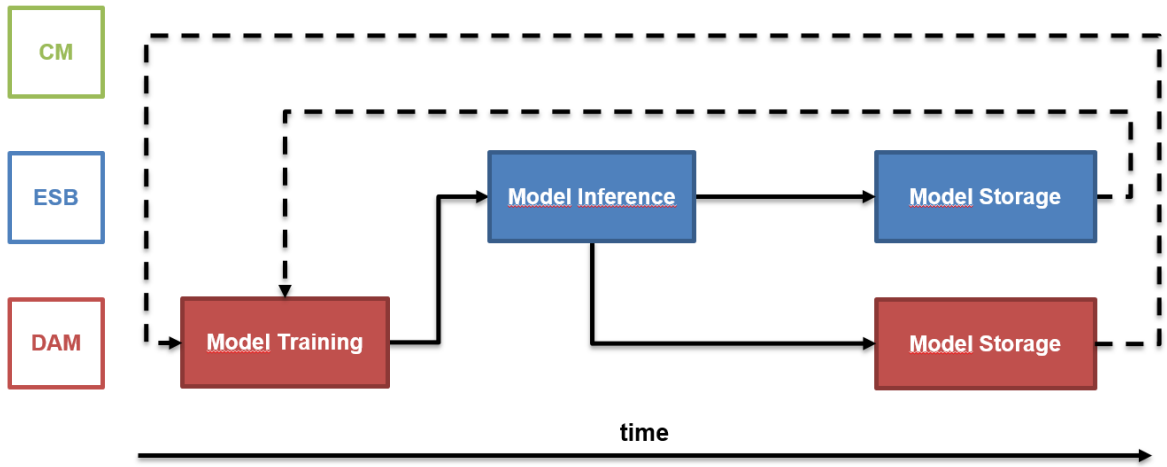

Figure 19: Deep neural network schematic workflow F in the DEEP-EST MSA. Note that the dashed lines represent optional flow extension.

6.3.3.1 Model Training

Training neural networks is a time-consuming task, in particular convolution and matrix operations benefit from powerful hardware accelerators such as GPGPUs and specialised software frameworks such as TensorFlow, Keras, PyTorch, or other. In DEEP-EST we will be comparing the usage of both CPUs and GPGPUs for training using two different workflows. We chose the TensorFlow platform with the Keras extension for our application as it combines ubiquity and simplicity with a strong support community.

- Workflow E stores the training samples in the NAM in the ESB module and therefore, it logical to use its many-core module for training, using an optimised version of TensorFlow intended for CPUs and AVX-512 instructions.
- Workflow F stores the training samples in the NVRAM in the DAM, and its powerful GPGPUs are perfectly suited for training neural networks.

Note that multiple models, with slight or major variations, can be trained concurrently using both workflows.

6.3.3.2 Model Inference

Model inference is done with an exclusive dataset used only for testing the model´s accuracy. Our plan is to execute inference in parallel to training, in a different module to better exploit the system´s computational capabilities.

- Workflow E uses the DAM´s GPGPUs for inference.
- Workflow F uses for inference the many-core CPUs with AVX-512 instructions as provided by the ESB module.

6.3.3.3 Model Storage

After training and inference, the models are either stored in the NAM of the ESB module, or in the NVRAM of the DAM. It is not clear at this moment which approach is better suited to enhance the application and therefore both will be explored and evaluated. This applies to both deep learning workflows, E and F.

7 Task 1.7: High Energy Physics (CERN)

7.1 Introduction

The Compact Muon Solenoid (CMS) detector is a general-purpose particle detector consisting of several components: tracker, electromagnetic and hadronic calorimeters, magnet and muon systems. Each component (usually addressable as sub-detectors) accomplishes a different task. For instance, tracker (both Pixel and Strip parts) is the closest sub-detector to the interaction point and responsible for identifying the trajectories of charged particles. Calorimeters measure energy depositions of the particles passing through.

Two different applications will be evaluated on the DEEP-EST Modular Supercomputer Architecture (MSA): CMS event reconstruction and event classification. Event reconstruction is the Compact Muon Solenoid Software framework (CMSSW) data processing pipeline aiming to reconstruct a full LHC collision event (more on CMSSW was provided in D1.2). Event classification is an analytics workflow, which aims to train several Machine Learning (ML) models and perform a multi-event classification.

7.2 Application partitioning

7.2.1 CMS Event Reconstruction

The process of reconstruction consists of three consecutively applied stages: digitization, local and global reconstruction. Each stage has a mix of GPU and CPU based algorithms. Some parts are being ported to OpenCL to be able to utilise the FPGAs to be provided with the MSA.

7.2.1.1 Digitization

Upon recording the response of the CMS detector, physics data is packed in a highly efficient binary format that requires unpacking before it can be dealt with. The actual content of this format is the raw electrical signals that correspond to the amount of digitized charge. Digitization is the first phase in the reconstruction chain.

7.2.1.2 Local reconstruction

In order to perform physics analysis, it is necessary to reconstruct actual physical quantities of interest that physicists can further work with. Therefore, digitized signals are converted (or reconstructed) into physical quantities such as energy, time and position. This conversion is performed on a per sub-detector base.

Local reconstruction applies to a particular component of the CMS detector (sub-detector). For instance, a hadronic calorimeter contains thousands of channels and energy deposition, within each is computed a sophisticated regression procedure. Regression algorithms are typically implemented using third party libraries (e.g. Eigen) which incorporate optimised linear algebra routines. However, certain functionality has to be manually ported to CUDA/OpenCL in order to preserve the algorithm itself and utilise heterogeneous resources provided with the MSA.

7.2.1.3 Global reconstruction

Global reconstruction is the process of combining information from several components of the CMS detector in order to build high level physics objects such as electrons, photons, jets, etc.

This operation drastically improves the precision of the measurements of properties of high level objects.

7.2.2 Analytics: multi-event classification

A typical Machine Learning (ML) pipeline consists of three phases: feature engineering, model training (including cross-validation) and evaluation (inference).

7.2.2.1 Feature engineering

In a typical ML application, input data does not correspond one-to-one to the model's input. Therefore, a certain transformation algorithm has to be applied in order to prepare the input in a certain format. Apache Spark is used to perform Extracting Transforming and Loading (ETL) operations. The transformation involves taking collections of various particles (photons, electrons, etc.) and building an abstract two-dimensional image representing an event.

7.2.2.2 Model training

The model training phase is usually the most time-consuming part of the analytics workflow. At the current stage, GPUs provide the highest performance.

7.2.2.3 Model evaluation

Upon completing the training phase and finding the appropriate hyper parameters, inference is performed. The input data needs to be split at the previous stage so that a classifier does not see the data on which the inference is to be performed. The goal is to find the model giving the highest classification accuracy.

7.3 Application mapping

7.3.1 CMSSW

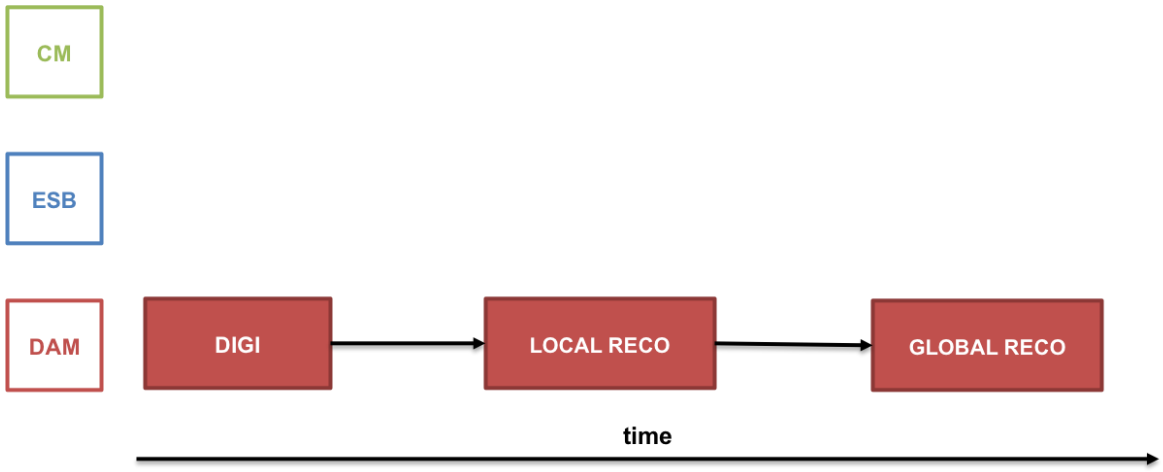

Figure 6: CMSSW Event Reconstruction in the DEEP-EST Modular Supercomputer Architecture (MSA).

Given that a typical data processing job is embarrassingly parallel, the absence of either inter-process (IPC) or remote-process (RPC) communication together with the goal of increasing the throughput by utilising heterogeneous resources implies that this application will be running only on the Data Analytics Module (DAM).

Current work on porting Pixel Tracking and Calorimeters algorithms to GPUs will allow exploiting the use of heterogeneous resources within the CMSSW reconstruction chain. The workflow contains internal hybrid parts and some of them run on the CPUs while others run on GPUs.

7.3.2 Analytics

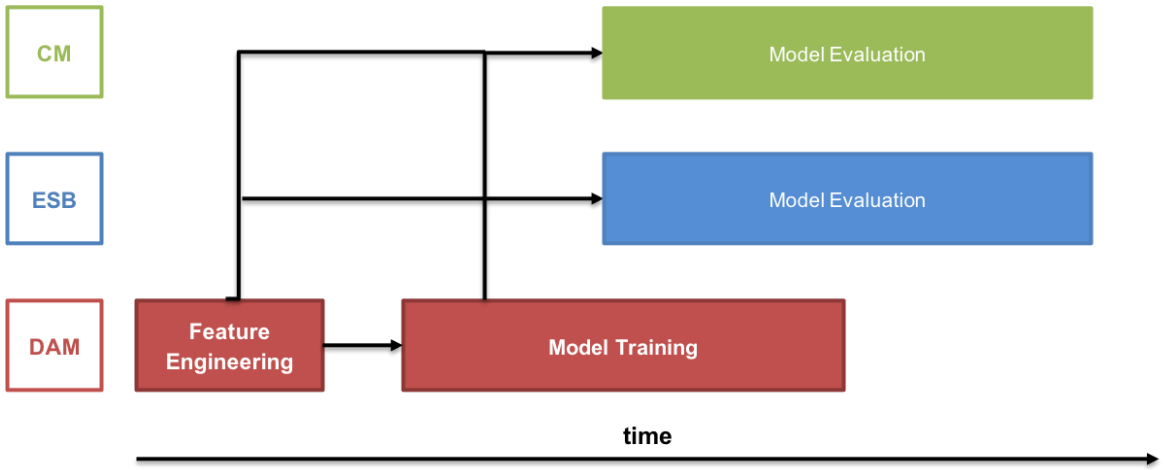

Figure 7: Event Classification in the DEEP-EST Modular Supercomputer Architecture (MSA).

The feature engineering step must run on the DAM due to the enlarged memory capacity and presence of 3D Point DIMMs. This will allow in-memory data analytics at a much higher throughput compared to other modules. Training part is targeted to run on the DAM due to the presence of GPUs. In the particular case of Deep Learning (DL) models, the training time is reduced significantly when using GPUs as they provide much higher level of parallelism for matrix operations.

The final step of the application is the model evaluation. The input dataset is split into several parts and, for inference, data not seen by the classifier must be used. Therefore, the input for the inference stage is the trained model itself and a subset of preprocessed features. It is worth mentioning that the model evaluation can start even before the training completes. The training of several models can be performed in parallel: some can finish training earlier and GPU resources should be left to the training stage only.

Furthermore, considering that model evaluation constitutes an embarrassingly parallel problem, the Extreme Scale Booster (ESB) with large number of core counts will be well suited for the last stage of analytics application. The most time consuming and compute intensive stage is the model training. As a consequence, GPU resources (in the DAM module) will be fully utilised by this step and, therefore, will not be available for the inference part. For the purpose of testing and evaluation, it will be informative to also consider using GPU resources for training and, at the same time, FPGA resources (from the DAM module) for inference, provided that the CPU cores (from the DAM module) are not fully saturated and are capable to feed both accelerators: this could prove an interesting option for Machine Learning workloads.

8 Global conclusion

The objective of this document is to detail how applications are mapped into the DEEP-EST's Modular Supercomputing Architecture. For this purpose, each application needs to be properly dissected into its logical parts matching a particular module of the MSA so they can be executed therein much more efficiently. This global conclusion gives a brief summary of the most important aspects regarding application mapping into the MSA:

- NMBU's interest on the MSA is the ability to perform "in-situ" analysis (post-treatment) of large-scale brain activity while the simulation runs in the background. In order to achieve this, the simulator application NEST will run on the CM and communicate through the MPI-based library MUSIC to either of the two post-processing applications Arbor/HybridLFP or Elephant (ASSET), which will run on the ESB or the DAM modules, respectively. This will demonstrate how the MSA allows real-time analysis as opposed to today's current practice that consists of running the analysis "only" after the simulation phase has finished.
- NCSA has identified two main executable parts of their application GROMACS. The first one, which calculates the motion of particles based on forces acting on them, uses efficient SIMD kernels for vector operations and therefore is most suitable for the many-core architecture that will integrate the ESB module. On the other hand, the second executable part of GROMACS performs three-dimensional FFTs that often work better on high-frequency cores, such as those found on the CM module (or DAM). Therefore, GROMACS will greatly benefit from running on the ESB and DAM modules simultaneously, which will result in an efficient overlapping of real-space calculations (on the ESB) with Fourier-space calculations (on the CM), leading to a significant decrease of the application runtime.
- Due to energy constraints and the seeking of maximum performance, GPUs and FPGAs fit best for ASTRON's applications; hence, their focus is mainly placed on the DAM. Nevertheless, the modularity of the MSA will allow overlapping FPGA design creation, carried out on the CM, with actual FPGA calculations performed simultaneously on the DAM. In addition, this will constitute an opportunity for ASTRON to experiment with new technologies integrated in the DEEP-EST project such as Intel's 3D Xpoint persistent memory in order to manage large sky images.
- KU Leuven's software environment consists of two applications, i.e. a deep learning application (DLMOS) and a numerical simulator (xPic), each of which can be partitioned in several executables targeting a different module of the MSA. KU Leuven contemplates to fully use the MSA: (i) data pre-processing and deep learning training is carried out in the DAM; (ii) these trained models will be executed on the ESB to create the input for the particles solver of the simulator xPic, also executed in the ESB; and (iii), the CM will be used for xPic's fields solver, which will be in constant exchange of information (via MPI) with the particles' solver running simultaneously on the ESB. Figure 13 clearly illustrates how the MSA will speed up KU Leuven's workflow.
- Overall the machine learning applications developed by UoI offer many choices to utilise the MSA. For example, two different workflows are envisaged for HPDBSCAN, one putting more emphasis on the ESB using local NAM, and the other on the DAM using local NVRAM. For the piSVM application another two workflows are proposed, which essentially differ on the way they fetch or store the data they manipulate. Finally, the deep learning application developed by UoI will be mapped into the ESB and DAM

modules considering two alternative partitions. All these six workflows are really interesting and clearly show the potential of the MSA for this type of machine and deep learning applications. Moreover, UoI is committed to explore and compare these various approaches in order to select the best candidates.

- The first workflow presented by CERN shares some similarities with ASTRON's approach: to get the maximum performance-energy ratio with the aid of GPUs integrated in the DAM. The second one is a typical data analytics workflow, which runs much more efficiently on GPUs (DAM). However, the last stage of data analytics, namely model evaluation or inference, can be carried out on either the CM or the ESB modules, thus allowing overlapping tasks to speed up computations, similarly to what KU Leuven proposes with their DLMOS application workflow.

8.1 Next steps

The next deliverable of WP1 entitled "Application initial ports" is due at month 24 of the project. During this period of one year, application developers must ensure the correct mapping of their applications into the DEEP-EST prototype. Moreover, results of application runs on equivalent hardware (SDVs) will also be detailed, comparing performance with previous versions of the code and other platforms using the benchmarks described in D1.2[2].

The successful porting of the applications during this period of time will involve a close collaboration with other work packages. In particular, with colleagues of WP5 who are in charge of (SLURM) job scheduling policies, in order to ensure the correct allocation of heterogeneous resources across different modules of the MSA. The cooperation with WP6 will also be crucial to ensure that applications use the programming environment being developed in the context of the DEEP-EST Project to fully support the MSA.

List of Acronyms and Abbreviations

A

AARTFAAC: The Amsterdam-ASTRON Radio Transients Facility And Analysis Center; a LOFAR-based, all-sky radio telescope

API: Application Programming Interface

ASTRON: Netherlands Institute for Radio Astronomy, Netherlands

B

BN: Booster Node (functional entity)

BoP: Board of Partners for the DEEP-EST Project

BSC: Barcelona Supercomputing Centre, Spain

BSCW: Repository used in the DEEP-EST Project to share all project documentation.

C

CERN: European Organisation for Nuclear Research / Organisation Européenne pour la Recherche Nucléaire, International organisation

CM: Cluster Module: with its Cluster Nodes (CN) containing high-end general-purpose processors and a relatively large amount of memory per core

CMS: Compact Muon Solenoid experiment at CERN's LHC

CMSSW: Compact Muon Solenoid Software framework

CN: Cluster Node (functional entity)

CNN: Convolutional Neural Networks

CPU: Central Processing Unit

D

DAM: Data Analytics Module: with nodes (DN) based on general-purpose processors, a huge amount of (non-volatile) memory per core, and support for the specific requirements of data-intensive applications

DDG: Design and Developer Group of the DEEP-EST Project

DEEP: Dynamical Exascale Entry Platform (project FP7-ICT-287530)

DEEP-ER: DEEP - Extended Reach (project FP7-ICT-610476)

DEEP-EST: DEEP - Extreme Scale Technologies

Dimemas: Performance analysis tool developed by BSC

DIMM: Dual In-line Memory Module

DSP: Digital Signal Processor

DN: Nodes of the DAM

DNN: Deep neural network

DRAM: Dynamic Random Access Memory. Typically describes any form of high capacity volatile memory attached to a CPU

E

EC: European Commission

ESB: Extreme Scale Booster: with highly energy-efficient many-core processors as Booster Nodes (BN), but a reduced amount of memory per core at high bandwidth

EU: European Union

Exascale: Computer systems or Applications, which are able to run with a performance above 10^{18} Floating point operations per second

Extrae: Performance analysis tool developed by BSC

F

FFT: Fast Fourier Transform

FMA: Fused Multiply Add; an operation of the form A * B + C

FP7: European Commission 7th Framework Programme

FPGA: Field-Programmable Gate Array, Integrated circuit to be configured by the customer or designer after manufacturing

G

GCE: Global Collective Engine, a computing device for collective operations

GFLOP/S: Gigaflop, 10^9 Floating point operations per second

GFLOPS/W: Giga ($10^{\wedge 9}$) Floating-Point Operations per Second per Watt, or alternatively: Giga Floating-Point Operations per Joule

GPU: Graphics Processing Unit

GROMACS: A toolbox for molecular dynamics calculations providing a rich set of calculation types, preparation and analysis tools

H

H2020: Horizon 2020

HBM: High Bandwidth Memory

HDL: Hardware Description Language

HPC: High Performance Computing

HPDBSCAN: A clustering code used by UoI in the field of Earth Science

HW: Hardware

I

Intel: Intel Germany GmbH, Feldkirchen, Germany

I/O: Input/Output. May describe the respective logical function of a computer system or a certain physical instantiation

J

JUELICH: Forschungszentrum Jülich GmbH, Jülich, Germany

K

KNL: Knights Landing, second generation of Intel® Xeon Phi™

KU Leuven: Katholieke Universiteit Leuven, Belgium

L

LHC: Large Hadron Collider (LHC), the world's most powerful accelerator providing research facilities for High Energy Physics researchers across the globe

LLNL: Lawrence Livermore National Laboratory

LOFAR: Low-Frequency Array, an instrument for performing radio astronomy built by ASTRON

M

MPI: Message Passing Interface, API specification typically used in parallel programs that allows processes to communicate with one another by sending and receiving messages

MSA: Modular Supercomputer Architecture

N

NAM: Network Attached Memory

NCSA: National Centre for Supercomputing Applications, Bulgaria

NEST: Widely-used, publically available simulation software for spiking neural network models developed by NMBU.

NMBU: Norwegian University of Life Sciences, Norway

NN: Neural Network

NUMA: Non-Uniform Memory Access

NVM: Non-Volatile Memory. Used to describe a physical technology or the use of such technology in a non-block-oriented way in a computer system

O

OmpSs: BSC's Superscalar (Ss) for OpenMP

OpenCL: Open Computing Language, framework for writing programs that execute across heterogeneous platforms

OpenMP: Open Multi-Processing, Application programming interface that support multiplatform shared memory multiprocessing

P

Paraver: Performance analysis tool developed by BSC

ParTec: ParTec Cluster Competence Center GmbH, Munich, Germany. Linked third Party of JUELICH in DEEP-EST

PCIe: Peripheral Component Interconnect Express; a bus that is often used to connect CPUs to GPUs, network devices, etc.

piSVM: Parallel classification algorithm

PME: Particle-mesh Ewald

PMT: Project Management Team of the DEEP-EST Project

R

RAM: Random-Access Memory

S

SCR: Scalable Checkpoint/Restart. A library from LLNL

SDV: Software Development Vehicle: HW systems to develop software in the time frame where the DEEP-EST Prototype is not yet available.

SIMD: Single Instruction Multiple Data

SIONlib: Parallel I/O library developed by Forschungszentrum Jülich

SKA: Square Kilometre Array

SSSM: Scalable Storage Service Module

SVML: The Short Vector Math Library

SW: Software

T

TCP: Transmission Control Protocol; a reliable, stream-based network protocol

TFLOP/s: Teraflop, 10^{12} Floating point operations per second

Tk: Task, Followed by a number, term to designate a Task inside a Work Package of the DEEP-EST Project

U

UDP: User Datagram Protocol; an unreliable, packet-based network protocol

UoI: Háskóli Íslands – University of Iceland, Iceland

W

WP: Work package

X

xPic Programming code developed by the KULeuven to simulate space weather