Modular Supercomputing for Neuroscience

Estela Suarez[1(✉)], Susanne Kunkel[2], Anne Küsters[3], Hans Ekkehard Plesser[2,4], and Thomas Lippert[1,5]

[1] Jülich Supercomputing Centre (JSC) - Forschungszentrum Jülich GmbH, Leo Brandt Strasse, 52428 Jülich, Germany
e.suarez@fz-juelich.de
[2] Faculty of Science and Technology, Norwegian University of Life Sciences (NMBU), Ås, Norway
[3] Simulation Laboratory Neuroscience – Bernstein Facility for High Performance Simulation and Data Analytics, Institute for Advanced Simulation, Jülich Aachen Research Alliance, Jülich Research Center, Jülich, Germany
[4] Institute of Neuroscience and Medicine (INM-6), Jülich Research Center, Jülich, Germany
[5] Frankfurt Institute for Advanced Studies (FIAS), Goethe-Universität Frankfurt, Ruth-Moufang-Straße 1, 60438 Frankfurt am Main, Germany

Abstract. The precise simulation of the human brain requires coupling different models in order to cover the different physiological and functional aspects of this extremely complex organ. Each of this brain models is implemented following specific mathematical and programming approaches, potentially leading to diverging computational behaviour and requirements. Such situation is the typical use case that can benefit from the Modular Supercomputing Architecture (MSA), which organizes heterogeneous computing resources at system level. This architecture and its corresponding software environment enable to run each part of an application or a workflow on the best suited hardware.

This paper presents the MSA concept covering current hardware and software implementations, and describes how the neuroscientific workflow resulting of coupling the codes NEST and Arbor is being prepared to exploit the MSA.

Keywords: Modular Supercomputing Architecture · MSA · Heterogeneous computer architectures · DEEP projects · Accelerators · Workflow · Neuroscience · Arbor · NEST

1 Introduction

Since the construction of the first cluster computer in the nineties [1], interconnecting a large number of commodity, general-purpose processors has become the most popular approach to build High-Performance Computing (HPC) systems. In recent years, these traditionally homogeneous clusters are being substituted by heterogeneous configurations employing a variety of acceleration technologies.

K. Amunts et al. (Eds.): BrainComp 2019, LNCS 12339, pp. 63–80, 2021.
https://doi.org/10.1007/978-3-030-82427-3_5

Computing devices are considered as *accelerators* when they have been designed to perform specific operations very fast. In principle, such definition would apply even to individual execution units within the CPU, such as tensor cores or advanced vector registers. However, in this paper we denote as accelerators only those out-of-package devices made of a very large number of relatively simple compute cores. Under this definition, the most frequently used example of an accelerator is a graphic processing unit (GPU). As their name says, GPUs were originally developed to very efficiently render and visualize graphics, but today their compute performance is also employed to perform floating point and tensor operations in all kinds of applications. Since accelerators are designed to execute auxiliary operations, they frequently depend on a CPU (considered as its host) to carry out important actions such as booting the accelerator and enabling it to communicate through the cluster-level high-speed network.

Accelerators rely on exploiting parallelism to compute, as their large number of compute cores/units are operated at relatively low frequencies. In consequence, they are able to achieve high peak-performances using less power than standard CPUs. Their energy efficiency – expressed through a high Flop/Watt ratio – is in fact the main reason for the success of accelerators in HPC. Clusters with accelerators are generally more energy-efficient than those without, and this difference becomes a major cost-factor in the operation of very large-scale platforms.

With regards to the system-level architectures of accelerated clusters, one can observe that typically one, two or more accelerators are integrated inside a node connected to a general purpose CPU via a PCIe interface. This node configuration is then multiplied several thousands of times, and the CPUs are interconnected with each other via some high-speed network. Recently, interconnection of the GPUs within the node has become possible, improving the ability to exchange data between them. In consequence the trend goes towards *GPU-islands* with four, six or even more GPUs per node. The negative consequence is a dramatic growth in compute power inside the accelerated-node, which is not compensated with a proportional increase in inter-node communication bandwidth. Therefore, though the scaling inside the node improves, the system-level scaling of codes is impeded by the imbalanced compute-to-communication capabilities between nodes.

The traditional programming model for node-level accelerated clusters is to run the main program in the host CPUs and offload compute-intensive kernels to the accelerators. For the large problems tackled by HPC, multi-node executions require exchanging data between the parts of the application running on the host and the accelerator. However, the static assignment of accelerators to CPUs within the node, added to the additional communication latencies that apply when inter-accelerator communication is executed through the host – what today is not always necessarily the case anymore –, limits the scalability and flexibility of this *node-level heterogeneous cluster* concept.

For example: an application running on a cluster with four GPUs per node might fully exploit the host CPUs but use only one of the GPUs attached to each one of them. In such case it will be hardly possible for other applications to use these free devices, as access needs to go through the CPU, which is busy.

In consequence, in this example almost 3/4 of the computation power of the system will remain idle. Obviously such situation would be avoided if the system is deployed with exactly the amount of accelerators per node required by the applications that will run on it. However, finding the right static configuration for all the users of the HPC system becomes impossible since the application portfolio is getting more and more diverse.

Today, the users of HPC systems employ codes ranging from high-scale, tightly coupled simulations, to high-performance data analytics (HPDA) and deep learning (DL). In fact, not only the applications are very different from each other, but even the workflows from individual users combine codes with very diverse requirements.

This is particularly the case in neuroscience, which aims at better understanding the behaviour of the possibly most complex organ in nature: the human brain. The huge scale spans (from nanometers to centimeters), the complexity of the involved physical and biological effects, and the tight interrelation between all of these aspects require the combination of various codes in order to reproduce the behaviour of the brain with some accuracy. All these codes present generally different requirements, making the overall usage scenario a natural candidate for using the a Modular Supercomputing Architecture (MSA).

The particular case addressed by this paper is the coupling of NEST and Arbor, two neuroscientific codes that can together bring a deep insight in the functions of the human brain. NEST simulations of large-scale networks of simple integrate-and-fire type model neurons are memory-bound due to the communication and memory accesses required to reproduce the exchange of neuronal signals, which dominate the total runtime [23]. Therefore, NEST runs best on general purpose clusters. On the other hand, Arbor simulates multi-compartment neurons with a very high computational cost per neuron and is therefore compute-bound, making it the ideal candidate to run on accelerators. The coupling of both codes could therefore profit from an MSA system in which CPU and accelerator resources can be reserved and allocated independently.

This paper explains the MSA and how a neuroscientific workflow combining the codes NEST and Arbor aims at employing it. Section 2 explains the architecture concept, while Sect. 3 and 4 describe current hardware platforms and their software environment. Section 5 describes the above-mentioned Nest-Arbor neuroscientific workflow, and the paper is summarized in Sect. 6.

2 The Modular Supercomputing Architecture (MSA)

The Modular Supercomputing Architecture (MSA) developed at the Jülich Supercomputing Centre (JSC) within the series of EU-funded *DEEP projects* [2, 3] aims at providing cost-effective computing capabilities fitting the needs of a wide range of computational sciences [4,5].

The MSA segregates heterogeneous resources and implements heterogeneity at system level, instead of node level (see Fig. 1). In its simplest configuration (the so-called cluster-booster approach [6]), a cluster made of general purpose CPUs

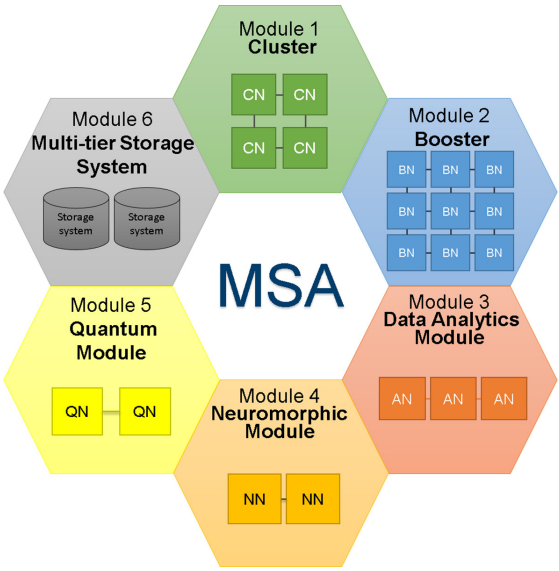

Fig. 1. Sketch of the modular supercomputing architecture. Note that this diagram reflects the general concept and does **not** represent any specific computer. Example modules: Cluster (*CN*: Cluster node), Booster (*BN*: Booster node), Data Analytics (*AN*: Analytics node), Neuromorphic (*NN*: Neuromorphic node), and Quantum (*QN*: Quantum node). For a schema of the MSA realization in the DEEP-EST prototype see Fig. 3, left.

is attached to a cluster of accelerators (the booster). In the latter accelerators are considered and operated as first-class computing devices. Furthermore, nodes on cluster and booster can be allocated independently and according to the needs of each application, so that no resources are blocked by allocating others.

In the first hardware realizations of the cluster-booster concept (e.g. JURECA, see Sect. 3.1), the booster used many-core processors that could boot and communicate through the system-level network without relying on host-CPUs. Fully autonomous accelerators are ideal for the MSA, as they enable scaling-up the cluster and the booster independently. In particular, the energy-efficient booster can be built at very large size (e.g. exascale), while the cluster is kept at a relatively small size to cover the needs of low-scaling parts of the applications without impacting on the overall power consumption of the system. Note that this is not possible in the traditional accelerated-node approach, where increasing the number of accelerators implies a proportional increase in the amount of general-purpose CPUs due to the static assignment between both.

Unfortunately, today's GPUs still rely on a host-CPU and cannot be operated autonomously. Still, the booster philosophy can be kept if one employs a low-power (and computationally weak) CPU, reducing its role to the orchestration and operation of the attached GPU(s). In this case, even if the number of CPUs increases when scaling-up the system, their contribution to the overall power consumption

is very small. Another key goal of the booster is achieving a good intra-node and inter-node communication-to-computation balance. If the selected GPU is computationally very strong, it might be beneficial to keep a low number of them per node (eventually only one), in order to fully exploit all its bandwidth towards the network. These kind of considerations are crucial to achieve the goal of the booster: good system balance and energy-efficient scalability at system level.

Note, however, that the MSA is much more general than the cluster-booster concept, which is very much focused on matching the different concurrency levels in applications. In the same way as the cluster provides hardware support to run the low/medium scalable part of codes while the booster does the same for highly-scalable codes, some applications need different acceleration technologies and varying sizes of memory devices and capacities. The MSA aims at fulfilling the needs of very diverse application requirements by interconnecting a variety of compute modules. Each module is a multi-node system of potentially large side, designed with specific hardware configurations that target a part of the application portfolio.

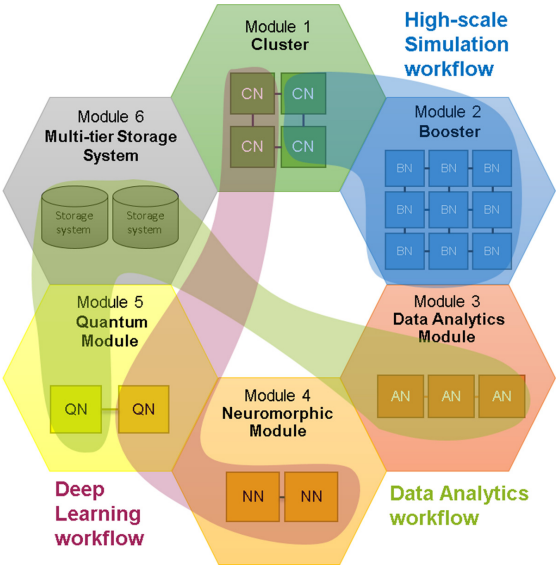

Fig. 2. Distribution of three different (hypothetical) workflows on an MSA system. See in Fig. 5 the mapping on the DEEP-EST prototype of the neuroscientific workflow matter of this paper.

One of the goals with MSA is to enable application developers to distribute their codes over a diversity of modules, such that each part of their workflows runs on the most suitable hardware (see Fig. 2). A further goal is to facilitate the adoption of new computing technologies in HPC. Therefore – though not yet implemented in existing platforms – the concept includes the option of integrating future technologies such as neuromorphic and quantum computing, providing

seamless integration into a traditional HPC environment in order to enable their use in scientific workflows.

3 Current Hardware Platforms

Several MSA platforms have been deployed at JSC. Here two systems are described, showing how the architecture itself evolves with time employing the newest available technologies.

3.1 JURECA Cluster-Booster

In November 2017, with the deployment of its booster module, the JURECA cluster-booster system became the first modular supercomputer worldwide to be listed in the Top 500 list, reaching position 29 with the Linpack benchmark running over both partitions [7]. Both modules can obviously be operated separately, but what makes JURECA unique is that complex applications can also run across both, using it as one MSA system.

While the cluster uses multicore processors (Intel Xeon *Haswell*) and 100 Gb/s Mellanox (EDR) InfiniBand, the booster uses multi-core processors (Intel Xeon Phi *KNL*) and 100 Gb/s Intel Omni-Path. Bridging the two different high-speed network technologies is possible in JURECA through a customized development in the ParTec ParaStation Software Suite [8,9], which is continuously researched and optimized.

3.2 DEEP-EST Prototype

The DEEP-EST project has built an MSA-prototype with three compute modules: cluster module (CM), extreme scale booster (ESB), and data analytics module (DAM) – see Fig. 3. The main hardware characteristics of each module are detailed in Table 1. It is worth noting that, unlike in JURECA, the DEEP-EST booster is built using an GPU attached to an x86 CPU. As mentioned in Sect. 2, the role of this host-CPU is reduced to an auxiliary function and it is not intended to employ it for application computing.

The DAM module is intended to run the parts of applications dealing with large amounts of data. Therefore, the DAM is provided with very large memory capacity, combining both volatile and non-volatile technologies. Codes that can particularly profit from such capabilities are those performing data-analytics, or running machine learning or deep learning algorithms. The latter benefit also from acceleration devices containing tensor cores and support for mixed precision. For this reason, the DAM contains both NVIDIA GPUs and Intel Stratix10 FPGA units. With its variety of components the DAM is the module offering maximal flexibility. This comes at the prize of a higher energy consumption. However, since the DAM is only used for small-scale problems its node-number can be kept low.

Additional to the three compute modules, the DEEP-EST prototype contains two storage modules: the all-flash storage module (AFSM) and the hard-disk

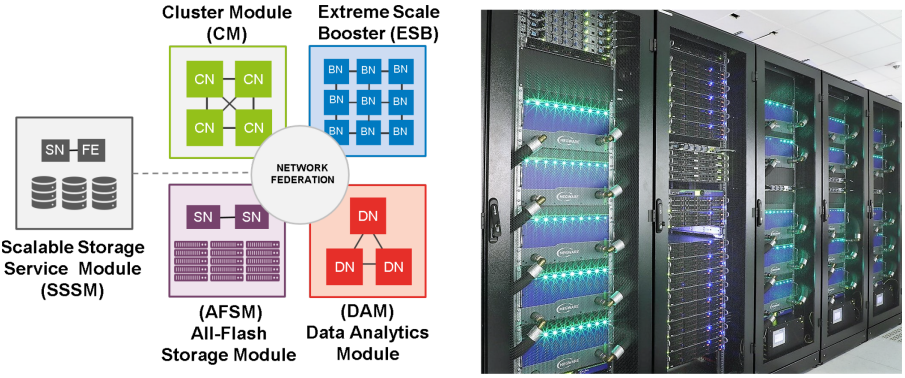

Fig. 3. Schema and picture of the DEEP-EST prototype at JSC (as of March 2021, fully installed).

Table 1. Key hardware features of the DEEP-EST MSA prototype.

DEEP-EST	CM	DAM	ESB
Time of deployment	2019	2019	2020
Node count	50	16	75
CPU type	Intel Xeon 6146	Intel Xeon 8260M	Intel Xeon 4215
CPU codename	Skylake	Cascade Lake	Cascade Lake
Cores @ frequency	12 @ 3.2 GHz	24 @ 2.4 GHz	8 @ 2.5 GHz
Accelerators per node	n.a	1× Nvidia V100 GPU	1× Nvidia V100 GPU
		1× Intel Stratix10 FPGA	
DDRA4 capacity	192 GB	384 GB	48 GB
HBM capacity	n.a	32 GB (GPU)	32 GB (GPU)
Node max. mem BW	256 GB/s	900 Gb/s (GPU)	900 GB/s (GPU)
NVM capacity	n.a	2/3 TB	n.a
NVMe/SATA SSD	512 GB	3 TB	250 GB
Power/node	500 W	1600 W	500 W
Cooling	warm-water	Air	Warm-water
Network technology	EDR-IB (100 Gb/s)	EXTOLL (100 Gb/s)	EDR-IB (100 Gb/s)
		Ethernet (40 Gb/s)	EXTOLL (100 Gb/s)
Topology	Fat-tree	Switched 2D-torus	tree/grid

based scalable storage module (SSSM), to enable fast I/O, run the file system and provide external storage capabilities.

All the compute and storage modules have been already installed and are up-and-running at JSC. The DEEP-EST prototype continues in operation beyond the end of the EU-funding time-frame and runs in near-production environment. It is being used for further development of the software stack and programming model of MSA systems in the DEEP-SEA project [2], as well as to run application tests to evaluate the benefits of its architecture and the functionality of its software stack.

4 Software Environment

The previous section showed how the MSA can be realized with very different hardware components. In fact, one could consider any heterogeneous computer as an MSA-system, as long as it can be operated in such a way that individual applications can run over various kinds of nodes, and these can be scheduled and allocated according to diverse application needs. Therefore, one could argue that the MSA is more a software infrastructure enabling the dynamic operation of a heterogeneous computer, rather than the hardware architecture of the system itself.

The MSA software stack enables application developers to map the intrinsic scalability patterns of their applications and workflows onto the hardware: highly parallel code parts run on the large-scale, energy-efficient booster, while less scalable code parts can profit from the high single-thread performance of the cluster, or from the high memory capacity of the data-analytics module. Users can freely decide how many nodes to use in each module, so that the highest application efficiency and system usage can be achieved [10].

4.1 Scheduling

The scheduling software used in the current MSA systems is SLURM [11]. Hardware heterogeneity is supported with SLURM's job-pack functionality, which provides semantics to express the amount of nodes to be reserved in each partition of an heterogeneous platform. The same annotation enables a user to run his/her workflow across nodes on different modules of an MSA system. However, in its current implementation SLURM statically reserves all nodes for the whole duration of the job, regardless of the fact that they are continuously used or not. For example, for a workflow performing pre-processing on the cluster and running then a long simulation on the booster, the nodes on the cluster will be kept reserved (and idle) until the simulation finishes in the booster. This is certainly not the wished behaviour on the MSA.

Extensions to the SLURM scheduler are therefore being implemented, aiming at reserving and releasing nodes for a job-pack when necessary. The DEEP-EST implementation relies on a new `---delay` switch, which can be used to inform the scheduler of the time-span between the start of one and the next job in a job-pack. Based on this information, the reservation of the second module can be started when it is actually needed, and not before. Further extensions to the scheduling and resource managing system for MSA are envisioned within the DEEP-SEA project.

4.2 Programming Environment

In order to facilitate portability, the MSA software stack aims at supporting the hardware complexity, while providing all the needed functionality and facing the application developers with the well-established interfaces and programming models that they know and use in other HPC systems. Therefore, the MSA programming paradigm is based on MPI. To support MSA-systems employing

different network technologies in different modules (such as JURECA) a gateway protocol has been developed [12]. For application developers, this protocol is fully transparent and hidden behind the MPI library.

The simplest way of running an application on an MSA system is using only one module. Monolithic, highly scalable codes will actually likely run this way. On the other hand, codes that perform multi-physics or multi-scale simulations can run across compute modules and exchange data between them via MPI. This is the scenario displayed in Fig. 4, where an MPI application running on the cluster spawns part of its code to the booster.

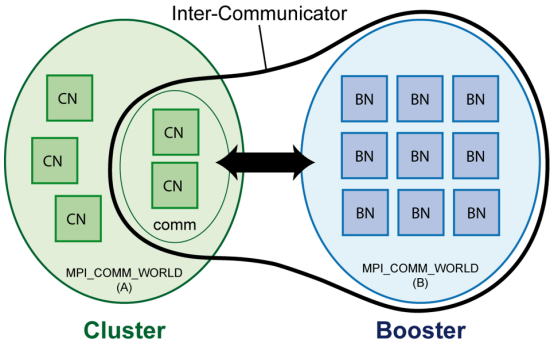

Fig. 4. Scheme of an MPI application running on the cluster and spawning part of its code to the booster.

MPI codes can be distributed over the MSA employing any of the collective instructions in the MPI standard allowing to connect two `MPI_Comm_World()` with each other. For instance, a subset of MPI tasks can collectively spawn a new `MPI_Comm_World()` to another module via the instruction `MPI_Comm_Spawn()`. Its inter-communicator connects the children to the parent processes and enables transferring data between them. Similarly, two `MPI_Comm_World()` running on different MSA modules can be connected with each other via the instruction `MPI_Connect()`. Furthermore, an `MPI_Comm_World()` can be split into two by using `MPI_Split()` and then send each one to a different module.

Arguably, splitting an MPI programm across modules is the most difficult way of using an MSA system. Distributing workflows is much simpler since one does not need to take care of the MPI communicators. In general, workflows use different codes used to execute different actions after (or in parallel) to each other. For example, a user may need to pre-process data before running a long simulation, then perform data-reduction, and ultimately visualize the final result. Running these codes on different modules consists simply on indicating to the scheduler on which nodes to execute each step. In the SLURM language a workflow is named a job-pack (see Sect. 4.1), and a set of SLURM instructions enables running each step on a different partition of module of an heterogeneous system.

Between the codes of a workflow data is currently transferred via the file-system, which means that it is written onto the external storage in one step,

and then re-read in the next workflow-step. Taking into account the time and power consumed in such write-read operations, this approach is not necessarily the fastest, and certainly not the most scalable. Because of that, the DEEP-EST project has investigated the potential implementation and benefits of directly transferring data between workflow steps via MPI. The data can reside directly at the node-memories or be stored in new kind of network-attached memory devices [13].

5 Neuroscience Workflow on MSA

Computational neuroscience, in its attempt to better understand the human brain via simulations, uses both multi-scale applications and complex workflows, and should therefore profit from the MSA concept. To prove it, the DEEP-EST project has studied the use of MSA for a neuroscientific workflow in which NEST and Arbor (described in Sects. 5.1 and 5.2, respectively) are the main components. A schema of the workflow distribution on the DEEP-EST prototype is given in Fig. 5.

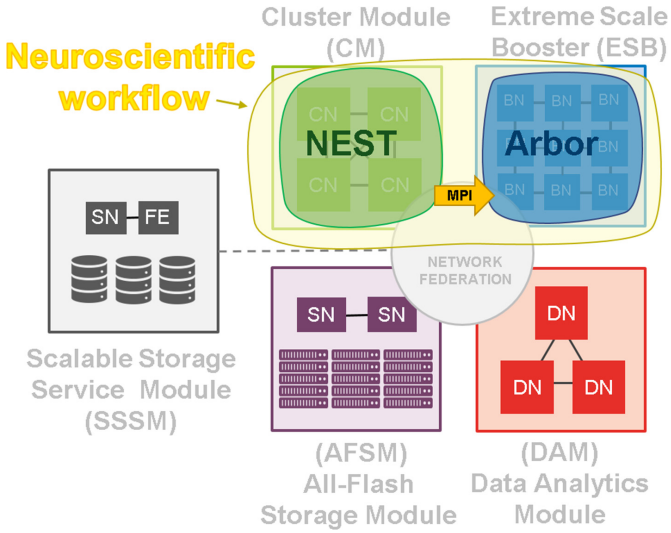

Fig. 5. Distribution of the neuroscientific workflow (NEST, Arbor) on the architecture of the DEEP-EST prototype.

Brain function involves the interaction of neurons located in different brain areas. Therefore, spiking neural network models representing multiple brain areas are becoming more and more popular in Computational Neuroscience. The multi-area model [14] is an early brain-scale model at the resolution of single neurons that incorporates experimental data defining the connections between neurons, called synapses. It comprises approximately four million highly simplified

model neurons and on average 6000 incoming synapses per neuron. Recording all neuronal activity – called spikes – for the entire duration of a simulation could easily reach the terabyte data-volume range. Interpreting such data in a meaningful way is even more challenging, also because experimental techniques currently can only record spike activity of a small proportion of neurons in a brain area, limiting the experimental data available for comparison to simulation results.

Neuroscientists therefore also record mesoscale signals such as local field potentials (LFP): A single micro-electrode or an array of micro-electrodes is inserted into the brain tissue in order to record the electrical activity, especially input currents, of all neurons in a volume roughly 1 mm in diameter. Due to the use of highly simplified point neurons in, e.g., the multi-area model [14], LFP signals cannot be obtained directly from simulations of that or similar models.

The Python package LFPy[1] enables the calculation of local field potentials by driving simulations of uncoupled compartmental neuron models by the spike output of point neuron network models [16]. This multi-scale simulation thus allows for the comparison of LFP signals from brain-scale network models to experimental data. As the simulation of brain-scale point neuron networks and uncoupled compartmental neuron models create very different computational loads, the prediction of LFPs from brain-scale models presents an important neuroscience application for MSA systems such as the DEEP-EST prototype (see Sect. 3.2).

With this target neuroscience application in mind we have investigated the limits to NEST-Arbor co-simulation on the DEEP-EST prototype. Figure 6 illustrates the concept, while Sect. 5.1 and Sect. 5.2 describe the involved simulators NEST[2] and Arbor[3], respectively. The overall runtime of the multi-scale simulation depends on the individual runtimes of the simulators and the latency of the frequent collective MPI communication between CM and ESB.

5.1 NEST

NEST is a simulator for spiking neural network models that focuses on the dynamics, size and structure of neural systems. In such networks neuron models are typically simple: they do not account for any neuronal morphology and the dynamics is governed by a small number of coupled differential equations, which in some cases can even be exactly integrated [17]. This enables the simulation of large-scale networks, where each compute node hosts many neurons and their incoming synapses. As in biologically realistic models of the cortex each neuron connects to a few thousand other neurons, an inherent bottleneck of the simulation of such networks is the frequent communication of neuronal signals (spikes) between compute nodes and the delivery of the spikes to their local targets.

Large-scale neural network simulations with NEST make use of a hybrid parallelization scheme combining MPI and OpenMP threads, where users typically

[1] lfpy.readthedocs.io; github.com/LFPy/LFPy.
[2] nest-simulator.org; nest-simulator.readthedocs.io; github.com/nest/nest-simulator.
[3] arbor.readthedocs.io; github.com/arbor-sim/arbor.

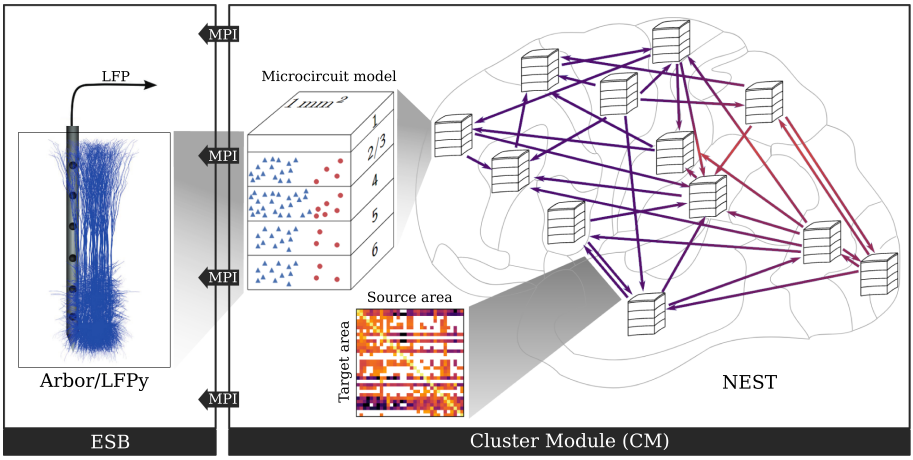

Fig. 6. Multi-scale simulation of a brain-scale network and concurrent calculation of LFPs as the target neuroscience application on the DEEP-EST prototype, requiring frequent transfer of neuronal activity data from the cluster module (CM) to the extreme scale booster (ESB) through collective MPI communication. *Right:* Simulation of the multi-area model [14] at single-neuron level resolution on the CM using NEST. Each area is represented by an adapted microcircuit model [15] with area- and layer-specific population sizes. Blue triangles and red dots in the magnified microcircuit-model illustration indicate two different types of neurons and their varying population sizes across layers. Connectivity between areas is based on experimental data and varies depending on source and target area as indicated by the connectivity matrix. Adapted from Fig. 1 and Fig. 4D in [14]. *Left:* Simulation of one of the areas at sub-neuronal resolution using Arbor on the ESB, and continuous calculation of LFPs using LFPy. Morphologies of the multi-compartment neuron models are based on experimental data. Neurons are not connected as all spike input is obtained from the multi-area model simulation on CM. Adapted from Fig. 1 in [16]

define the network models and steer the simulations through the Python based interface PyNEST [18]. A variety of neuron and synapse models are already included in NEST but it also offers the possibility to define custom models using the domain specific language NESTML [19]. NEST has an interface to the Multi-Simulator Coordinator (MUSIC) [20], which enables multi-scale simulations. Besides, NEST's refactored recording infrastructure [21] facilitates the definition of communication interfaces to other simulators such as Arbor. The NEST code is open source. All contributions to the code-base undergo review and are automatically tested by a continuous integration system running the NEST testsuite [22].

NEST is a simulator with versatile applications: from interactive explorations of small-scale networks on laptops to simulations of brain-scale networks on supercomputers. With the introduction of the 5g simulation kernel [23,24] the scalability of NEST has extended even further with respect to both runtime and memory usage, see Fig. 5.1.

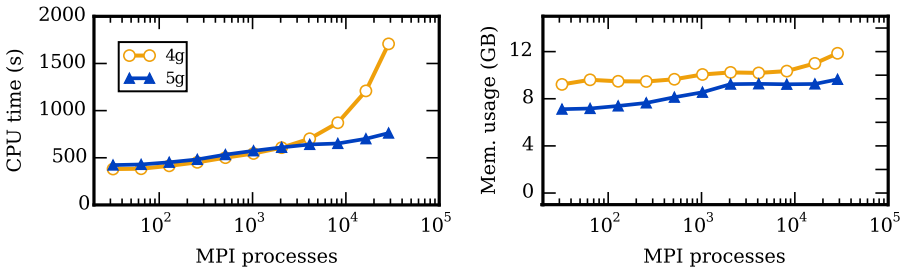

Fig. 7. Weak scaling of the NEST HPC benchmark on JUQUEEN for the current and the previous simulation kernel (NEST 5g and 4g, respectively). CPU time and memory usage per compute node for a network simulation for 1 s of biological real time, where each compute node hosts 18,000 neurons with 11,250 incoming synapses per neuron; 64% of all synapses have dynamically changing weights. Simulations were performed using 1 MPI process per compute node and 8 threads per MPI process. Adapted from Fig. 7 in [23].

The roadmap for the development of the simulation technology is defined by the NEST Initiative[4]. Current work comprises performance profiling and redesign of the algorithms underlying spike communication and spike delivery, and the development of more efficient ways of handling neuronal populations. This enables faster construction and simulation of highly structured networks such as the multi-area model (Fig. 7).

5.2 Arbor

Arbor is a performance-portable library for simulation of large networks of morphologically-detailed neurons on modern high-performance computing systems [25,26]. Arbor simulates networks of spiking neurons, particularly multi-compartment neurons. In these networks, the interaction between cells is conveyed by spikes and gap junctions and the multi-compartment neurons are characterized by axonal delays, synaptic functions and cable trees. Each cell is modelled as a branching, one-dimensional electrical system with dynamics derived from the balance of transmembrane currents with axial currents that travel through the intracellular medium, and with ion channels and synapses represented by additional current sources. Arbor additionally models leaky integrate-and-fire cells and proxy spike sources.

The Arbor library is an active open source project, written in C++14 and CUDA using an open development model. It can scale from laptops to the largest HPC clusters using MPI. The on-node implementation is specialized for GPUs, vectorized multicore, and Intel KNL with a modular design for enabling extensibility to new computer architectures, and employs specific optimizations for these GPU and CPU implementations. The GPU is deployed for updating currents and integrating gating variables using an optimized parallel GPU solver for

[4] nest-initiative.org.

sparse matrices with an optimized memory layout and reduced memory access. In detail, the GPU solver uses fine grained parallelization with one dendrite branch per thread, and a cell distribution into CUDA blocks to avoid global synchronization. In the simulation setup work balancing per thread avoids idle threads by sorting all submatrices on a level in a block by size. To maximize the utilization of the GPU memory bandwidth the memory layout is optimized by storing data in an interleaved format for each branch. Memory read access is reduced by storing only one parent compartment for each branch (Fig. 8).

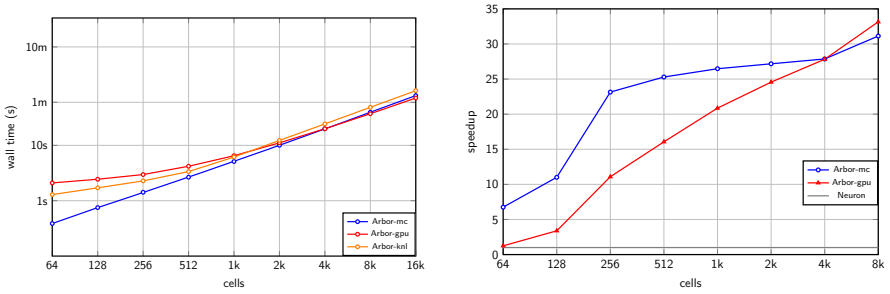

Fig. 8. Performance of Arbor. *Left*: Single node wall time of Arbor running on Piz Daint multicore, GPU and Tave KNL. *Right*: The single node speedup of Arbor running on Piz Daint multicore and GPU relative to NEURON on multicore. Adapted from Fig. 5 in [14].

By implementing the design goals of scalability, extensibility and performance portability, Arbor is an order of magnitude more efficient than existing simulation engines [25]. Arbor does this without sacrificing ease of use and flexibility. Arbor's single node speedup performance has been analyzed using a randomly connected network benchmark employing CSCS' Piz Daint multicore, GPU and KNL clusters. For more than 4,000 cells the GPU is utilized enough to run the benchmark more efficiently in terms of the wall time than on multicore or KNL (Fig. 5.2, left panel). Compared to NEURON [27], the most widely used software for general simulation of networks of multi-compartment cells, Arbor is 5–10× faster for fewer than 128 cells, and for more than 256 cells it is over 20× faster (Fig. 5.2, right panel).

Benchmarking and validation of Arbor and other simulators can be performed with the NSuite performance and validation testing suite[5] which is on-going work in Arbor development. Full support for the SONATA [28] model exchange format is under active development, as well as a Python API. Arbor will provide APIs for integration with other tools and simulators, including co-simulation with NEST. On a technical level a NEST-Arbor two-way coupled co-simulation imply some specific challenges, e.g., enabling injection of external spikes, as well as new initiation steps to align time delays and the number of external cells.

[5] nsuite.readthedocs.io; github.com/arbor-sim/nsuite.

6 Summary

The Modular Supercomputing Architecture (MSA) proposes a new philosophy for the integration and use of heterogeneous computing resources. Instead of regarding acceleration devices as intra-node entities and using them to speed-up very concrete kernels of the codes executed on general-purpose host-CPUs, the MSA strives for operating accelerators as first-class, autonomous compute elements. The MSA segregates the heterogeneous resources into individual modules, each one being a multi-node platform of potentially large size tailored to specific kinds or parts of applications. Each module can be sized differently, according to the energy efficiency of the hardware and the needs of the users. At the same time, applications and workflows can be distributed over different modules using the overarching MSA software environment, enabling each step to be executed on the best suited hardware.

The field of Computational Neuroscience is already preparing to employ the MSA approach, targeting the DEEP-EST prototype with a workflow that combines the codes NEST and Arbor. This multi-scale neuronal network simulation connects two types of neuronal simulations that are fundamentally different in computational load, memory access behaviour, and communication requirements.

A simulation of the multi-area model with NEST is not particularly computationally costly as it involves only the update of simple model neurons. In such large-scale network simulations it is rather the frequent and unpredictable exchange of neuronal signals that imposes stress on MPI communication and memory access, and thus dominates the total runtime. The cluster module is therefore best suited for this type of simulation.

For the Arbor simulation of multi-compartment neurons the computational costs per neuron are much higher than for a large-scale point-neuron network simulated with NEST. At the same time, the communication of spikes is of minor importance for the overall runtime of the Arbor simulation because it is much smaller in terms of number of neurons. In the planned application, the compartmental neurons are not even connected and communication of neuronal signals within Arbor is thus not required. Therefore, the Arbor simulation is more compute bound and can benefit from the GPUs of the DEEP-EST booster.

The installation of the DEEP-EST prototype has been completed with the deployment of its third and last module – the booster – in early 2020. NEST and Arbor have been adapted to run on the prototype, to show the benefits of executing an important neuroscientific workflow across the modules of an MSA platform. Adaptations to NEST and Arbor include the development of corresponding interfaces for spike exchange between the simulators, using MPI laying the groundwork for the neuroscience workflow (Sect. 5). Co-simulation benchmarks show that spiking network simulations with NEST running on the cluster module can be extended such that spikes generated in NEST drive compartmental neurons simulated with Arbor on the booster module without runtime penalty [29]. Moreover, the simulation time of NEST has been significantly reduced by optimizing the spike-delivery algorithm hiding memory fetch latency [29], which

contributes to more efficient co-simulation. We consider the optimizations a generally useful contribution to large-scale network simulation as they are applicable to other simulators for pulse-coupled networks with high connection degrees.

Acknowledgements. S. Kunkel and H.E. Plesser thank the NEST developer community and Arbor developers for excellent collaboration. A. Küsters would like to thank Wouter Klijn and Ben Cumming for contributing to Sect. 5.2. With many thanks to the Arbor developers Nora Abi Akar, Benjamin Cumming, Felix Huber, Wouter Klijn, Alexander Peyser and Stuart Yates, as well as the SimLab Neuroscience at the Jülich Supercomputing Centre.

E. Suarez thanks all the institutions and individuals involved in the DEEP projects, who have contributed to the development of the MSA architecture, its prototype hardware implementations and its software environment.

Funding. This work has been partially funded by the European Union's Seventh Framework (FP7/20017-2013) and Horizon 2020 Framework Programmes for Research and Innovation under grant agreements 287530 (DEEP), 610476 (DEEP-ER), 754304 (DEEP-EST), 720270 (HBP SGA1), and 785907 (HBP SGA2). The present publication reflects only the authors' views. The European Commission is not liable for any use that might be made of the information contained therein.

The Arbor library is developed by the Swiss National Supercomputing Center and the Jülich Supercomputing Center under the auspices of the Human Brain Project, funded from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No. 720270 (Human Brain Project SGA1) and Specific Grant Agreement No. 785907 (Human Brain Project SGA2).

The authors also gratefully acknowledge the funding provided by the Helmholtz Programme *Supercomputing & Big Data* to realize the JURECA Cluster and Booster.

References

1. Becker, D.J., Sterling, T., Savarese, D., Dorband, J.E., Ranawak, U.A., Packer, C.V.: BEOWULF: a parallel workstation for scientific computation. In: Proceedings International Conference on Parallel Processing, vol. 95 (1995). http://www.phy.duke.edu/~rgb/brahma/Resources/beowulf/papers/ICPP95/icpp95.html
2. DEEP projects. http://www.deep-projects.eu
3. Eicker, N., Lippert, T., Moschny, T., Suarez, E.: The DEEP project - an alternative approach to heterogeneous cluster-computing in the many-core era. Concurrency Comput. Pract. Experience **28**, 2394–2411 (2016). https://doi.org/10.1002/cpe.3562
4. Suarez, E., Eicker, N., Lippert, T.: Supercomputer evolution at JSC. In: Proceedings NIC Symposium, vol. 49, p. 1–12 (2018). http://juser.fz-juelich.de/record/844072
5. Suarez, E., Eicker, N., Lippert, Th.: Modular supercomputing architecture: from idea to production. In: Vetter, J.S. (ed.) Contemporary High Performance Computing: From Petascale Toward Exascale, ch. 9, vol. 3, pp. 223–251. CRC Press (2019). https://juser.fz-juelich.de/record/862856
6. Eicker, N., Lippert, T.: An accelerated cluster-architecture for the exascale. In: PARS 2011, PARS-Mitteilungen, vol. 28, pp. 110–119 (2011)

7. Krause, D., Thörnig, P.: JURECA: modular supercomputer at Jülich supercomputing centre. J. Large-Scale Res. Facil. **4** (2018). https://doi.org/10.17815/jlsrf-4-121-1
8. ParaStationV5. http://www.par-tec.com/products/parastationv5.html
9. Clauss, C., et al.: Allocation-internal co-scheduling - interaction and orchestration of multiple concurrent MPI sessions. In: Advances in Parallel Computing, vol. 28, pp. 46–68. IOS Press BV (2017)
10. Kreuzer, A., et al.: Application Performance on a Cluster-Booster System. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (2018). https://doi.org/10.1109/IPDPSW.2018.00019
11. SLURM. https://slurm.schedmd.com/
12. Eicker, N., Galonska, A., Hauke, J., Nüssle, M.: Bridging the DEEP gap - implementation of an efficient forwarding protocol. Intel European Exascale Labs - Report 2013, vol. 1, pp. 34–41 (2014). http://juser.fz-juelich.de/record/171982
13. Schmidt, J.: Network attached memory, Chapter 4 of the Ph.D. thesis, Accelerating Checkpoint/Restart Application Performance in Large-Scale Systems with Network Attached Memory, Ruprecht-Karls University Heidelberg (Fakultät für Mathematik und Informatik). http://archiv.ub.uni-heidelberg.de/volltextserver/23800/1/dissertation_juri_schmidt_publish.pdf
14. Schmidt, M., Bakker, R., Hilgetag, C.C., Diesmann, M., van Albada, S.J.: Multi-scale account of the network structure of macaque visual cortex. Brain Struct. Funct. **223**(3), 1409–1435 (2017). https://doi.org/10.1007/s00429-017-1554-4
15. Potjans, T.C., Diesmann, M.: The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model. Cereb. Cortex **24**(3), 785–806 (2014). https://doi.org/10.1093/cercor/bhs358
16. Hagen, E., Næss, S., Ness, T.V., Einevoll, G.T.: Multimodal modeling of neural network activity: computing LFP, ECoG, EEG, and MEG signals with LFPy 2.0. Front. Neuroinform. **12**, 92 (2018). https://doi.org/10.3389/fninf.2018.00092
17. Rotter, S., Diesmann, M.: Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. Biol. Cybern. **81**(5–6), 381–402 (1999). https://doi.org/10.1007/s004220050570
18. Eppler, J.M., Helias, M., Muller, E., Diesmann, M., Gewaltig, M.O.: PyNEST: a convenient interface to the NEST simulator. Front. Neuroinform. **2**, 12 (2009) https://doi.org/10.3389/neuro.11.012.2008
19. Plotnikov, D., Rumpe, B., Blundell, I., Ippen, T., Eppler, J.M., Morrison, A.: NESTML: a modeling language for spiking neurons. In: Modellierung 2016, Lecture Notes in Informatics (LNI), pp. 93–108 (2016) https://doi.org/10.5281/zenodo.1412345
20. Djurfeldt, M., et al.: Run-time interoperability between neuronal network simulators based on the music framework. Neuroinformatics **8**, 43–60 (2010). https://doi.org/10.1007/s12021-010-9064-z
21. Eppler, J.M., Peyser, A., Schenck, W.: The NESTio project - replacement data recording backend for NEST. In: NEST Conference (2017). juser.fz-juelich.de/record/842049
22. Eppler, J.M., Kupper, R., Plesser, H.E., Diesmann, M.: A testsuite for a neural simulation engine. Front. Neuroinform. Conference Abstract Neuroinformatics (2009). https://doi.org/10.3389/conf.neuro.11.2009.08.042
23. Jordan, J., et al.: Extremely scalable spiking neuronal network simulation code: from laptops to exascale computers. Front. Neuroinform. **12**, 2 (2018). https://doi.org/10.3389/fninf.2018.00002

24. Linssen, C., et al.: NEST 2.16.0 Zenodo (2018). https://doi.org/10.5281/zenodo.1400175
25. Akar, N.A., et al.: Arbor – a morphologically-detailed neural network simulation library for contemporary high-performance computing architectures. In: Proceedings of 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 274–282 (2019). https://doi.org/10.1109/EMPDP.2019.8671560
26. Akar, N.A., et al.: arbor-sim/arbor: Arbor Library v0.2 Zenodo (2019). https://doi.org/10.5281/zenodo.2583709
27. Carnevale, T.N., Hines, M.L.: The Neuron Book. Cambridge University Press (2006). https://doi.org/10.1017/CBO9780511541612
28. Dai, K., et al.: The SONATA data format for efficient description of large-scale network models. bioRxiv 625491 (2019, in preprint). https://doi.org/10.1101/625491
29. Kreuzer, A., et al.: DEEP-EST deliverable D1.5: final report on applications experience (2021). https://www.deep-projects.eu/project/deliverables.html