Performance Comparison for Neuroscience Application Benchmarks

Andreas Herten[1], Thorsten Hater[1], Wouter Klijn[1], and Dirk Pleiter[1]

[1]Forschungszentrum Jülich, JSC, 52425 Jülich, Germany
{a.herten,t.hater,w.klijn,d.pleiter}@fz-juelich.de

Abstract. Researchers within the Human Brain Project and related projects have in the last couple of years expanded their needs for high-performance computing infrastructures. The needs arise from a diverse set of science challenges that range from large-scale simulations of brain models to processing of extreme-scale experimental data sets. The ICEI project, which is in the process of creating a distributed infrastructure optimised for brain research, started to build-up a set of benchmarks that reflect the diversity of applications in this field. In this paper we analyse the performance of some selected benchmarks on an IBM POWER8 and Intel Skylake based systems with and without GPUs.

This article has been published with DOI 10.1007/978-3-030-34356-9_31.

Keywords: OpenPOWER, high-performance computing, data analytics, GPU acceleration, computational neuroscience

1 Introduction

As new computational and data science communities emerge, needs arise for having a benchmark suite reflecting the requirements of the respective communities, also for potential procurement of IT equipment. At the same time, experience needs to be collected regarding performance observations on different types of hardware architectures. In this contribution we address the latter by selecting a recently developed benchmark suite and comparing performance results obtained on servers based on different processor architectures, namely POWER8 and Skylake, with and without GPU acceleration.

The science community, on which we focus here, is the brain research community organised in the Human Brain Project (HBP).[1] HBP is a large-scale flagship project funded by the European Commission working towards the realisation of a cutting-edge research infrastructure that will allow researchers to advance knowledge in the fields of neuroscience, computing, and brain-related medicine. As part of HBP, the ICEI project (Interactive Computing e-infrastructure for the Human Brain Project) was started in early 2018. This project plans to deliver a set of e-infrastructure services that will be federated to form the Fenix Infrastructure.[2] The European ICEI project is funded by the European Commission and is

[1] https://www.humanbrainproject.eu/en/
[2] https://fenix-ri.eu/

formed by the leading European Supercomputing Centres BSC in Spain, CEA in France, CINECA in Italy, CSCS in Switzerland and Jülich Supercomputing Centre (JSC) in Germany. To guide the creation of this infrastructure, the ICEI project started to build-up the "ICEI Application Benchmark Suite", which we use for this contribution.

This paper is organised as follows: We start with giving an overview of the ICEI benchmark suite as well as the systems used for collecting performance results in section 2 and 3, respectively. The obtained results are documented in section 4. We finally provide a summary and conclusions in section 5.

2 ICEI benchmark suite

The components of the "ICEI Application Benchmark Suite" have been chosen such that it represents the breadth of research within HBP. The subset of benchmarks, which we consider in this paper, is directly based on real-life applications. *NEST* [8] is one of several simulators that became part of the benchmark suite. It is a simulator for spiking neural network models that focuses on the dynamics, size, and structure of neural systems rather than on the exact morphology of individual neurons. Recently, a significantly improved uptake of this simulators in different areas of brain research has been observed. NEST is a community code with an active user base. A key design goal is extreme (weak) scalability, which could be demonstrated different supercomputers (see, e.g., [5]). The program is written in C++ and Python, and uses MPI and OpenMP for parallelisation.

Unlike NEST, *Arbor* [2] is a simulation library for networks of morphologically detailed neurons. Simulations progress by taking half time steps for updating the states of the cells. This allows overlapping the exchange of the spikes generated by the cells. During the communication of spikes with other cells, similar operations need to be performed as in case of NEST. The performance in this step will mainly depend on memory and network performance. The step of updating the cells is, however, more compute intensive and can potentially benefit from compute acceleration through SIMD pipelines or GPUs. Cells are represented as trees of line segments, on which partial differential equations for potentials are solved using the finite-volume method. For complex cell models the second step will dominate application performance. Arbor is mainly written in C++ and employs MPI and OpenMP as well as CUDA for parallelisation.

The Virtual Brain (TVB) [3,4,11] is an application that aims at full brain network simulation. It uses mesoscopic models of neural dynamics, which model whole brain regions. For the interconnection of the different regions structural connectivity data sets are used. The application can generate outputs on different experimental modalities (for instance EEG or fMRI) and thus allows to compare simulated and experimental data. To enable exploitation of supercomputers, a new version of the application is being implemented, which is called *TVB-HPC*. TVB-HPC is written in Python and aims to automatically produce code for different targets, including processor architectures with SIMD pipelines or GPU

accelerators. One of the targets is Numba [7], which is a tool that translates Python functions to optimized machine code. TVB-HPC is using Numba for the benchmark at hand to just-in-time-compile Python code to CPU assembly. MPI is used to distribute tasks.

While the previous three applications enable different kind of brain simulations, the remaining applications, which have been used for the "ICEI Application Benchmark Suite" and are considered here, address data analysis tasks.

This includes *ASSET* [13], which is part of the Elephant (Electrophysiology Analysis Toolkit). Elephant is a library comprising a set of tools for analysing spike train data and other time series recordings obtained from experiments or simulations. Elephant is written in Python and relies on NumPy and SciPy for numerical tasks and MPI/mpi4py parallelisation. The tool ASSET (Analysis of Sequences of Synchronous EvenTs) was developed to automatise processing of spike data for sequences of synchronous spike events. In the ASSET benchmark at hand, one of the main compute kernels is compiled with Cython.

Another type of data processing challenge occurs in the context of analysis of high-resolution images of histological brain sections. To automatise the analysis of such images, applications based on deep learning techniques have been developed [12]. The *Neuroimaging Deep Learning* benchmark is derived from one such application. It is based on TensorFlow in combination with Horovod for parallelisation, using TensorFlow's GPU backend in the benchmark presented here.

3 Test systems

The "ICEI Application Benchmark Suite" has been executed on a variety of systems to improve portability and collect performance results for different architectures. Here we focus on results obtained on two systems installed at Jülich Supercomputing Centre:

- JURON is a pilot system dedicated to users from HBP, which was delivered by IBM and NVIDIA in the context of a pre-commercial procurement that was executed during an the initial phase of the HBP.
- JUWELS is a flagship cluster system at JSC, which is one of the PRACE Tier-0 systems that are accessible for European researchers at large.

The 18 compute nodes of JURON are IBM S822LC servers (also known under the codename *Minsky*). Each node comprises two IBM POWER8 processors and four NVIDIA P100 GPUs. Each group of one processor and two GPUs is interconnected via NVLink links. The compute nodes are connected via Mellanox ConnectX-4 Infiniband EDR network adapters to a single switch. In the following we use the term "CPU-only nodes" when referring to JURON nodes where the GPUs are not used.

The JUWELS cluster comprises 2511 CPU-only and 48 GPU-accelerated compute nodes. Each comprises two Intel processors of the Skylake generation. The GPU-accelerated nodes are additionally equipped with four NVIDIA V100

GPUs. While the four GPUs are interconnected via NVLink in an all-to-all topology, each GPU is only connected via one PCIe Gen3 link to one of the CPUs. The compute nodes furthermore comprise a single Mellanox ConnectX-5 Infiniband EDR network adapter through which they are interconnected using a fat-tree topology.

A more detailed comparison of the hardware capabilities of the nodes used for either system are collected in Table 1. As the benchmarks considered here are compute-only (any time spent in I/O is not considered), we do not report on I/O capabilities of both systems.

Table 1. Comparison of node-level aggregated hardware parameters.

	JURON	JUWELS
Type of CPU	POWER8	Intel Xeon Platinum 8168 / Intel Xeon Gold 6148 (GPU-acc.)
Number of CPUs	2	2
Number of cores	20	48 / 40
Number of hardware threads	160	96 / 80
SIMD width / bit	128	512
Throughput / Flop/cycle	160	1536 / 1280
Memory capacity / GiB	256	$\geq$96
Memory bandwidth / GB/s	230	255
LLC capacity / MiB	160	66 / 27.5
Number of GPUs	4	– / 4
Type of GPU	P100 SXM2	V100 SXM2
Throughput / Flop/cycle	14 336	20 480
Memory capacity / GiB	64	64
Memory bandwidth / GB/s	2880	3600

4 Results

In this section we document selected results for the benchmark derived from the applications introduced in section 2, which have been obtained on the systems introduced in section 3.

4.1 NEST

The benchmark is based on Version 2.14 of NEST [10].[3] Simulations are performed using a randomly connected network of 112 500 neurons with each neuron being

[3] https://github.com/nest/nest-simulator.git

connected to about 10 % of the other neurons. While the problem size is kept fixed, the number of MPI tasks and OpenMP threads can be varied. Internally, NEST defines virtual processes (VP) and assigns one VP to each thread. The application first builds a network, i.e. creates all neurons and connects them. In a second step simulations are performed. Here, 1000 ms biological time are simulated. For this paper the GCC C++ compiler version 5.4.0 and 5.5.0 have been used on JURON and JUWELS, respectively. Since NEST does not support GPU acceleration, we use the CPU-only nodes on JUWELS.

In Fig. 1 and 2 we show how simulation time scales on a single node as a function of the number of VPs. The number of VPs is equal to the number of threads, which is the product of the number of nodes, tasks per node, and threads per task. The results for multi-node scaling are shown in Table 2.

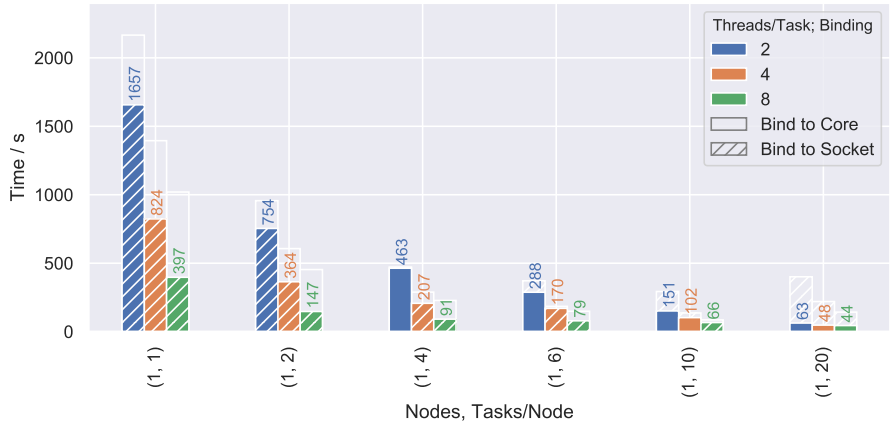

Fig. 1. NEST benchmark – JURON: Simulation time on a single JURON node for different numbers of virtual processes (VP, the product of the number of nodes, tasks per node, and threads per task). Different strategies of mapping and binding MPI tasks to the system have been tested and the best performing configuration selected; a shaded bar denotes *binding to socket*, a solid bar denotes *binding to core*.

NEST can efficiently exploit node- and thread-level parallelism and therefore exhibits a good scaling behaviour on both architectures. On POWER8 processors the application to some extent benefits from using up to 8 hardware threads per core. This trend can not compensate for the larger number of cores available on the Skylake processor. Since NEST is not capable of exploiting SIMD parallelism, the wider SIMD units of the Xeon processors do not add benefits for this application.

For NEST it is of special importance how the VPs are distributed to the available hardware resources. NEST uses OpenMP for shared-memory parallelism (*Threads/Task*) and MPI for task-based parallelism (*Tasks/Node*), potentially

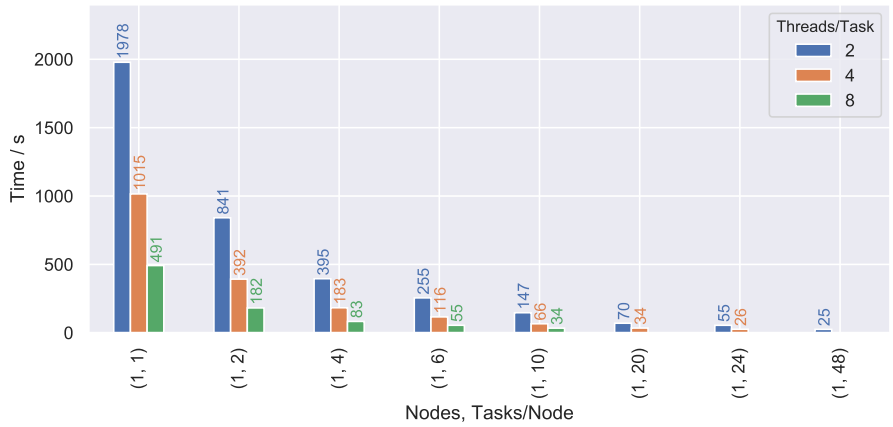

Fig. 2. NEST benchmark – JUWELS: Same as in Fig. 1 but for JUWELS.

Table 2. NEST benchmark: Build and simulation time for 1, 2 and 4 nodes and optimal number of tasks and threads per node.

	JURON		JUWELS	
Nodes	Build / s	Simulation / s	Build / s	Simulation / s
1	2.58	44.50	1.25	25.15
2	2.34	32.39	0.81	15.15
4	1.39	18.80	0.63	5.88

across node borders. The employed strategy to distribute OpenMP threads uses the `OMP_PLACES` environment variable to distribute along physical cores (`cores`); this variable might possibly interact with MPI binding options. To fix each OpenMP thread to a certain place, the affinity variable `OMP_PROC_BIND` is set to `TRUE`. Of more importance for the two-socket system JURON is the employed distribution of MPI tasks. In Fig. 1, two binding schemes of Open-MPI were used. The hatched bars (usually for lower number of VPs) bind tasks to cores (`--bind-to core`) whereas the solid bars bind tasks to sockets (`--bind-to socket`); in both cases, a mapping along sockets is chosen (`--map-by socket`). Setting the *wrong* binding can entail serious performance penalties (see also the white-outlined bars in the background of Fig. 1). Fig. 3 compares the two binding strategies for four selected distributions of 20 VPs along tasks and threads. It can clearly be seen that for few tasks and many threads in the left of the figure, `--bind-to socket` is the more beneficial binding option. For many tasks with each few threads (right of the figure), `--bind-to core` is the more sensible choice. While these results might be expected, they might

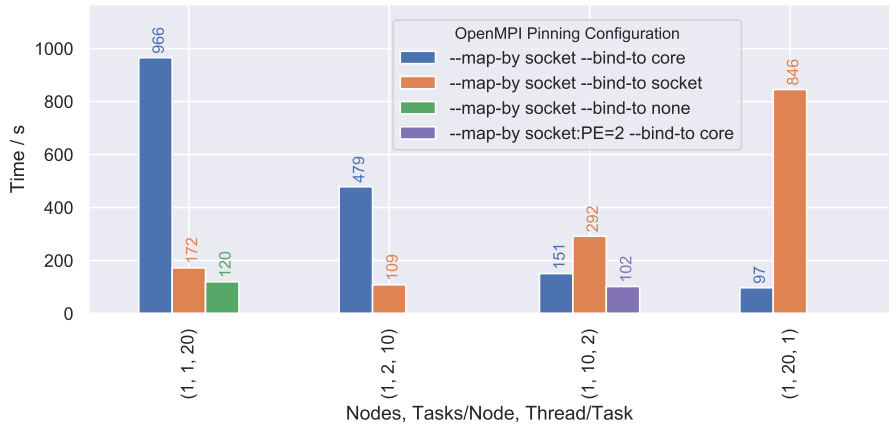

Fig. 3. NEST benchmark – JURON: Different binding strategies for different distributions of VPs.

not be the choice of distribution for the employed MPI. Fig. 3 also shows two further optimized binding configurations: For the configuration of 1 Task and 20 Threads (the very left), disabling binding can improve performance further (`--bind-to none`) as now both sockets can be used by the 20 threads. Another way to bind can be seen for the case of 10 Tasks and 2 Threads per Task; by binding to cores but also mapping to the sockets such that exactly 2 cores (`PE=2`) are bound to each task, the MPI runtime has all information about tasks and threads to create a performance-beneficial task distribution (each task has two associated OpenMP places).
The mapping and binding options for JUWELS are more limited as the combination of job scheduler and MPI runtime currently don't offer similar high-level functionality as in the case for JURON. For JUWELS, the shown values are measured by using a pinning mask manually created by the tool `hwloc`, distributing tasks across the node architecture. In general, the default pinning of JUWELS is much better than the default pinning of JURON.

One of the main performance limiters for NEST on JURON are stalled processor cycles, as shown in Fig. 4. The figure shows measurements of a selection of hardware performance counters using the `perf` utility. Most of the stalls can be attributed to misses of the data cache, more specifically to cases were data was not in the L3 level cache.

4.2 Arbor

Arbor is a rather new simulator, which has been designed from scratch with the goal of supporting different HPC architectures in a performance portable manner.

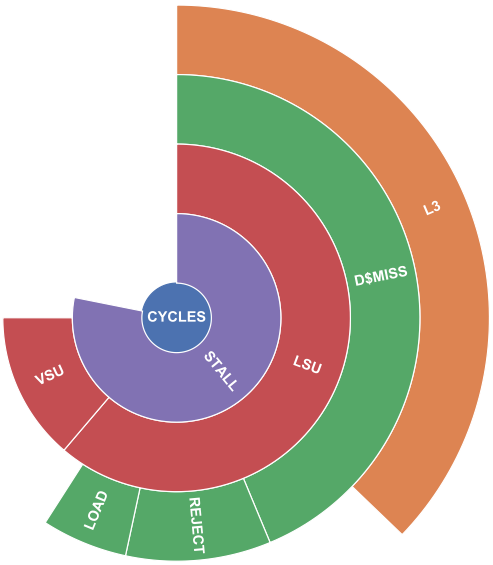

Fig. 4. NEST benchmark – JURON: Performance counters.

We use version 0.1 of Arbor[1].[4] The different simulation phases are similar as for NEST, but given that simulation of multi-compartment models of neurons are much more expensive, the benchmark focuses exclusively on the simulation phase. Simulation proceeds in a lock-step manner by first updating all cells half a time step and then overlapping the exchange of spikes with the further half time steps. Arbor allows to group cells in a flexible manner depending on the target architecture. For instance, large cell groups are used for GPUs that require a very high level of parallelism. For this paper the GNU C++ compiler version 6.3.0 and 8.2.0 have been used on JURON and JUWELS, respectively, when running the CPU version of the benchmark, and 6.3.0 and 7.3.0, respectively, when running the GPU version of the benchmark. For the GPU benchmark CUDA 9.2.148 was used on JURON and CUDA 9.2.88 was used on JUWELS. We perform simulations involving 1000 cells covering a biological time of 1 s (GPU version) or 0.1 s (CPU version).

In Fig. 5 we show how performance scales on a single CPU-only node using 2 MPI tasks and a variable number of threads per MPI task. On POWER8 little benefit is observed from using multiple hardware threads per physical CPU core.

[4] https://github.com/arbor-sim/arbor.git

The application is significantly faster on the Xeon nodes as it can efficiently exploit the wide SIMD units. The observed performance ratio roughly matches the ratio of throughput in floating-point operations (see Table 1). The higher clock frequency of the POWER8 processor does not seem to help improving performance significantly.

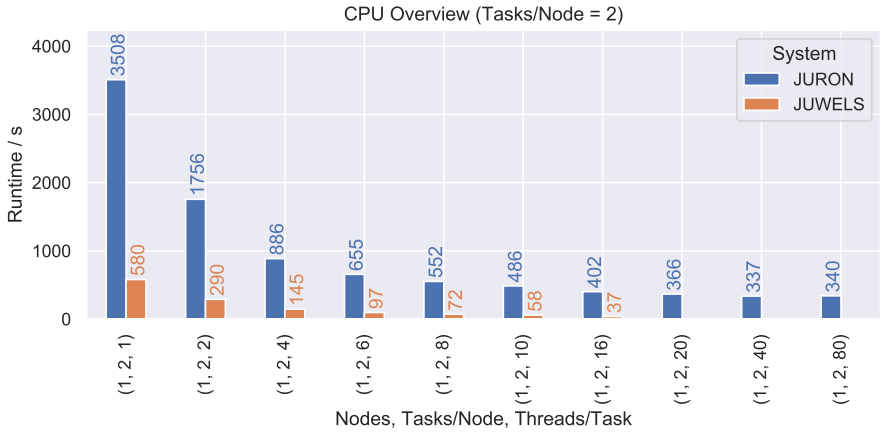

Fig. 5. Arbor benchmark: Time needed to simulate 100 ms biological time on a single CPU-only node.

Next we compare performance using Arbor on GPU-accelerated nodes. The results are shown in Fig. 6. To highlight differences in scaling among different numbers of GPUs, the simulation time has been increased ten times compared to the CPU-only benchmark. Performance on JUWELS and JURON now is similar when taking into account that the V100 GPUs used in JUWELS have an about 40 % higher throughput of double-precision floating-point operations compared to the P100 GPUs used in JURON.

4.3 TVB-HPC

The HPC version of TVB, which uses Numba for code generation, is still in development and we therefore use a pre-release version. The benchmark uses a simple mesoscopic model based on the Kuramoto model [6], which is used for the study of neuronal oscillations and synchronization. 1600 time steps are simulated. We use Python 3.6.1 (3.6.6) as well as version 0.39.0 (0.40.1) of Numba and version 1.14.2 (1.15.2) of Numpy on JURON (JUWELS).

In Fig. 7 we show the scaling of the benchmark on up to two nodes. The observed scaling behaviour is similar on both architectures with a parallel efficiency of about 80 % when using 16 MPI tasks on a single node. On a single node the benchmark execution time on JURON is consistently about 35 % slower

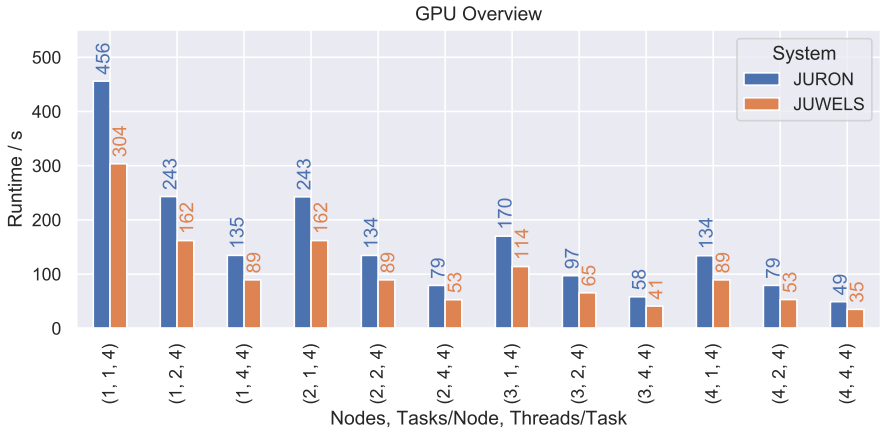

Fig. 6. Arbor benchmark: Time needed to simulate 1000 ms biological time on one or more GPU-accelerated nodes.

compared to JUWELS when using the same number of MPI tasks. The difference drops to about 13 % when using two nodes. This version of TVB-HPC is not able to exploit SIMD parallelism. This observation plus the limited scalability results in performance differences between JURON and JUWELS that are relatively small.

4.4 Elephant ASSET

The benchmark is based on version 1.0 of ASSET[5] as well as Python libraries neo (version 0.6.1), sklearn (version 0.19.1), and elephant (version 0.5.0), respectively. Most of the time is spent computing a set of survival functions, which requires computing of statistical distributions based on the input data. This makes speed of memory access in general a performance limiting factor, with efficiency of handling of Python arrays being an implementation specific aspect. Parallelisation is realised by distributing the input data in an approximately fair manner to all available MPI tasks. While MPI parallelisation is done explicitly within the application, exploitation of any additional thread-level parallelism is left to Python.

In Figure 8 we show benchmark run-time as well as the execution time of the main kernel, which computes the joint survival function, as a function of the number of tasks. While execution time on JUWELS is first lower than on JURON, the scaling behaviour on JUWELS is slightly worse, resulting in JURON having the lowest achieved benchmark time with a sufficient number of tasks (30).

[5] https://github.com/INM-6/pcp_use_cases.git

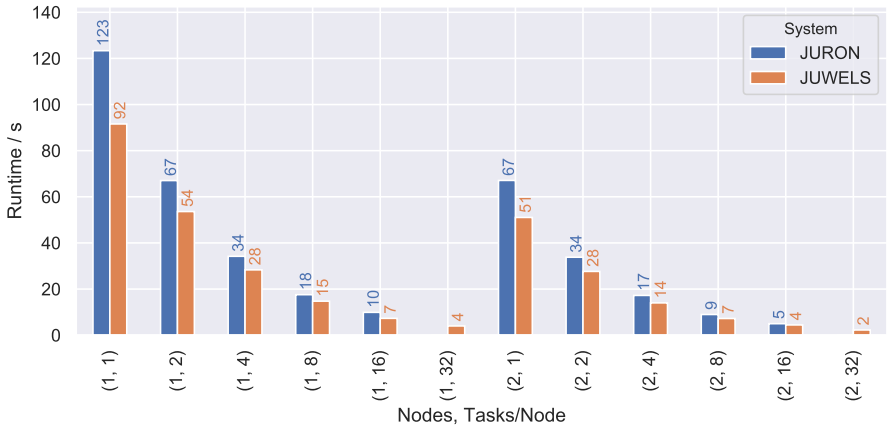

Fig. 7. TVB-HPC benchmark: Scaling of the benchmark on up to 2 CPU-only nodes as a function of the number of MPI tasks.

4.5 Neuroimaging Deep Learning

The benchmark is a mini-application version of the real code. It is extracted such that input data is loaded first to main memory to separate performance impacts related to the capabilities of the storage system, in which the input data resides.[6] For the results shown below TensorFlow version 1.4.1 (1.8.0) and Horovod version 0.14.1 (both) was used on JURON (JUWELS).

A selection of benchmark results are listed in Figure 9. The performance is measured in terms of time needed to process a single image. On both of considered architectures the intra-node scaling as well as the inter-node scaling behaviour is fair. The performance on JUWELS suffers efficiency losses of (max.) 30 % and 23 % for intra- and inter-node scaling, respectively; for JURON it is 10 % and 6 %. Absolute performance and scaling performance is significantly better on JURON despite an older generation of GPUs being used. This may be an indication of better data transport capabilities having an important performance impact in this application.

5 Summary and Conclusions

In this paper we presented selected performance results obtained for the "ICEI Application Benchmark Suite" using a slightly older system based on IBM POWER8 processors plus NVIDIA P100 GPUs (*JURON*) as well as a more recent system based on Intel Skylake processors plus NVIDIA V100 GPUs (*JUWELS*).

[6] I/O performance for this application is crucial and has been analysed in [9].

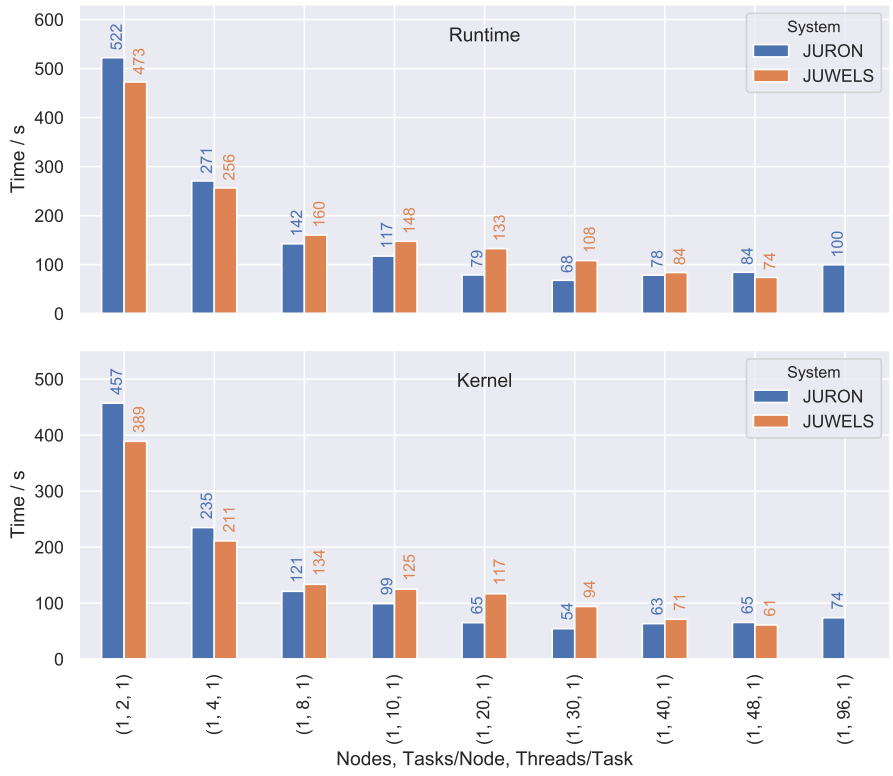

Fig. 8. Elephant-ASSET benchmark: Benchmark (top) and kernel (bottom) execution times as a function of tasks on one node for JURON and JUWELS.

The example of Arbor indicates that for compute-limited applications, which can exploit wide SIMD pipelines without using GPUs as compute accelerators, the performance of CPU-only JUWELS nodes exceed those of JURON significantly. For processors based on the POWER architecture being competitive, higher throughput of floating-point operations would be desirable.

For applications, which cannot exploit SIMD parallelism, performance on JURON and JUWELS was found to be similar in case of TVB-HPC and ASSET. In case of NEST, which compared to the other applications is able to scale slightly better by making use of the larger number of available cores, JUWELS was found to perform better.

Finally, for the machine learning application the POWER-based system was found to be better.

The results provided in this paper give an overview over the performance trend for different applications from a benchmark suite that aims for reflecting the needs of the relatively diverse community of brain research. We plan for further efforts to analyse the causes for the different performance behaviour. This

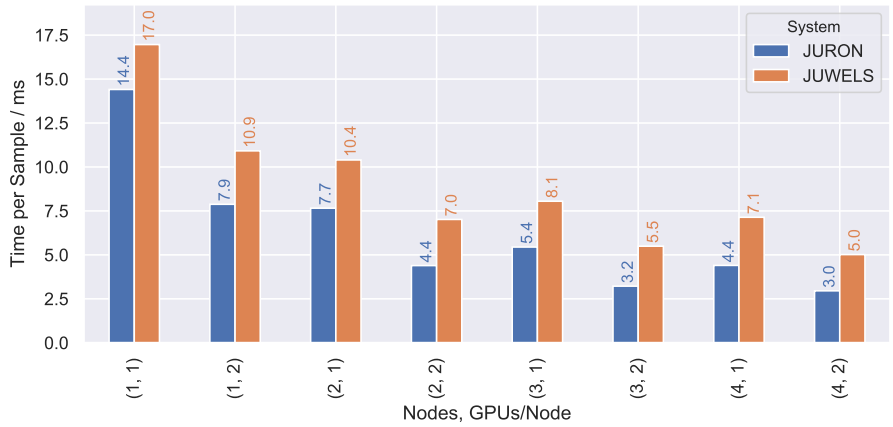

Fig. 9. Neuroimaging Deep Learning benchmark: Time per sample as a function of GPUs and tasks for JURON and JUWELS.

will help to guide designing future e-infrastructures optimised for this community. We believe that the increased choice of architectures and technologies, which can be used for this purpose, is helpful.

Acknowledgements

We would like to thank the many people that have contributed to the creation of the ICEI Benchmark Suite, which was used for this paper. This includes in particular the following persons: Ben Cumming (CSCS, Switzerland), Sandra Diaz (JSC, Germany), Pramod Kumbhar (EPFL, Switzerland), Lena Oden (JSC and FU Hagen, Germany), Alexander Peyser (JSC, Germany), Hans Ekkehard Plesser (NMBU, Norway), Alper Yegenoglu (FZJ, Germany), and all the collaborators of the respective community codes used. Funding for the work is received from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 785907 (HBP SGA2) and No. 800858 (ICEI).

References

1. Akar, N.A., Biddiscombe, J., Cumming, B., Kabic, M., Karakasis, V., Klijn, W., Küsters, A., Martinez, I., Peyser, A., Yates, S.: arbor-sim/arbor: Version 0.1: First release (Oct 2018), https://doi.org/10.5281/zenodo.1459679
2. Akar, N.A., Cumming, B., Karakasis, V., Küsters, A., Klijn, W., Peyser, A., Yates, S.: Arbor - A morphologically-detailed neural network simulation library for contemporary high-performance computing architectures. In: 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2019, Pavia, Italy, February 13-15, 2019. pp. 274–282. IEEE (2019), https://doi.org/10.1109/EMPDP.2019.8671560

3. Jirsa, V., McIntosh, R., Ritter, P., Mersmann, J., et al.: http://www.thevirtualbrain.org/
4. Jirsa, V.K., Sporns, O., Breakspear, M., Deco, G., McIntosh, A.R.: Towards The Virtual Brain: network modeling of the intact and the damaged brain. Archives italiennes de biologie 148(3), 189–205 (2010)
5. Jordan, J., Ippen, T., Helias, M., Kitayama, I., Sato, M., Igarashi, J., Diesmann, M., Kunkel, S.: Extremely scalable spiking neuronal network simulation code: From laptops to exascale computers. Frontiers in Neuroinformatics 12, 2 (2018)
6. Kuramoto, Y.: Self-entrainment of a population of coupled non-linear oscillators. In: Araki, H. (ed.) International Symposium on Mathematical Problems in Theoretical Physics. pp. 420–422. Springer Berlin Heidelberg, Berlin, Heidelberg (1975)
7. Lam, S.K., Pitrou, A., Seibert, S.: Numba: A llvm-based python jit compiler. In: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC. pp. 7:1–7:6. LLVM '15, ACM, New York, NY, USA (2015), http://doi.acm.org/10.1145/2833157.2833162
8. Linssen, C., Lepperød, M.E., Mitchell, J., Pronold, J., Eppler, J.M., Keup, C., Peyser, A., Kunkel, S., Weidel, P., Nodem, Y., Terhorst, D., Deepu, R., Deger, M., Hahne, J., Sinha, A., Antonietti, A., Schmidt, M., Paz, L., Garrido, J., Ippen, T., Riquelme, L., Serenko, A., Kühn, T., Kitayama, I., Mørk, H., Spreizer, S., Jordan, J., Krishnan, J., Senden, M., Hagen, E., Shusharin, A., Vennemo, S.B., Rodarie, D., Morrison, A., Graber, S., Schuecker, J., Diaz, S., Zajzon, B., Plesser, H.E.: NEST 2.16.0 (8 2018), https://zenodo.org/record/1400175
9. Oden, L., Schiffer, C., Spitzer, H., Dickscheid, T., Pleiter, D.: IO challenges for human brain atlasing using deep learning methods - an in-depth analysis. In: 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2019, Pavia, Italy, February 13-15, 2019. pp. 291–298. IEEE (2019), https://doi.org/10.1109/EMPDP.2019.8671630
10. Peyser, A., Sinha, A., Vennemo, S.B., Ippen, T., Jordan, J., Graber, S., Morrison, A., Trensch, G., Fardet, T., Mørk, H., Hahne, J., Schuecker, J., Schmidt, M., Kunkel, S., Dahmen, D., Eppler, J.M., Diaz, S., Terhorst, D., Deepu, R., Weidel, P., Kitayama, I., Mahmoudian, S., Kappel, D., Schulze, M., Appukuttan, S., Schumann, T., Tunç, H.C., Mitchell, J., Hoff, M., Müller, E., Carvalho, M.M., Zajzon, B., Plesser, H.E.: Nest 2.14.0 (Oct 2017), https://doi.org/10.5281/zenodo.882971
11. Ritter, P., Schirner, M., McIntosh, A.R., Jirsa, V.K.: The Virtual Brain integrates computational modeling and multimodal neuroimaging. Brain Connectivity 3(2), 121–145 (2013)
12. Spitzer, H., Amunts, K., Harmeling, S., Dickscheid, T.: Parcellation of visual cortex on high-resolution histological brain sections using convolutional neural networks. In: 2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017). pp. 920–923 (April 2017)
13. Torre, E., Canova, C., Denker, M., Gerstein, G., Helias, M., Grün, S.: ASSET: Analysis of sequences of synchronous events in massively parallel spike trains. PLOS Computational Biology 12(7), 1–34 (07 2016)