D6.4

Report on the general challenges still to overcome to bridge the gap between pre-exascale and exascale simulations

Fabio Affinito, Riccardo Bertossa, Pietro Delugas, Alberto García, Anton Kozhevnikov, Luigi Genovese, Pablo Ordejón, Nicola Spallanzani, Daniel Wortmann

Due date of deliverable:	30/09/2022
Actual submission date:	18/10/2022
Final version:	07/10/2022

Lead beneficiary:	ICN2 (participant number 3)
Dissemination level:	PU - Public

Document information

Project acronym:	MaX
Project full title:	Materials Design at the Exascale
Research Action Project type:	European Centre of Excellence in materials modelling, simulations and design
EC Grant agreement no.:	824143
Project starting / end date:	01/12/2018 (month 1) /30/09/2022 (month 46)
Website:	www.max-centre.eu
Deliverable No.:	D6.4

Authors: Fabio Affinito, Riccardo Bertossa,Pietro Delugas, Andrea Ferretti, Alberto García, Luigi Genovese, Anton Kozhevnikov, Savio Laricchia, Pablo Ordejón, Nicola Spallanzani, Daniel Wortmann

To be cited as: Fabio Affinito, Riccardo Bertossa, Pietro Delugas, Andrea Ferretti, Alberto García, Luigi Genovese, Anton Kozhevnikov, Savio Laricchia, Pablo Ordejón, Nicola Spallanzani, Daniel Wortmann (2022): Report on the general challenges still to overcome to bridge the gap between pre-exascale and exascale simulations. Deliverable D6.4 of the H2020 project MaX (final version as of 07/10/2022). EC grant agreement no: 824143, ICN2, Barcelona, Spain.

D6.4 Report on the general challenges still to overcome to bridge the gap between pre-exascale and exascale simulations

Content

1. Executive Summary

In this deliverable we collect and discuss the difficulties encountered when running the demonstrator use cases reported in D6.3, as well as the challenges foreseen to be faced when taking and or scaling them on future pre- and exascale machines. In this perspective, this deliverable represents a collection of lessons learned that are rationalised and discussed in terms of expected hardware deployment and evolution.

For each MaX flagship code, different classes of challenges are addressed, including difficulties (i) at the application level (porting problems, parallelization problems, reliability issues, robustness); (ii) at the software level (related to the environment on the HPC systems, i.e. libraries, schedulers, compilers); (iii) at the system level (not enough memory on the node, poor bandwidth, internode latency, too much/too few cores/gpus, system reliability such as crashes of the nodes, etc); (iv) related to other aspects of the interaction with workflow management systems like AiiDA (problems with very large systems - datasets - numbers of jobs in the workflows, etc).

Overall, diverse difficulties and challenges have been identified and reported by different codes, in view of both their specific theoretical and technical features, as well as due to the different Demonstrators deployed and studied (while FLEUR studied large magnetic motives in chiral magnets, BigDFT addressed biomolecules, and Yambo tackled on excitations in photocatalytic systems). Nevertheless, some common difficulties can be envisaged, including: (1) the fast evolving software stack and programming models to best exploit GPUs (especially for Fortran community codes). In particular, a different level of maturity has been found for different software stacks. (2) The performance of the codes is very dependent on vendor-specific optimised libraries and tools, underlying a strong need for some missing or still evolving components, such as, e.g., distributed linear algebra on GPUs. (3) The combination of GPU support with internode-node MPI communication may be critical, and is foreseen to become even more so on exascale machines, representing an open challenge for the HPC community at large. Last, (4) while we have not encountered sensible difficulties in deploying the AiiDA workflow manager on existing HPC machines, looking at exascale-class machines we foresee the need to include an extra software layer (meta-scheduler) in between AiiDA and the actual machine scheduler (e.g. SLURM) in order to increase the flexibility of job and workflow scheduling.

2. QUANTUM ESPRESSO

In the following we report about the challenges met and solved during the deployment of the demonstrator "On-the-fly computation of heat currents with cp.x and QEHeat applications".

2.1 Difficulties with current applications and software

The calculations of this demonstrator aimed at building a time-series of the heat-flux in a supercell of 125 water molecules, using the applications cp.x and QEHeat from the Quantum ESPRESSO suite. Such time-series were computed on-the-fly. While cp.x computed the Ab-initio Molecular Dynamics (AIMD) trajectory, saving data on disc, the QEHeat application was run concurrently and extracted a snapshot of the dynamics each 3 fs and computed the heat-flux. AIMD was executed using 4 nodes of the A3 partition of Marconi@CINECA cluster. In order to demonstrate the on-the-fly concept we tried to set up the calculation so that the current time-series evolves at the same rate as the dynamics without accumulating an excessive lag.

The first main difficulty is at the application level, because we had to use separate applications for performing the AIMD and the current calculations at each snapshot. We had to synchronise these applications at the level of bash script. Since the first calculations, the situation has slightly improved on this aspect because the later versions of QEHeat were refactored in order to perform directly the preliminary ground-state calculation, reducing the synchronisation issues and the use of disk I/O.

One other issue is the different scalability of the two tasks, with a much larger speedup of cp.x with respect to the flux calculator. This is in part due to the fact the QEHeat workflow includes two self-consistency loops, a preliminary one for computing the charge density and the wave-functions plus a linear response one to finally compute the heat current. This bottleneck can be straightforwardly overcome by increasing the number of QEHeat workers that were executed concurrently with the AIMD. After different attempts we verified that from 6 to 8 different QEHeat instances running each one on 1 node were needed to keep the rate of the AIMD evolutions.

While this solution worked for our demonstrator, it presents a few drawbacks that make it infeasible for the ordinary usage:

- the resource usage is inefficient because it is necessary to allocate the nodes for the QEHeat instances at the same time that the AIMD starts but the calculations can start only after that a sufficient number of snapshots has been generated;

- it is necessary to require a large number of nodes (10 - 12) for the maximum time allowed (24 hrs), and this tends to lower the priority of the submitted jobs;
- performing the synchronisation via bash script has turned out to be complicated and error prone, even considering that the simulation needs to be restarted many times and that the management of the recovery phase was particularly cumbersome.

On the other hand, we did not find any particular hurdle at the system level.

2.2 Perspectives and proposed features

These difficulties, in part also reported in MaX deliverables D6.1 and D6.3, clearly showed that the execution of on-the-fly calculations of the heat-flux along an AIMD trajectory would become much simpler and possibly more efficient if the synchronisation was performed at the application level.

We have been working on a proof of concept implementation of such synchronisation using the TCP/IP stack. This proof of concept leverages the ZeroMQ framework (http://www.zeromq.org) to synchronise and exchange data between the different tasks of the workflow. These are organised in MPI groups and are run as a single application. For doing this we have first written an interface module for accessing ZeroMQ functionalities (https://github.com/rikigigi/fortran_cloud), and used this in our proof-of-concept application: https://gitlab.com/rikigigi/q-e/-/tree/on_the_fly .

This has also allowed for the implementation of the binary data-exchange between the AIMD quantum-engine and the property calculator. The reuse of the binary data will in part reduce the scalability imbalance between the two parts of the workflow, favouring a more uniform work distribution. The first results and plans for extending this proof-of-concept can be found in the MaX report D6.3.

Among the requirements for future software and systems, we thus would like to point out the availability of some of the features of the ZeroMQ paradigm:

- an API that allows for the introduction of a transparent and efficient data-transfer and synchronisation in already existing interfaces;
- the possibility of dynamically reassigning the MPI ranks to different applications;
- the possibility to run on different partitions of an HPC cluster.

These features should be helpful for implementing any on-the-fly generation of complex time-series based on AIMD and, in general, complex HPC workflows where the synchronisation and data-exchange among different concurrent applications is needed.

3. Yambo

In the following we discuss the challenges related to the deployment of the demonstrator "Defected Titanium-dioxide slabs for photocatalytic applications".

As outlined in the deliverable D6.3, the combined GW/BSE scheme provides a powerful but computationally demanding tool for describing from first principles optical excitations in titanium-dioxide slabs relevant for photocatalysis. The application of such a scheme to a large hydroxylated 4x2 supercell TiO_2 (110) slab with four trilayers made up of 209 atoms represents a challenge for our simulation codes on the current HPC architecture, as well as for perspective use on pre-exascale and future exascale HPC machines. The achievements reported in D6.3 have been possible thanks to the interplay between methods development and increasing computational power and memory of the GPU cards, along with the GPU porting of the Yambo code. Nevertheless, this study allowed us to highlight the main bottlenecks in the MBPT calculations of large slabs and the plans for the next implementation actions. A detailed description is presented below.

3.1 Necessity of a GPU-aware distributed linear algebra library

The system analysed in D6.3 shows strongly bound excitons in the interval of energies 2-4 eV, which can be explained in terms of a sensible localization of the excitonic wave functions. This allows us to quickly achieve the convergence of the optical spectra with respect to the number of trilayers composing the slab, so that BSE calculations performed for a four trilayer slab made up of 209 atoms will be accurate enough to be comparable with experimental results. However, in case of defective materials with more delocalised exciton wave-functions one should further increase the thickness of the slab to get more reliable BSE optical spectra, resulting thus in an even larger number of reciprocal lattice vectors and bands required for the computation of the irreducible polarizability X_0.

Given the very effective scalability of Yambo, able to reach in the order of thousands MPI tasks also in the presence of accelerators, the main issue hindering an efficient use of Yambo on GPU-accelerated machines is given by the memory footprint and the load balancing. In fact, GPUs typically available during the reporting period (NVIDIA V100 and A100) have 16 to 40 Gb RAM per card, but the numerical requirements of Yambo can easily overwhelm such memory when dealing with very large systems, especially if the bands are not distributed over a big enough number of MPI tasks. In this respect, the inversion of the Dyson equation for the response function could be problematic, easily giving out-of-memory errors, due to the lack of parallel distributed linear algebra support on GPUs.

For example, a spin polarised G_0W_0 calculation for the above case study making use of the plasmon pole approximation (PPA), 4000 bands (including 777 valence states), a X_0 plane wave cutoff of 8 Ry, a 15 Å thick vacuum layer, and a 2x2x1 sampling of the BZ requires at least 12.53 GB per GPU (including 9.03 Gb to allocate the wave functions and 1.08 Gb for the allocation of each X_0 matrix with size n=11809) just before the inversion of the dielectric matrix when distributing the states over 100 nodes on the system Marconi100, i.e. over 400 V100 GPU cards with 16 GB RAM each. It is clear that an increase in the plane waves cut off or in the thickness of the vacuum layer will require the distribution of linear algebra kernels in the G_0W_0 calculation. It is important to note that this is a general issue for most of the MaX codes (the bottleneck being the solution of a dense linear system or a dense eigenproblem, depending on the cases), and for the electronic structure community at large.

A similar situation was also found when diagonalising the two-particle Hamiltonian within the Bethe-Salpeter framework. Therefore, the adoption of a parallel distributed linear algebra library for dense matrices on GPUs (pretty much what Scalapack is for CPUs) is one of the crucial priorities for our future work. Suitable candidates under scrutiny are: (i) the Software for Linear Algebra Targeting Exascale (SLATE) libraries which provides, e.g., basic dense matrix operations, linear system and eigenvalue solvers, and (ii) the NVIDIA maths libraries (cuSOLVER in particular), available as part of the NVIDIA HPC software stack.

3.2 Programming models and software stack

Generally speaking, in order to deploy large scale Yambo calculations on modern, GPU-based HPC architecture, we had to invest significant work in defining the optical porting strategy for the code, while targeting specific HPC machines. Once decisions were made, the current performance-portability implementation in Yambo makes use of CUDA-Fortran for NVIDIA GPUs, ideally paralleled with OpenACC and OpenMP as alternative backends (particularly useful for AMD and INTEL GPUs). In doing this, these languages / programming models were far from robustly supported by even vendor specific compilers. In general, it was common experience (not only of the Yambo developing team) to find and file a number of bugs in both compilers as well as performance libraries. While the interaction with the vendors has been extremely useful and fruitful to promptly fix or workaround the problems, working with such a software environment turned out to be sub-optimal from the point of view of scientific software developers.

This also matched with the need to support and link multiple external performance libraries, which had to be often built and linked on multiple HPC platforms. This built into a more and more complex task and, in order to sort out the issue, we resorted to good practice approaches in HPC such as the use of Spack. Spack is a package manager for supercomputers, Linux, and macOS that simplifies the installation of scientific software. In order to make Yambo installable with spack it was necessary to write a “package recipe”, in the form of a python script, that collects all the installation instructions, and sometimes tricks. It is a very useful tool, for a developer, because depending on the machine architecture, both for host and accelerating device, it is possible to set different software stacks in terms of all the possible combinations of compilers and libraries. This solution provides a good tool also to the user because it hides all the difficulties of the installation process. In addition, as a beneficial side effect, the development of this recipe led to the discovery of bugs related to the autotool procedure of the Yambo code.

4. SIESTA

From the work done in the technical work packages and the demonstrators described in deliverables D6.1, D6.2, and D6.3, we have identified the following bottlenecks of the SIESTA code towards its full deployment on pre- and exascale infrastructures.

4.1 Challenges at the application level

- The advent of new architectures with GPUs has shown the need to re-examine the strategy used in SIESTA of relying on sparse data structures. This seems the most obvious and memory-saving choice given the finite support of the basis set, but these sparse structures do not lend themselves to vectorization/streaming on GPUs. Thus there might be room to trade memory for speed in some areas of the code. Optimised sparse linear algebra libraries, critical for the code performance, for GPUs will be a key aspect of taking profit of the power of these architectures. An example in SIESTA is the Distributed Block Compressed Sparse Row (DBCSR) library (https://github.com/cp2k/dbcsr), which was recently incorporated into SIESTA to handle the distributed sparse linear algebra operations involved in the Order-N scaling solvers in SIESTA.
- Developments within the MaX series have improved the experience of pseudopotential handling (now with curated databases, exploiting the PSML format). But the generation of

appropriate basis sets in SIESTA remains a stumbling block for new users and a barrier for high-throughput calculations. In this area, we have some ideas regarding the use of machine-learning techniques, including proper representations of the chemical environment of an atom in a structure, to provide good-quality basis sets for arbitrary systems. In this respect, we have also implemented an on-the-fly basis set contraction scheme within SIESTA, based on previous work by Ozaki[1]. The method allows for the adjustment of the contraction coefficients during a molecular dynamics simulation, and we can systematically improve the quality of the contracted basis set by increasing the number of primitives considered. Since diagonalization and density matrix operations are only done in the contracted basis space, this leads to significant computational savings. In addition, the methodology can be used to automate the generation of an optimised low-cardinality basis set, further facilitating and enhancing the user experience of the code. In our implementation, we offer two differences from the original proposal made by Ozaki: first, contraction coefficients can be optimised for each MD step, not only at the beginning. Second, a contracted orbital can be any possible combination of primitive orbitals from the same atom, without necessarily keeping the quantum numbers m and l.

- A critical need for the use of the TranSIESTA module in problems related to quantum electronic transport (in electronic devices), and electrochemistry (the electrified electrode-electrolyte interface) in pre- and exascale infrastructures is the extension of the MPI parallelization to the distributed matrix inversion involved in the calculation of the Green's functions, as described in detail in deliverable D6.3 (similar kernels exist, in a different context, also within the GW and BSE calculations performed by Yambo). The hybrid parallelization scheme currently available in the code (distribution of energy points over MPI processes for the complex energy integration contour, and OpenMP directives within each MPI process) allows for the use of a maximum of around 1,000 cores, which is far from the massive parallelization required for the efficient use of these new infrastructures. Our current work of implementing a third layer of parallelization by distributing each matrix inversion (currently done by a single MPI process) to many cores using distributed linear algebra libraries, will multiply the number of cores by a factor of 100, allowing us to move to massive execution in very large number of nodes. Adapting distributed linear algebra libraries with support for execution in GPUs (as we did for the diagonalization in SIESTA, through the use of the GPU-ready ELPA and ELSI libraries) should allow us to swiftly move to massive execution in GPU-based infrastructures.

[1] Ozaki, T., Phys. Rev. B 67 (2003), 155108. DOI: 10.1103/PhysRevB.67.155108

4.2 Challenges at the software level

- The process of building and deployment of the recent version of SIESTA, incorporating the advances done in MaX towards its deployment in HPC infrastructures, is increasingly complex and problematic, due to the significant increase in the number of dependencies brought about by modularization. This affects demonstrators as the ramp-up to deploy on a new architecture was non-negligible, and also because some of the code interfacing needed in some of the demonstrators demands a fine control of building options. Progress on the building and deployment has been significant in the latter stages of the ending MaX project, and now we have a cmake-based scheme and spack scripts to ease the task. However, their further development, optimization and refinement are some of the main tasks to be fully developed during the next MaX phase, starting in 2023.

- Compiler support in some of the new machines and architectures is spotty, and there is still not a clear choice of programming model for the accelerators. Although the SIESTA team identified this problem in the context of its deployment in the new available HPC machines, it likely affects all of the MaX codes.

- The lack of a proper performance model affects decisions as to the best strategies for the actual running of the calculations (number of MPI processes, balance of OpenMP/MPI, GPU vs CPU, etc). This will be very important, not only for demonstrators, but for the end users of the codes, who would need guidance to choose among machine parameters and algorithms to maximise their return, be it in the form of minimum time-to-solution or minimum total cost or energy-to-solution.

5. FLEUR

5.1 Software challenges for FLEUR

Pre-exascale and exascale HPC infrastructures make considerable use of GPU accelerators. To fully exploit these hardware capabilities, in the FLEUR code computationally expensive code sections have been ported to such hardware by making use of OpenACC directives. The effectiveness and efficiency of this approach has been demonstrated.

From a deployment perspective, however, this implies a major challenge because the related software stacks from the hardware vendors are still unstable and partially incomplete. Basically, so far we could only utilise the NVIDIA compiler and software stack to deploy FLEUR on GPU architectures. In particular, our experiments on the AMD-based test setups we had access to had been unsuccessful. In some cases the compilers were not even able to handle the CPU host code and the GPU related code was clearly beyond the capabilities provided.

Also, the availability of GPU-ported versions of critical software libraries is limited. In particular, we could not identify a performant, reliable, and usable library for solving on GPUs the dense generalised Hermitian matrix eigenvalue problem for matrices distributed across many compute nodes. This affects a performance-critical code section and at the moment limits the adopted parallelization schemes such that this part can only be addressed on a single node. In practice, it narrows the set of efficiently addressable simulation challenges on this kind of hardware.

5.2 Hardware limitations for FLEUR

The performance and scalability of FLEUR is hindered by two major issues. On the one hand, most of our computational relevant kernels are limited by memory bandwidth and thus improvements can be hard to achieve. This might be mitigated to some extent by better connectivity between different CPUs or GPUs enabling further parallelization and data distribution. However, some of our kernels also intrinsically only show limited possibilities for such parallelization and hence we also will require some ”more classical HPC capabilities” with hardware featuring high memory bandwidth combined with a powerful CPU for such parts of the simulation. In general, the fact that we have not a single computationally relevant kernel, but a multitude of such code parts tends to induce different hardware challenges and hence a suitable combination of resources should be provided with the option to choose the most appropriate hardware for the different kernels provided by a fast data-transfer between these resources.

6. BigDFT

BigDFT code can run in two modes: an approach that scales cubically with the number of atoms, and a second one that is linear scaling (LS). In the LS approach the total workload, which is divided among MPI tasks, grows in proportion to the number of atoms, so large core counts are more naturally accessible to LS-BigDFT than by using cubic scaling approaches. This makes LS-BigDFT an excellent candidate for exploiting large scale HPC and enabling first-principles simulations of systems of thousands of atoms not accessible with traditional DFT codes. In the future it will be important to address limitations identified in the current MPI communication scheme it relies on, and to further increase the scale of the performance of the shared memory parallelisation, as defined through the following actions.

One key area of application for LS-BigDFT is in biology, where biomolecules are typically large by themselves, and, due to complex interactions with the environment, also need to be treated as even larger systems. Traditionally, such simulations have been performed within, e.g., the quantum mechanics/molecular mechanics (QM/MM) framework. However the ability to perform purely QM simulations leads to new opportunities for extracting information about biological systems, which cannot be obtained with a QM/MM approach. For example, the complexity reduction framework of LS-BigDFT enables the automatic partitioning of complex systems such as proteins into fragments, and the subsequent generation of graph networks quantifying fragment interactions, which may be used to, e.g., understand enzyme-substrate interactions. This forms a growing research area for LS-BigDFT, for which the BigDFT development team will engage with collaborators in the computational biology community. Through such efforts, we expect to bring in new users of BigDFT from diverse user communities, who will directly benefit from the proposed developments[2].

6.1 Replacement of One-Sided MPI Communications in LS-BigDFT

One-sided communications are used in BigDFT itself (notably the communication of the potential, as well as in a handful of other routines), and in CheSS, which defines its own sparse matrix format, performs sparse matrix algebra and communications, and contains the implementation of the Fermi Operator Expansion approach, which is well suited to the basis set used in LS-BigDFT, and is one of the key functionalities enabling LS cost.

LS-BigDFT uses a range of blocking and non-blocking MPI routines. Unlike cubic scaling BigDFT, this includes one-sided communications. One-sided MPI communications are relatively under-used, and are therefore less thoroughly tested. Previously, a bug was discovered in MPICH when running LS-BigDFT on BG/Q, with calculations giving incorrect results for certain parallel layouts. To identify the problem, internal test routines were written to check one-sided communications. A bugfix was introduced in MPICH, however similar problems have arisen when porting for instance to the ARCHER2 UK supercomputer, with

[2] In the following the reader can find some relevant publications on the subjects discussed above:

- L.E. Ratcliff et al., WIREs Comput. Mol. Sci. **7**, e1290 (2017)
- L.E. Ratcliff et al., J. Chem. Phys. **152**, 194110 (2020)
- L.E. Ratcliff, L. Genovese, S. Mohr T. Deutsch, J. Chem. Phys. **142**, 234105 (2015)
- S. Mohr, M. Masella, L.E. Ratcliff, L. Genovese, J. Chem. Theory Comput. **13**, 4079 (2017)
- S. Mohr et al., J. Chem. Theory Comput. **13**, 4684 (2017)
- W. Dawson et al., J. Chem. Theory Comput. **16**, 2952 (2020)

some calculations giving incorrect results (including NaN) and others hanging, while the internal test routine showed similar errors. Although workarounds are being explored, they are specific to the supercomputer and software environment being used.

This part of the standard, employed by LS-BigDFT, appears not to be tested frequently when a novel MPI library comes out, which may lead to inconveniences such as the need to file bug reports to the vendors. This happened a few times in the past years, even though the usage of the MPI communications in LS-BigDFT is compliant with the MPI standard. In the future we will therefore replace the one-sided MPI communication scheme used by LS-BigDFT, which is not widely used in large-scale production codes and for which in-practice unrobust MPI library and transport layer support has been found to limit portability and usability.

Sparse matrix algebra and related operations in LS-BigDFT are performed using the CheSS library, which is available both as part of the BigDFT repository and as a standalone library. Like the code in LS-BigDFT itself, CheSS also uses one-sided MPI communications, therefore future work should also be required in CheSS, in order to fully remove LS-BigDFT's reliance on one-sided communications. Since one of the key motivations for using one-sided communications is the ability to overlap communications with other operations, they will have to be replaced with traditional non-blocking communications (i.e. mpi_isend/mpi_irecv), preserving much of the current code structure.

Removing the reliance on one-sided communications will lead to a more robust version of LS-BigDFT, reducing the CPU time usage and additional developer and supercomputing support time associated with overcoming such problems when porting to a new machine.

6.2 Improvement of shared memory thread concurrency

LS-BigDFT was designed to exploit HPC from the beginning, with a long-standing hybrid MPI/OpenMP approach. However it has been several years since the performance of the low level OpenMP parallelisation has been assessed, with many of the routines having been designed or tested only on low thread counts. Currently, the OpenMP speedup is interesting up to 8-16 threads/MPI, however this performance strongly depends on a number of simulation parameters. The OpenMP parallelisation should then be revisited in some of the cases, by introducing, for instance, two-level hierarchy in the shared memory. Since the choice of number of nodes to be used for a given run is often based on memory limitations, not just parallel efficiency, these improvements aim at better exploiting the computing power of future architectures for current core counts, while also increasing the upper limit in the number of cores which can efficiently be used for a given problem size.

7. CP2K

Below we give an overview of the software challenges and hardware limitations that we observed working on the CP2K demonstor for RPA and double hybrid based MD simulations of condensed phase systems.

7.1 Software challenges for CP2K

CP2K is a large community code written in Fortran90 with OpenMP/MPI programming models. Relying a lot for its performance on the external libraries (such as DBCSR, COSMA, ELPA, FFTW, BLAS/LAPACK) CP2K itself is platform-agnostic, meaning that code does not contain specific GPU code or compiler pragmas such as OpenACC or CUDA Fortran. As such, the performance of CP2K depends sensibly on the performance of the underlying libraries. Most of the libraries mentioned above provide some of the linear-algebra functionality which is common for electronic structure codes. It is well known that distributed linear algebra algorithms are intrinsically communication-bound and the effort in scaling these algorithms should be directed towards the development and implementation of communication-optimality. Our work on COSMA (communication-optimal matrix multiplication library) and COSTA (communication-optimal shuffle and transpose library) indicates that this is a fruitful direction which should be pursued further. We observed two challenges while developing the RPA and MP2- based MD simulation functionality in CP2K:

- *Long term support for the development of domain-specific and general purpose libraries.* Writing initial proof-of-a-concept implementation is only part of the deal. A longer commitment is required to support the software products and enhance the functionality. We believe that long-term funding and the emergence of the Research Software Engineering (RSE) profession are a key solution to this problem.
- *Combinatorial explosion of the dependencies*. With at least two major CPU architectures, several NVIDIA and AMD GPU architectures, several BLAS/LAPACK/FFT providers, several MPI implementations, different Linux-based operating systems deployed on the HPC platforms, it becomes challenging to provide test coverage for all combinations. We believe that at least good software-building practices, such as using common build systems (for example, cmake) and package managers (for example, spack) can be a relevant step towards build reproducibility. The now-true statement “We are not paid for improving the build system” should

become false in the future. This goes back to scientific software-engineering as a recognised profession.

7.2 Hardware limitations for CP2K

The CP2K team did not observe any particular hardware limitations that impact the overall performance of the big simulations. We can wish for instantaneous network communication, but the network infrastructure develops at its pace. Compared to speed of network communication, the performance of the node-level hardware is already instantaneous. What we can mention here is the overall stability of modern and very complex HPC platforms that depend on numerous low-level drivers and middleware components together with the quality of the programming environments. Both have room for improvement and we have confidence that HPC vendors will continue their commitment in that regard.

8. Conclusions

In summary, the effort leading to the deployment of MaX demonstrators has also provided us with relevant insight into existing and future challenges related to pre- and exascale machines. In particular, in this Deliverable we examined the challenges posed by the MaX Demonstrators, as relevant use cases for the domain of Material Sciences, when running on current machine architectures and looking in projection at the next generation. Notably, many different remarks emerged as common from all the experiences, even from codes that use kernels that have a substantially different nature (from CPU-intensive to BW-limited). In the following, we will try to summarise the most relevant points emerged from the experience of the code owners when running the demonstrators use-cases:

- In general, it emerged that most of the concerns are about the software stack rather than the system level. In all cases, the developers' communities have shown a good readiness to change and adapt their codes when encountering a new hardware architecture. However, the lack of a universal standard for accelerator devices involved a serious trouble for the developers, also considering the huge investment in terms of porting effort.
- Also, the diversity of accelerators, each one with its own software stack, reflected on the difficulty of managing the installation of the applications with their required libraries. This pushed on the adoption of tools like Spack or with the use of CMake as a building tool.
- The evolution and implementation of programming paradigm standards is challenged by the fast technological evolution of HPC hardware. Large scale systems require a constant evolution of the message-passing protocols (MPI, ZeroMQ, etc.) and new interconnects (GPU-GPU, CPU-GPU) push changes into programming languages. All these innovations should be implemented as soon as possible in a stable and efficient way in compilers and programming libraries.
- The concept of "separation of concern" and the consequent adoption of specialised libraries is well implemented in all MaX flagship codes. Remarkably, most of the codes' concerns are about the lack of updated and stable libraries on the HPC systems that could permit a good exploitation of the accelerators. In particular, the absence of a de-facto standard for distributed linear algebra library for GPUs is acknowledged as a critical limiting factor. In the case of CP2K, where such domain-specific libraries were produced by the CP2K developers, the challenge is given by the necessity of establishing and keeping a long-term maintenance and support.
- Large availability of computing resources is also enabling new approaches for computational sciences other than "classical" HPC. New scientific questions can be answered by a combination of HPC with Artificial Intelligence techniques or using mixed HPC and High-Throughput computations. The next generations of

supercomputers should be able to manage these complex workflows enforcing new concepts of resilience and malleability of resources.

In conclusion, our experience with the MaX demonstrators has shown how the codes' developers, with the support of MaX, have proven their readiness to face the challenges of new hardware architectures. We wish that soon the limiting factors, here discussed, will be removed, enabling this community to leverage on the next Exascale systems to respond to more complex and difficult scientific challenges.