D1.3

Second report on software architecture and implementation planning

P. Delugas, F. Affinito, A. Alcaraz, O. Baseggio, L. Bellentani, C. Cardoso, I. Carnimeo, F. Ferrari Ruffino, A. Ferretti, A. García, L. Genovese, P. Giannozzi, R. Grima, J. Gutiérrez Moreno, G. Michalicek, M. Montagna, S. Orlandini, F. Paleari, F. Pedron, D. Sangalli, G. Sesti, N. Spallanzani, D. Varsano, F. Vaverka, N. Wittemeier, D. Wortmann, S. Baroni

Due date of deliverable	30/06/2025 (**month 30**)
Actual submission date	30/06/2025
Final Version	16/05/2025
Lead beneficiary	SISSA (participant number 2)
Dissemination level	PU - Public

Document information

Project acronym	MAX
Project full title	Materials Design at the Exascale
Research Action Project type	Centres of Excellence for HPC applications
EuroHPC Grant agreement no.	101093374
Project starting/end date	01/01/2023 (month 1) / 31/12/2026 (month 48)
Website	http://www.max-centre.eu
Deliverable no.	D1.3

Authors	P. Delugas, F. Affinito, A. Alcaraz, O. Baseggio, L. Bellentani, C. Cardoso, I. Carnimeo, F. Ferrari Ruffino, A. Ferretti, A. García, L. Genovese, P. Giannozzi, R. Grima, J. Gutiérrez Moreno, G. Michalicek, M. Montagna, S. Orlandini, F. Paleari, F. Pedron, D. Sangalli, G. Sesti, N. Spallanzani, D. Varsano, F. Vaverka, N. Wittemeier, D. Wortmann, S. Baroni

To be cited as	Delugas et al. (2025): Second report on software architecture and implementation planning. Deliverable D1.3 of the HORIZON-EUROHPC-JU-2021-COE-01 project MAX (final version as of 29/06/2025). EC grant agreement no: 101093374, SISSA, Trieste, Italy.

Disclaimer

Contents

Executive Summary

In this report, we present the update of the software development plan and discuss the main novelties, concepts, and needs that have arisen in our community and groups.

The Software development plan is still current, on schedule, and effective; the Lighthouse developers propose no significant updates. Regarding the discussion of new needs, concepts, and actions planned for the future, one crucial point concerns the emerging technologies and architectures for which developers are already planning to port updates. To the extent possible, these plans should be integrated into the current MaX work on the codes. Another point of concern is the arrival on the market of GPUs with massive throughput at lower precision. This has attracted the attention of WP1 developers for two main reasons: On the one hand, they want to prevent the impoverishment of the computational power in future GPU-accelerated machines; on the other hand, finding the way to exploit this throughput will give advantages in terms of energy efficiency and may enable the usage of user entry GPU, such as cards used for gaming or AI inference. All groups are thus experimenting on these aspects, collecting use cases, and staying on the lookout for any upcoming vendor libraries, which will very likely be the most efficient solution for exploiting low-precision GPU cores.

One common update to the plans is the development, collaboration with the community, and other types of support for the `EasyBuild` package manager. This support builds upon the existing support for `Spack` that was already present in the first SDP. The reason for this addiction is primarily due to the clear focus of WP1 developers to address – and deploy on – EuroHPC machines, that have in many cases adopted `EasyBuild` as the default package manager and that will be very likely the pivot around which to leverage for the distributed CI/CD in these systems, which is a long time target of WP1 together with WP3.

- **QUANTUM ESPRESSO** updates on optimizing GPU acceleration for small-size calculations; improving FFTXlib communication efficiency through NCCL integration and adaptive batching; refactoring band parallelism with non-blocking MPI communications; enhancing build systems and CI/CD pipelines.
- **SIESTA** updates focus on enhancing ELPA library integration and mixed-precision arithmetic for reduced-precision hardware; improving parallel efficiency by optimising MPI-GPU balance using MPS/NCCL frameworks and developing configurable MPI teams for k-point/spin parallelisation. Additional planned work includes streamlined deployment through Spack/EasyBuild and enhanced interoperability features, such as improved I/O handling and API enhancements.
- **FLEUR** updates aims at the enhancement of maintainability, scalability and add scientific capabilities primarily based on linear response. The plan to add to the code mixed-precision arithmetic, refactoring some program internals as the charge density generation, and the GPU acceleration of their newly introduced DFPT features.
- **BIGDFT** focuses its updates on improving MPI communications, adding and implementing non-blocking reduction. They have also updated their plan giving increasingly more room to SYCL-based offloading.

- **YAMBO** main updates are on the refactoring of compute-intensive kernels (such as the calculation of the irreducible polarizability) by using level 3 BLAS operations, in turn also enabling the future exploitation of reduced precision HW; new APIs for distributed linear algebra; the `YamboPy` interoperability tool; new optimized property calculators for the scientific workflow such as self-consistent GW, electron-ion dynamics and convergence accelerators.

1 Introduction

The development plan outlined in D1.1 provides a comprehensive roadmap for the third phase of the MAX CoE. Its primary focus is the realisation of scientific workflows designed to leverage exascale computing systems for scientific calculations that were previously unattainable in the fields of electronic structure and materials science. The plan addresses all essential aspects required to achieve this goal, including making the codes and workflows deployable modern HPC machines (notably including exascale candidates), implementing local parallelism with threads or accelerators, ensuring internode parallelism via MPI, adhering to best practices in software engineering, and developing both the scientific property calculators and the interoperability layer needed for the workflows.

While the development of these workflows is the main objective, the plan makes it clear that they are derived from the MAX lighthouse codes: QUANTUM ESPRESSO , SIESTA, FLEUR, BIGDFT, and YAMBO. Alongside adapting these codes for the workflows, a key component of the plan is optimizing the codes for exascale infrastructures. The optimization of these lighthouse codes for exascale machines is crucial. This is a result as significant as the scientific workflows themselves, as it enables large communities of scientists and developers to efficiently access pre-exascale and exascale systems. This access helps maintain the vital connection between the materials science community and high-performance computing (HPC), which, during this rapid evolution of HPC technologies, was at risk of disruption.

Over two-thirds of the original plan implementation has been completed, with many milestones achieved and key objectives addressed. This update allows for a thorough evaluation of the implemented components, assessing their validity, effectiveness, and any potential areas requiring further development, integration, or refinement. It also highlights opportunities to enhance these components, leveraging new libraries, communication methods, programming models, and techniques. This integration, along with updates to the plan, will outline our response to evolving HPC features and the needs of the user community in adapting to these advanced systems.

A dedicated section addresses a major topic: the growing shift in GPU evolution, specifically the transition from double-precision computational power to lower-precision data types. This shift is primarily driven by the increasing demand for computational power in artificial intelligence (AI) and machine learning, which is becoming more significant even in technical and scientific fields. In addition to market forces, this shift is also motivated by energy efficiency considerations and the potential for increased computational power, including for double-precision scientific calculations, provided the codes are adapted accordingly. Such adaptations would enable scientific applications to exploit entry-level GPUs in laptops and workstations for these calculations.

The update also discusses various strategies for optimising communication. These include refactoring communication schemes to replace blocking operations with asynchronous ones, or using vendor-specific communication libraries. While the latter may introduce additional complexity and require more configuration, they can significantly reduce communication overhead. The ability to incorporate such detailed updates is a result of WP1 ongoing collaboration with WP3 and WP4, particularly through the use of mini-apps for co-design.

The report is structured into code-specific sections, each detailing the relevant up-

dates. These are generally aligned with one of the four tasks of WP1, though many updates span multiple tasks. After the four code-focused sections, we present a general overview of WP1's direction, particularly regarding the exploration of emerging architectures and the exploitation of AI-optimized GPUs. The report concludes with a section summarizing key insights and future steps.

2 QUANTUM ESPRESSO

This section provides updates to the development plan for Quantum ESPRESSO. As per the original plan, we will organize these updates according to the four tasks outlined in the WP1 DoA.

- **Task 1:** We present a brief update on the status of OpenMP offloading, the adoption of libraries for double-precision accelerated linear algebra on the reduced precision cores of upcoming AI-optimized accelerators, and mention potential interactions with the DARE project's plans for experimenting with QUANTUM ESPRESSO on their Risc-V-VEC processor using OpenMP6 directives.
- **Task 2:** For completeness, we refer to the results of our interactions with WP3, which have already been reported in their M18 deliverable. Here, we translate their proof-of-concept work into code actions aimed at optimising inter-node GPU communications for our 3D FFTW library, FFTXlib. We will also focus on optimising band parallelism by addressing workload imbalances, exceeding memory footprints, and mitigating synchronisation impacts.
- **Task 3:** We confirm the plans laid out in the initial development plan, providing additional details about the work planned for unit testing. Noteworthy developments include our efforts to contribute to the `EasyBuild` recipes by enhancing their integration with our build systems. Additionally, we present improvements for the Autoconf toolchain, particularly enabling the V-PATH feature. This will allow for the separation of build directories from the source folders, as is already done with the CMake build system.
- **Task 4:** We provide details on the interoperability hooks, the status of the Non-Adiabatic Dynamics workflow, and plans for improving and extending meta-GGA support within QUANTUM ESPRESSO.

2.1 Task 1 – Node-level Performance

2.1.1 Porting of QE on LUMI and AMD GPU

Status Update. As of early 2024, we have successfully ported the QUANTUM ESPRESSO suite to AMD GPUs and the LUMI software stack. A notable milestone was the acceleration of the FFTXlib library, which is essential for Fast Fourier Transform operations in plane-wave calculations. This porting effort primarily utilized OpenMP 5.0+ directives, while the HIP language was leveraged in performance-critical sections, with the hipFFT backend handling 1D and 2D operations. The suite was compiled and executed on LUMI using the Cray 15 and ROCm 5.2 software stacks. The first version released for LUMI,

based on version 7.2, demonstrated comparable performance to the Leonardo machine. Currently, active development is ongoing in the `develop_omp5` branch, which is synchronised with the main development branch. Key achievements so far include:

- Successful release of version 7.4.1omp for LUMI, incorporating new features and GPU-AWARE MPI testing.
- Continuous updates to the branch to align with newer releases of the Cray, ROCm, and LUMI software stacks.

SDP un to M48. For the upcoming period, we have identified several key focus areas to enhance the efficiency and functionality of the code for AMD GPUs, thereby making them fully accessible to the user community.

- **Upgrade Compiler Support:** We will transition the develop_omp5 branch to support Cray 19, working with the LUMI support team and developers from AMD and Cray to ensure compatibility and optimisation.
- **Expand Scientific Features:** We plan to extend OpenMP 5.0+ offloading in PWscf by adding new scientific features, including components related to ultra-soft pseudopotentials and PAW potentials. The porting of the PHonon code will also continue.
- **Enhance Eigensolver Performance:** We are actively addressing the performance of the eigensolver, currently reliant on the CPU. Adapting the iterative diagonalization algorithm to work with a Scalapack eigensolver distributed across CPU cores will optimize performance, and future tests will focus on integrating updates from rocSOLVER.
- **Evaluate New Compiler:** We will continue testing new compilers, including AMD's optimized flang-new, for both scalar and GPU-targeted builds, broadening our compilation options beyond Cray compilers.

2.1.2 Targeting GPUs with low-precision throughput

Many of the new GPU cards that will populate the HPC market, as well as the single-user entry-level market, will prioritise the throughput of AI workloads and thus trade reduced FP64 computational power for increased computational power for low-precision and integer operations. This is illustrated in the historical GPU performance data shown in Figure 7 –courtesy of NVIDIA Corp. – FP64 performance improved from the Volta to the Ampere and Hopper architectures, but begins to decline with the introduction of the Blackwell generation, which shows lower FP64 capabilities than its predecessor.

By a preliminary analysis conducted in collaboration with WP3 and WP4 and with GPU vendors, we estimate that from the QUANTUM ESPRESSO point of view, this trend will not significantly affect the distributed FFT workload that, being memory-bound, it exploits the GPU primarily for its memory bandwidth and throughput capacity rather than its computational power. A significant performance impairment is instead expected for matrix operations and for many compute-intensive loops. Such an impact

is common to all compute-bound applications that require double precision. Recognising this challenge as an opportunity, the key is to leverage the enormous computational throughput that modern graphics cards offer for integer and low-precision arithmetic.

To address this, HPC specialists, academic researchers, and hardware vendors are creating dedicated numerical methods – including those based on Ozaki schemes – that perform high-precision operations using low-precision data formats. Notably, these techniques preserve the numerical accuracy of calculations while requiring only a minimal set of input parameters, which can ultimately be concealed from end users. In collaboration with NVIDIA, we have started experimenting with one of these libraries. In Figure 1, numerical tests recently performed on the QUANTUM ESPRESSO code by Massimiliano Fatica's groups, on a Blackwell RTX card with and without an Ozaki-type emulator, are compared with a single execution of the same test case on an A100 card on the Leonardo cluster. In all cases, serial executions have been performed, neglecting the MPI features at this stage.

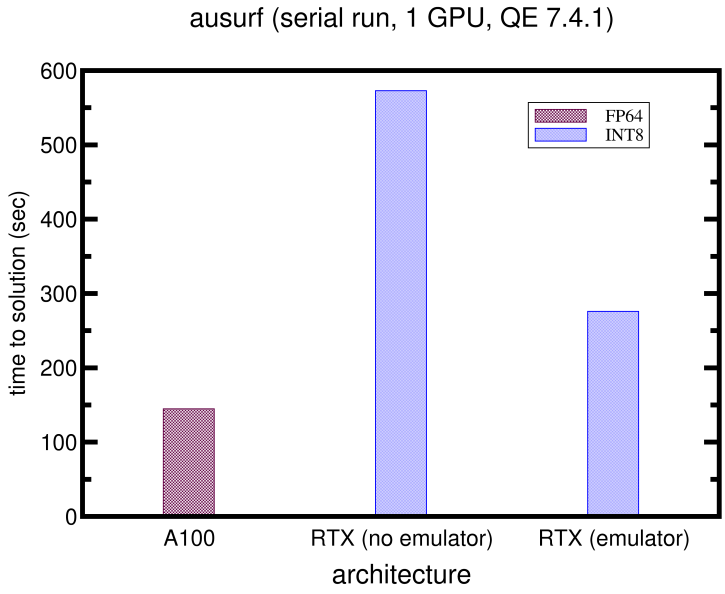

Figure 1: Data on Blackwell RTX cards kindly provided by Massimiliano Fatica, NVIDIA Corp.

Beyond the exact performance numbers, which may differ depending on the specific Blackwell architecture type (here, a very small gaming card has been used), the figure is intended to highlight the broader trend and the effect of the emulator of restoring the performance boost of double precision algorithms executed on single precision tensor cores. After the experimental phase, these kernels will be directly available as built-in features of cuBLAS libraries, accessible through compilation flags.

We are currently working to incorporate support for emulation into the QUANTUM ESPRESSO build system, as it will be released in a future version of the HPC SDK software stack. We are also analysing other possible implications of using single-precision algorithms in different parts of the QUANTUM ESPRESSO code. To fully exploit such libraries, we have also initiated the reorganisation of loops to express them in terms

of matrix-matrix and matrix-vector operations that can be dispatched to the reduced-precision GPU dedicated cores. Such last action will also benefit the general portability of the codes' performance because the compute-intensive kernels will run using vendor-optimised backends.

2.1.3 Targeting single-GPU workloads: speedup and/or batching of small-size computations.

Current local parallelism in GPU versions of Quantum ESPRESSO is optimised for medium-to-large systems. These systems possess a sufficiently large number of plane waves to saturate GPU throughput during computational loops and 3D FFT operations with FFTXlib. Additionally, they contain enough bands to ensure that `cuSolver` operation throughput compensates for `cuSolver` latency.

However, small systems fail to meet these requirements. For such systems, loops and FFTs execute more slowly on GPUs compared to CPU execution, while inner matrices are too small for efficient GPU-based `cusolver` processing. Two approaches are currently employed to address this limitation:

Pool Parallelism with GPU Oversubscription. This strategy leverages the numerous k-points required by small system calculations (typically tens to hundreds for metals) by implementing pool parallelism with GPU oversubscription. NVIDIA MPS (Multi-Process Service) is utilized to enhance computational efficiency.

Modified SCF Loop Implementation. Dal Corso and Gong [1] recently proposed an alternative SCF loop execution that processes both $\mathbf{k}$-point and plane wave loops on the GPU. This implementation inlines matrix diagonalisation using a GPU-optimised version of the LAPACK zsyegv routine. This solution can be combined with pool parallelism and MPS oversubscription, yielding excellent performance results. The primary limitation of this approach is its requirement for CUDA Fortran implementation, which conflicts with our strategic goal of transitioning to OpenACC for all high-level code components. This transition aims to ensure easy portability to other accelerators and vector machines using OpenMP constructs.

We plan to refactor the Dal Corso and Gong implementation [1] to maintain an OpenACC directive-based approach while optimising GPU utilisation for small systems. For this `OpenACC` refactoring, we employ a hierarchical batching strategy with three levels of organisation. At the pool level, K-points are collected into batches, which are then processed sequentially through a batch-level loop. Each batch is further subdivided into sub-batches, enabling asynchronous processing. The core implementation uses an asynchronous execution pipeline designed to maximise GPU utilisation. Sub-batches are processed asynchronously using OpenACC streams, allowing the diagonalisation of the current sub-batch to overlap with processing of subsequent batches. This overlap strategy effectively hides latency and ensures continuous GPU engagement.

Two diagonalisation approaches will be implemented and benchmarked against each other. The GPU-based collective diagonalisation utilises cuSolver to diagonalise collected matrices from each sub-batch, making it optimal when batch sizes provide sufficient computational density. Alternatively, the CPU-based individual diagonalisation

performs CPU diagonalisation of individual small matrices for each k-point, exploiting asynchronous execution to hide data transfer overhead.

The implementation offers several key benefits. Portability is maintained through the OpenACC directive approach, ensuring compatibility across multiple accelerator architectures. The hierarchical batching provides scalability that adapts to different system sizes and GPU capabilities. Efficiency is achieved through asynchronous overlap that maximises resource utilisation, while flexibility comes from dual diagonalisation paths that allow optimisation based on specific system characteristics. This same strategy can be easily translated into an analogous OpenMP one, which will use task parallelism on host instead of on device streams.

Critical technical considerations include careful GPU memory management for batched operations, dynamic load balancing that adjusts batch and sub-batch sizes based on system characteristics, and comprehensive performance profiling to determine the optimal diagonalisation strategy for different scenarios.

2.2 Task 2 – Parallel Efficiency

2.2.1 Improved communications in FFTXlib

The plan for enhancing communication efficiency in Quantum ESPRESSO will first and foremost target FFTXlib, which is essential for parallelism in its applications. FFTXlib's performance depends on communication efficiency and is suited for multi-FFT batching due to its multi-band structure. Improvements include:

- Implementing topology-aware APIs like NCCL;
- Optimizing batching techniques in FFTXlib in order to increase the joint throughput and increase the computation/communication overlap.

The details of these planned improvements are based on the tests and results performed within Task 2 of WP3 and have been partially reported in their deliverable (M18). We report and update here those results that, thanks to QE modularity, will be almost directly applicable to the production code. The first decision made for updating the plan was in fact to continue using our distributed library instead of the general-purpose multi-node GPUs. To motivate this choice and quantify the expected improvements, we briefly present the results of our comparison of FFTXlib, refactored with NCCL, versus Nvidia cuFFTMp.

To achieve this, we have written a mini-app to disentangle FFTXlib from the main code and enable independent testing. The *R*&*G* data distribution of FFTXlib and QUANTUM ESPRESSO is not fully reproducible in cuFFTMp; therefore, we have chosen the most similar distribution, which is the 2D slab distribution. We have also added the option to remove the FFTXlib spherical cutoff in reciprocal space, allowing us to isolate and estimate the contribution of the cutoff to the overall performance. This enables a fair comparison with cuFFTMp, which does not implement this option. In Fig. 2 the results of the benchmark are shown:

The introduction of the cutoff in reciprocal space (the radius is typically 1/4 of the square grid side) makes FFTXlib outperform cuFFTMp in the range of interest for all four grids considered. However, cuFFT is highly optimized, and this is clear from the

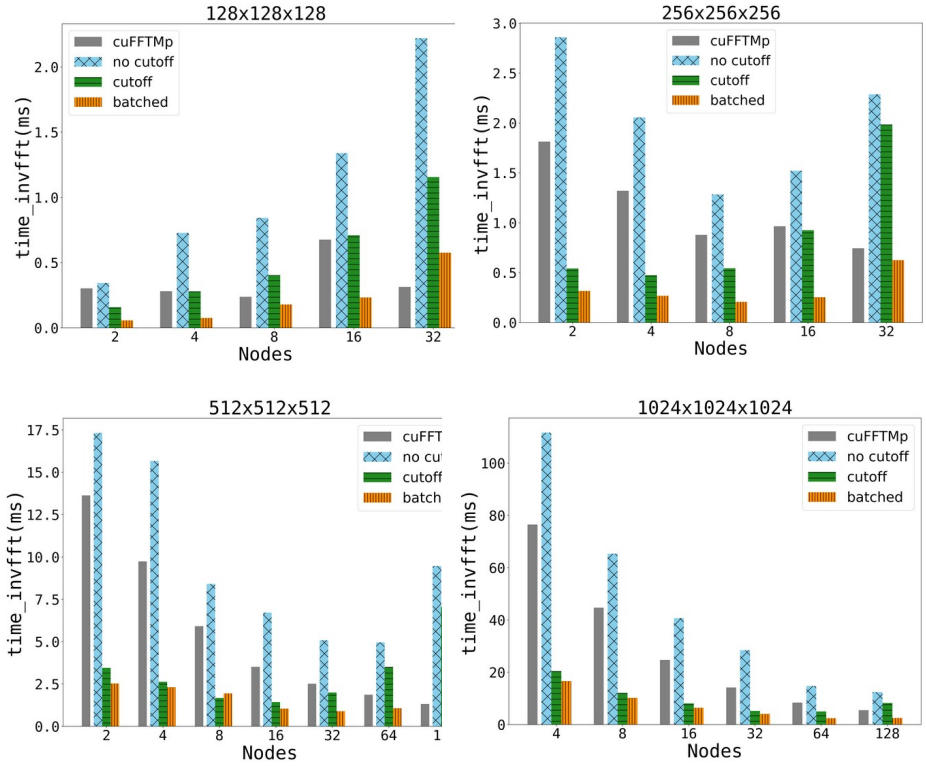

Figure 2: Benchmark comparison of the average FFT execution time between cuFFTMp API (gray columns) and QUANTUM ESPRESSO FFTXlib, with three different options. Cyan bars are related to FFTs without reciprocal space cutoff (for direct comparison with cuFFTMp), green bars to FFTs with cutoff enabled and orange bars to FFTs with cutoff and batching enabled (with standard batch-size of 16 bands)).

comparison with FFTXlib without cutoff or batching (blue bars). This shows that there is room for improvement at the low level. The inclusion of the NCCL library in FFTXlib has been preceded by a throughput analysis performed with an ad-hoc mini-app, which reproduces the all-to-all pattern of FFTXlib, consisting of a loop of asynchronous ISEND and IRECV calls over all processes.

The NCCL-based implementation of this all-to-all algorithm closely follows the MPI one, but relies on a dedicated GPU stream and must be enclosed in a so-called group call, so that each send/recv merges into a single GPU kernel, where the entire collective communication is managed. We mostly focused on the Leonardo machine, equipped with Nvidia A100 GPUs and Mellanox InfiniBand HDR 200 Gbps, and compared the Nvidia HPC-X backend library with the NCCL one (more recently, we have applied the same analysis on Marenostrum 5 with H100 GPUs and NDR200 InfiniBand 800 Gbps). In Fig. 3, the all-to-all throughput on Leonardo for different numbers of nodes is shown both for NCCL and HPC-X backends.

FFT buffers in a typical QE run hardly overcome the size of 10Mb. Consequently, the maximum communication throughput is rarely reached in most cases. For the same

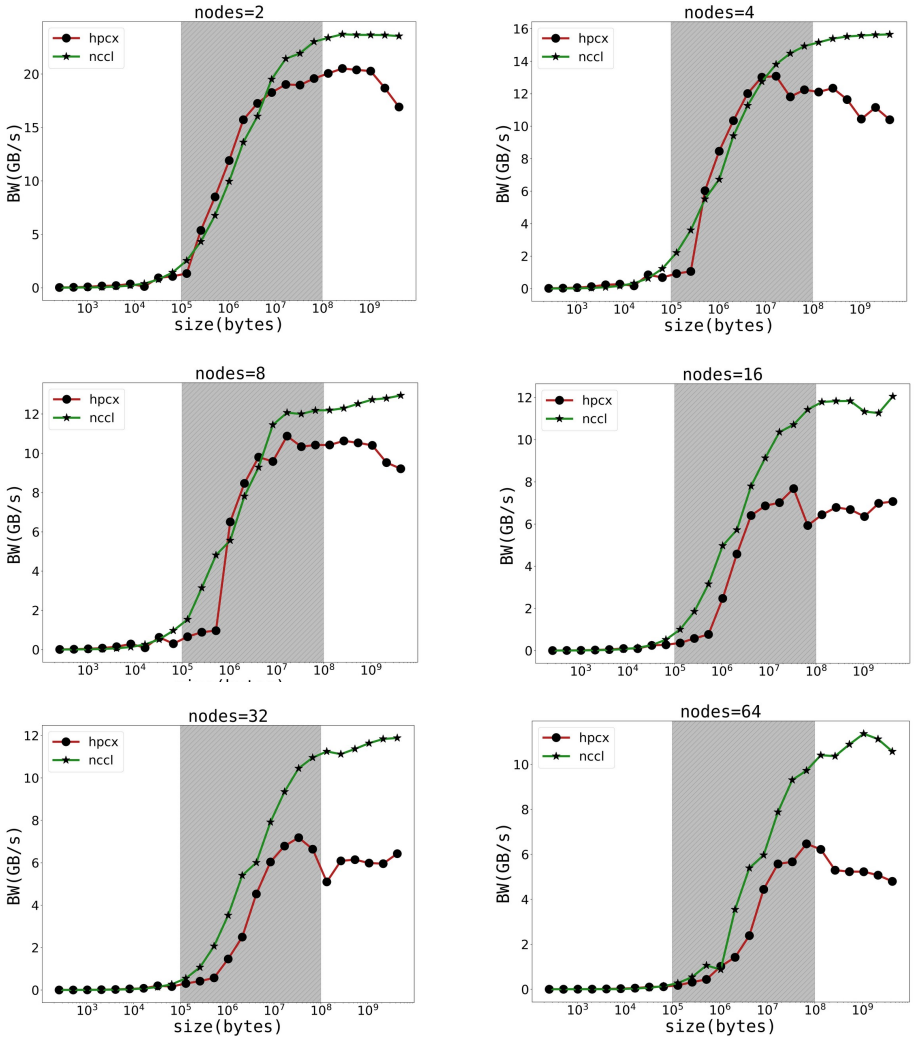

Figure 3: Throughput of the all-to-all communication pattern (made of ISEND and IRECV) of QUANTUM ESPRESSO FFTXlib measured on Leonardo machine (CINECA) with, respectively, the HPC-X communication library (red line) and the NCCL one (green line). The gray sector highlights the region of interest for QUANTUM ESPRESSO in terms of buffer size.

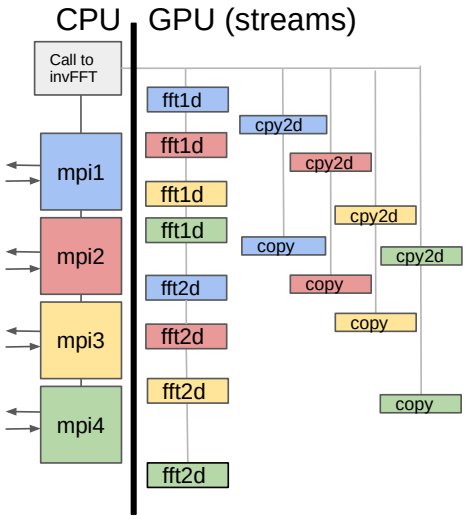

Figure 4: Schematic depiction of the execution flow of FFTXlib batched FFTs. Each colour corresponds to a different sub-batch. While the mpi communications of a given sub-batch are executed, the computation and data movements of the other sub-batches are performed on dedicated GPU streams.

reason, the advantage gained from using a topology-aware library, such as NCCl (shown in Fig. I), cannot be fully exploited, as the throughput difference with respect to HPC-X reaches its maximum for larger buffers. To overcome this obstacle, we relied on a variation of the FFT batching scheme implemented in FFTXlib. The standard scheme is depicted in Fig. 4.

The wavefunction to be Fourier transformed is segmented into batches and sub-batches, each containing a fixed number of bands, and sub-batches are processed asynchronously by means of GPU streams. In the batched algorithm all the 1d and 2d FFTs needed to fully Fourier transform a batch are performed on one dedicated stream (usually stream 0), while data movement and copies are performed on different streams, one for each sub-batch. MPI communications are initiated asynchronously for each sub-batch with respect to the computation and data movements related to all other sub-batches. In this way, the overlap in time between communication and computation is brought out at its maximum.

The original implementation is based on two hard-coded parameters, the batch size and sub-batch size of, respectively, 16 and 4 bands. While these are sufficient to ensure the overlap between computation and communication in most of the cases, they are not enough to maximize the collective all-to-all communication BW. Therefore, we have introduced an adaptive batching. Having a (typically) very large total number of bands to be processed, we can enlarge the batch and subbatch size at will in order to always be able to reach the buffer size where the all-to-all throughput is maximum, as depicted in Fig. 6.

In Fig. 5 the time to solution of inverse FFTs for different grid sizes is shown and compared between the standard batching scheme and the adaptive one (both with HPC-X and NCCL backends). While the standard batching is sufficient to achieve full commu-

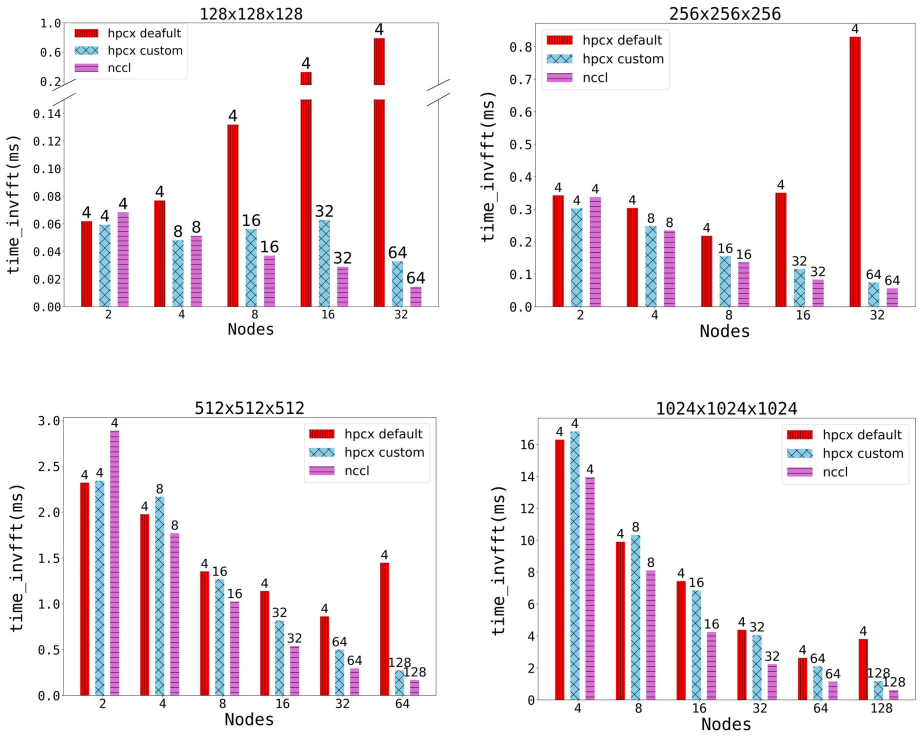

Figure 5: Comparison of the FFT time to solution of four different grids for three different implementations. Red bars are related to the standard batched FFT (one batch of 16 bands, with sub-batches of 4) with HPC-X communication library, cyan bars to FFTs with adaptive batching with HPC-X and violet bars to FFTs with adaptive batching and NCCL library.

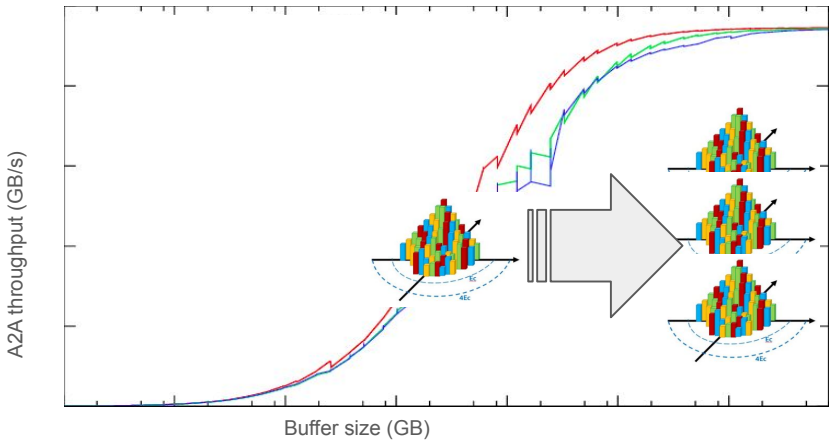

Figure 6: Schematic depiction of the adaptive batching concept. By having a large number of relatively small FFTs, multiple buffers can be merged together in order to maximize the collective communication bandwidth.

nication bandwidth (BW) in cases with a low number of nodes, this is no longer true above a certain threshold, which depends on the grid size. For example, for grids of 128^3 and 256^3, standard batching fails to ensure strong scaling after 8 and 16 nodes, respectively. After this threshold, the communication buffers become too small to maintain the communication in the BW regime and enter into the inefficient latency regime. The same behaviour is observed for 512^3 and 1024^3 grids after, respectively, 64 and 128 nodes. Adaptive batching, on the contrary, ensures good scaling because it keeps buffers constantly large enough to remain in the BW regime even at a high number of nodes.

2.2.2 Band parallelism

The second activity in the task is the reorganisation of the band parallelism in the suite. Thanks to its modular structure, QUANTUM ESPRESSO parallelisation schemes are reproduced and reused in most of the applications. For this reason, the plans for this activity are scheduled to initially develop a few new general tools, which will then be used with analogous schematics across all applications. Many of the significant workloads in `qe` are iterative or nested iterative procedures. This poses essential limitations in ex-ante distribution of the workloads. For this reason, the general utilities will not only have tools to distribute work and communicate data across band groups, but they will also have to implement methods for the runtime evaluation and re-equilibration of the work balance.

The general schematic of the refactored band parallelisation replaces the current complete sharing of the band data that is made by the allocation of all band data in all the nodes and updating these band structures using blocking `MPI_Allgther` or `MPI_allreduce` with a scheme where each band group stores its assigned blocks plus hosted buffers containing the data from other band groups that are needed for performing all two bands operations. Hosted buffers will be updated using non-blocking point-to-point MPI APIs, allowing for overlap between computation and communication.

When running QE applications, the code frequently needs to reorganise data across different parallel computing arrangements. For instance, during iterative solving, the program calculates Hamiltonian matrix elements using one type of data distribution (band group distribution). However, these matrix elements then need to be redistributed either to the MPI processes handling parallel linear algebra operations or collected onto the specific MPI processes that will run GPU-based solvers.

The new refactored band-group approach eliminates the previous method of gathering all data onto a single MPI process. Instead, this redistribution now requires a sophisticated collective operation involving multiple processes working together, which must be carefully optimised for performance. The general utilities and schematics will thus feature:

- Utilities for blocking and distributing work and data with:
  - contiguous band block distribution
  - cyclic round robin band block distribution
  - dynamic block assignment
- A smart `mp_barrier` utility that can determine the status of work balance using the MPI wait times of each group and provide an array of weights that can be given as input to the block distribution utilities.

- Data distribution and exchange utilities:
  - General API for non-blocking update of the host buffers
  - Utility for performing butterfly communications for performing non-commutative two-band operations and
  - Utility for efficient transposition of data from band-group distribution scheme to ortho-group distribution scheme
  - General APIs and descriptors for efficient transposition between different distribution schemes.

2.3 Task 3 – Software Engineering

The planned action for software engineering good practice continues as scheduled. We provide a brief update below on the status and news regarding this action.

Build systems and package managers. We currently provide two alternative build systems, one based on `autoconf` tools and one based on `CMake`, both of which support all the leading tool chains. We confirm the original plan of keeping both of them up to date and functional because they offer alternative advantages.

- `autoconf` generates a well-schematized and easily manageable and changeable include file, imported by a fixed `Makefile` placed in each source directory. As such, this build system also provides an emergency tool for manual compilation with new architectures and tool chains. We are working at enabling `autoconf` compilation outside the source directory.
- `CMake` build system offers the advantage of a more modular interaction with external packages and toolchains, and for this reason, is the preferred build system employed by package managers such `spack` and `EasyBuild` that need to integrate seamlessly QUANTUM ESPRESSO and its dependencies within the insulated environments.

One small planned intervention for what concerns package managers is the plan to insert in the recipes the option for building the code using the released tarballs from QUANTUM ESPRESSO website instead of the packages produced automatically by `GitLab` or `GitHub`.

CI/CD . Our CI pipeline is based on `GitLab` runners. Triggered by all merges, this script tests the compilation with `autoconf` and `CMake`. With the latter build, the `CI` also runs unit tests for the main mathematical libraries. We also verify the numerical correctness of the deployment on `x86_64` and on NVIDIA-accelerated machines using a regression suite of approximately 300 tests. Due to the increasing number of XML files our applications are expected to produce, we have developed a new testing tool that can run in parallel with the CD tests and verify the correctness of the XML files generated, ensuring they adhere to the specified schemas. Currently, this testing tool is used to verify only the XML files produced by `pw.x`; however, we plan to extend it to other XML and status files as well.

2.4 Task 4 – New features and Interoperability

2.4.1 Non-adiabatic molecular dynamics for excited states

The activities planned for T1.4 focus on the realisation of the scientific components for the Non-Adiabatic-Molecular-Dynamics framework and on the expansion of the general interoperability features of QUANTUM ESPRESSO .

Scientific features. Concerning the scientific features, the plan presented in D1.1 is on schedule. We have achieved our first milestone with the delivery of the calculator of the excited states energy gradients, which has been delivered in a reserved `GitLab` branch and is now available for testers and scientific collaborators. Current work is thus dedicated to:

- Further scientific features:
  - Enabling spin-flip transitions in `turbo_davidson` and spin-flipped excited stated energy gradients.
  - Excited States Energy gradients with hybrid functionals.
  - Calculator of excited states adiabatic coupling
- refactoring of the molecular dynamics feature of QE with alignment of `pw.x` and `cp.x` for a general dynamics engine.

Testing the excited state energy gradient. For this purpose, we are using an experimental dynamics and relaxation engine. The work on this engine also provides a first proof of concept for our general molecular dynamics framework that will be included in `pw.x`. At this notice, we also want to remark that some collaborators, independent from the MAX CoE, have recently added an engine for performing path integral molecular dynamics. We are currently studying how to include this new feature in the upcoming general dynamics framework.

QUANTUM ESPRESSO general interoperability architecture. We previously outlined the general interoperability architecture for QE in deliverable D2.2 of Work Package 2. This architecture builds on the most successful interoperability features that currently connect applications within the QE suite. Key examples of existing seamless integration include molecular dynamics through `cp.x`, the computational chain linking `pw.x` to linear response codes, and Nudged Elastic Bands (NEB) calculations—all designed to work smoothly within complex scientific workflows. Through our extensive work in the MAX project, we have also established robust interoperability between QE applications and workflow managers such as `AiiDA`. This interoperability framework relies on standardised data formats, specifically `XML` and `HDF5`, to ensure structured input/output operations.

The planned activities target extending and optimising these features. While duly discussed first under the supervision of WP2, the code implementation is an endeavour of WP1. For this reason, while we report here the main code-related points of the plan:

Checkpointing and semaphores. Some of our applications, designed specifically for workflow integration – including `ph.x`, `neb.x`, and `cp.x` – serve as the foundation for our broader checkpointing and semaphore system implementation across all scientific workflow components.

Formatted and standardised I/O. Our current I/O infrastructure leverages XML for compact datasets and HDF5 for large-scale binary data, supported by publicly available XSD schemas. We have developed a comprehensive Python toolchain that automatically generates Fortran modules for XML file operations, enabling automated parsing and calculation input generation. For HDF5 integration, we maintain a specialised library providing transparent file interaction capabilities, supporting metadata operations, multi-precision numerical datasets, and hyperslab data access patterns.

Error logging. To strengthen workflow manager integration, we are implementing an enhanced logging system using a two-level YAML format designed for automated parsing and independent of standard output streams. This system captures four distinct event categories:

- **Status events:** Checkpoint transitions with CPU timing and context-specific attributes;
- **Warnings:** Conditional alerts including severity levels, source routine identification, and descriptive messages;
- **Errors:** Critical conditions preventing calculation continuation, with routine context and error descriptions;
- **Debug events:** Compilation-time enabled tracing for targeted code regions and nested execution contexts.

3 SIESTA

In D1.1 we provided the first version of the Software Development Plan. Here we update the content related to SIESTA development by taking into account the work already done and the activities well on their way to completion, as well as a series of actions on interoperability and workflows that were not included in D1.1 but were documented in deliverable D2.2 from WP2.

3.1 Task 1 – Node-level Performance

Up to M30 we have continued the monitoring and interface-design related to the ELPA library. A number of enhancements have been introduced: GPU offloading to the Cholesky-decomposition step in the diagonalization workflow, and improvements and bug fixes related to the efficiency of other operations. In particular:

- New wrapping code and heuristics for the computation of the density-matrix from the eigenvectors have been added to our fork of the ELSI library, allowing it to offload properly this operation when using external ELPA libraries. We plan to port

this DM-building piece of ELSI code as an alternative path in the DM-building routines in SIESTA, which currently use only sparse algorithms that are not amenable to GPU streaming.

- In version 2025 of ELPA there have been enhancements to the omnibus routine, which computes the eigenvectors of the generalized eigenvalue problem in one call, keeping data transfer operations between host and device to the minimum. Integration of this routine in our interface could enhance the offloading efficiency in the solver calls.

As discussed in Sect. 7, reduced-precision hardware is likely to feature in upcoming HPC architectures, compromising the efficiency of the now-standard double-precision calculations. During the previous phase of the project, we started to explore the use of single-precision operations in the solver stage of SIESTA, particularly when calling the ELPA library through the ELSI layer. At that point, the strategy involved performing a prescribed number of self-consistency steps in single-precision, ending the loop with double-precision calls to the solver routines to bring the final results to the standard level of precision.

Our plans now are to perform the full self-consistency cycle in single precision, with a final single step in higher precision performed only if needed, with full control by the user and monitoring based on the precision of other operations in the run (i.e. initial stages of relaxation will not need full precision in any case). These code-based developments will be in addition and complementary to any automatic emulation that the libraries might offer, such as transparent use of Ozaki schemes.

3.2 Task 2 – Parallel Efficiency

- The enhancement of TranSiesta's scalability through the parallelization over orbitals and k-points should be completed relatively soon, complementing the existing parallelization over energy points in the complex-plane integration contour.

- An area of intense interest is the interplay of MPI parallelization and GPU usage. This is a matter that depends on architectural details but some general progress is being made by exploring vendor-provided frameworks for performance enhancement. In particular, the CINECA and BSC teams are collaborating to determine the optimal approach for balancing the number of MPI ranks and the number of GPUs per node, utilising the MPS framework and the NCCL support now provided by the ELPA library. Further tests might involve the new Leonardo software stack based on HPCX-MPI to support intra-node aware reductions, which has shown promising results in other codes.

- The lion's share of computing time in SIESTA is taken up by the solver stage, and so also the parallel scalability of the code is basically linked to that of the solver. Our solver of choice for massive scalability for large systems is PEXSI, which offers three intrinsic levels of parallelization: over orbitals, poles, and chemical potential interpolation references, and another potential parallelization over spins. The diagonalization-based solver of choice for medium to large systems is based in the ELSI-ELPA combination, and it also offers several levels of parallelization:

over orbitals, spins, and k-points when needed. A shortcoming of the current offering is that the parallelization must always be used in full, with a number of MPI ranks that is a multiple of the number of k-points and spins, and that could be in some cases unnecessarily large. We are developing a new approach based on a configurable number of teams of MPI ranks, each dealing with a single k-point and spin, and that can proceed sequentially if desired over the list of k-point/spin tasks.

- While not strictly related to efficiency, we mention here our ongoing work on refining the MPI interfaces in the code. We have added as a default the option to use the `mpi_f08` standard interfaces, which offer better argument checking and other enhancements. The legacy MPI interfaces are kept for those compiler/MPI library combinations that still do not support the modern idioms.

3.3 Task 3 – Software Engineering

During RP2, with the benefit of a new solid build system and the implementation of a set of dev-ops best-practices, we have much improved the pace of new releases of SIESTA, bringing into the community version key developments that had been in production but in side branches. While new features keep being introduced and developed in side branches, this release pace is expected to bring them sooner to the main trunk of development, avoiding long-lived branching.

The SIESTA group has developed a complete set of `Spack` recipes and has deployed the code in all EuroHPC machines. Further streamlining of deployment in some centers (a growing list in view of the EuroHPC guidelines) depends on the creation and refinement of EasyBuild scripts, which are tightly integrated with toolchains and can offer built-in reproducibility and streamlined deployment through the European Environment for Scientific Software Installations (EESSI) framework. Here, SIESTA has also achieved important advances, and a production version of SIESTA (5.2.2) is already offered in EESSI, with plans to integrate the more recent 5.4.X version in the next cycle.

Deployment is being done on a per-platform basis at this stage, but collaborations with WP3 and the wider EuroHPC community on CD systems are advancing. Our own CI deployment is experimental at this stage, and it will be further developed and put into production with consideration of the guidelines agreed at the consortium and EuroHPC level.

During previous phases of MaX, SIESTA has undergone a substantial modularization, delegating a number of tasks to libraries which in some cases have been spun off from the code itself. These libraries are in continuous development and refinement to offer better performance and new features. There is ongoing work in `libfdf` to improve the internal data structures to streamline lookup performance for large systems, and in `xmlf90` to improve the memory handling in the DOM subsystem, which has seen little use so far but can feature in a number of developments involving XML files produced by MaX codes. Further, `libgridxc` is being extended to deal with parametrized functionals used in connection with the exact exchange implementation, developed outside MaX but to be integrated in the coming months.

We have made substantial improvements to the documentation, integrating tutorials and other educational materials with online versions of the manuals that are kept up-to-date automatically. Part of the educational materials have been created for the various

schools and workshops organized in collaboration with WP5. This trend of documentation integration will continue until the end of the project, and will include sections on HPC-specific deployment and use.

3.4 Task 4 – New Features and Interoperability

The plans for the final part of the project in this area are to continue working on the interoperability-enhancing features, including IO handling, proper checkpointing, and better API-like execution of the code:

- New IO enhancements have been introduced in RP2, such as improved NetCDF wrapping, reordering of data structures for increased throughput, and more compact storage of hamiltonians for electron-phonon calculations. These address in part the checkpointing task, and also the interface with post-processing utilities and workflows. A number of enhancements have been coordinated also with the developers of the `sisl` package,[1] which is not part of MAX , but is tightly integrated with SIESTA).
- It has become apparent from our work in the deployment of SIESTA on the EuroHPC machines that a performance model is becoming a necessity, almost on par with the proper optimisation for particular architectures. The choice of algorithm and execution parameters is crucial to exploit the very varied characteristics and modes of operation of different machines. We will continue our data gathering and development of recommenders.
- Further interoperability hooks discussed in D1.1 included interfaces to the `ZeroMQ` library and to `HyperQueue` basic primitives, as well as a revamped API. Our interface work is ongoing, and the API improvements are focusing on the MPI-based implementation, with enhancements to the building of client programs and a cleaner handling of the fdf contexts for ensemble calculations.

In addition, after the implementation of the QM/MM subsystem and the enhancements of the TD-DFT module, the plans concerning features for functionality, include:

- The final integration of the implementation of spin-orbit coupling in `TranSiesta` [2].
- Work on the extension of the `Psolver` library integration to support implicit-solvent calculations.

4 FLEUR

4.1 Task 1 – Node-level Performance

First tests have shown that significant parts of the code do not have to utilize double precision arithmetic. For example, diagonalization of the standard eigenvalue problem can be done in single precision without changing the total energy significantly. However, there are also code parts where double precision arithmetic must be used. The reduction

[1] https://zerothi.github.io/sisl/

of the generalized eigenvalue problem to the standard problem is such an example, where the reduction of the precision quickly leads to a drastic loss in precision of final results. Therefore, we plan to study the effects of reduced precision arithmetic more systematically and to refactor the code such that the precision used in different code sections can be adjusted. To complicate matters, we also expect the needed precision to depend on the system (size) and the properties of interest. Hence, a flexible method of selecting the required precision is needed that can also be tailored to the specific scientific use case in the end.

4.2 Task 2 – Parallel Efficiency

Out of the main code section of the FLEUR code, the generation of the charge density still need significant improvements. In particular, two aspects make this code difficult to maintain, extend and optimize:

- The complex treatment of local orbitals: As our basis set comprises LAPW basis functions as well as local orbitals (LOs), the calculations in the muffin-tin spheres around the atoms have to take these different functions into account. So far this has been done rather explicit, i.e. separate code for the LAPW, the LOs as well as LAPW/LO cross-terms are present. By unifying the representation, the code can be significantly reduced and generalized. From our experience gained in other sections of the code, we expect that the small overhead which will occur since not all special properties of these basis functions are considered is most probably not significant. Perhaps clearer data access patterns can even lead to performance gains. In any case, the reduction of complexity of the code will significantly ease future performance tuning and portability of the code.

- The treatment of non-collinear magnetism also lacks generality. As a result of the historical development of features, the spin-off-diagonal contributions to the density matrix are calculated in code parts dedicated explicitly to this task, which formally can be seen as a special case of general charge density generation. By implementing this approach, the code will again be simplified and more robust.

In addition to these goals to create simpler, easier to maintain and better optimizable code, we also foresee that the simpler, cleaner and generalized interface will ease the implementation of calculators for further physical properties.

The refactoring planed in this section will expose new levels of parallelism, enabling better scalability and performance. In particular, the simplified structure and the transformation of specialized code sections into generalized subroutines will enable to achieve better load balancing and the possibility to distribute the load for different atom groups to different sets of MPI processes.

4.3 Task 3 – Software Engineering

The Development of the Density-Functional Perturbation Theory (DFPT) within the FLEUR code has been a significant focus for advancing materials simulations, particularly in the study of phononic properties and dielectric responses. In addition to the refactoring and restructuring already planned in the previous task, a lot of improvements are foreseen for the DFPT functionality in FLEUR:

- Refactoring of Code: The current code will be restructured to improve its maintainability, readability, and efficiency. This will involve reorganizing and simplifying the codebase, making it easier for developers to contribute to and extend the functionality of DFPT in FLEUR.
- Porting to GPU Architecture: While the code already uses existing optimized kernels from FLEUR, further new DFPT related kernels that are performance critical will be identified and ported to GPU architecture. This will significantly accelerate calculations, allowing users to tackle larger and more complex systems.

4.4 Task 4 – New Features and Interoperability

Our future work on interoperability will be mostly targeted around the DFPT functionality, the extensions planned for these and the implementation of Mössbauer parameters.

The initial implementation of DFPT in FLEUR was specifically tailored for phonon calculations. Building on the phonon framework, recent efforts have extended DFPT to handle dielectric response properties. This includes the calculation of static dielectric constants, by incorporating perturbations in the electric field. These extensions enhance FLEUR's capability to model polarization effects and electronic responses in materials under external electric fields, both for bulk materials and more recently also taking the film mode into account. These new features will be equipped with appropriate IO functionality and incorporated into complex workflows. In addition, a generalized interface will be created to facilitate the implementation of additional functionality, such as magnetic response and other types of perturbations. The key goal of this development is to provide future developers with an easily extendable codebase to be able to respond to emerging research needs and to make this functionality accessible and interoperable.

An obvious strength of an all-electron code such as FLEUR is the precise description of the electron charge and magnetization densities and the potential near the atomic nuclei. The calculation of the experimentally relevant Mössbauer parameters relies on such high-quality descriptions. In the past, the main applications of the FLEUR code were in fields unrelated to this kind of physics, so that Mössbauer parameters were never implemented. However, recently, several users asked about the Mössbauer parameters for their planned use of the code. We thus plan to implement the calculation of these parameters to exploit this obvious strength of an all-electron code. This covers the implementation of the calculation of isomer shifts, hyperfine fields, and electric field gradients. Again, we plan to add appropriate IO functionality and to incorporate these features into interoperable workflows.

5 BigDFT

We outline the software implementation plans for the `BigDFT` code in the upcoming phases of the MaX CoE. These plans aim to ensure that the code remains performant, portable, and maintainable on heterogeneous pre-exascale and exascale systems. The reference release for this phase is version `1.9.6` of `BigDFT`, which has now been published. To facilitate wider dissemination and accessibility, we are currently updating the corresponding `conda` recipes and `pip` packages, enabling simplified deployment

of both the core and client components of the suite. The implementation roadmap is organized around four key focus areas corresponding to Tasks 1 through 4.

5.1 Task 1 – Node-level Performance

The node-level performance strategy for the upcoming period focuses on enabling high and portable efficiency on heterogeneous architectures. Building on prior developments using CUDA and OpenCL, recent activities have laid the foundation for extending GPU acceleration through SYCL and are preparing for exploratory efforts with the Kokkos programming model.

- **SYCL-based performance portability via ONEMath.** We have implemented a backend selector written in SYCL that unifies access to various FFT libraries through a consistent API layer, termed `ONEMath`. The supported backends include Intel's `MKL`, NVIDIA's `cuFFT`, AMD's `rocFFT`, and our OpenCL-based `DBFFT` library. Notably, `DBFFT` has been adapted for seamless SYCL usage and proves especially valuable on ARM-based platforms, as it does not require the Intel graphics compiler and remains compatible across a broader ecosystem. This selector has demonstrated functionality on diverse hardware backends including Intel PVC, AMD MI200 series, and NVIDIA A100 devices.

- **Performance portability through Kokkos.** As a next step, we plan to explore the Kokkos programming model to express parallelism and memory movement in a hardware-agnostic fashion. The target for initial Kokkos porting is the Poisson solver, where memory and execution abstractions are already modular. Kokkos' strong support across GPU and CPU architectures makes it a promising candidate for unifying our offload strategy across future HPC systems.

- **Dynamic backend selection and runtime dispatch.** The ONEMath layer is being extended to support runtime dispatch based on hardware capability and user configuration, ensuring optimal backend choice without requiring recompilation. This will be essential for large-scale workflows running on diverse EuroHPC clusters.

- **Benchmarking and bandwidth-based profiling.** To guide optimization efforts, performance will be profiled in terms of node-level bandwidth saturation. Kernel-level instrumentation will be added to identify bottlenecks in memory-bound versus compute-bound phases, enabling fine-tuned optimization of FFT and convolution routines on a per-architecture basis.

- **Mixed precision experimentation.** We plan to experiment with selective mixed-precision computation, especially in FFT-based convolution and preconditioning steps of the solver, to improve throughput without loss of final accuracy. This will also support more efficient usage of GPU memory bandwidth and cache hierarchies.

These activities will ensure that the BigDFT code remains competitive and efficient on emerging HPC architectures while minimizing the overhead for developers and users in managing backend-specific optimizations.

5.2 Task 2 – Parallel Efficiency

Task 2 addresses the planned developments to further enhance the parallel efficiency and scalability of the `BigDFT` code, both in its linear-scaling and cubic-scaling modes, with a specific focus on better exploiting heterogeneous resources in multi-node settings.

- **Generalization of the GPU-accelerated Poisson solver to multi-node environments.** As a follow-up to the successful acceleration of the Poisson solver on single-GPU nodes via SYCL and the ONEMath interface, we plan to generalize this component for distributed-memory GPU workloads. This effort will focus on enabling GPU-GPU communication patterns that allow each MPI task (or group of tasks) to offload the Poisson kernel while maintaining scalability across multiple nodes. The goal is to reduce host-device transfers and fully leverage device-resident execution, including halo exchanges and Fourier-space operations.

- **Asynchronous collectives and fine-grained load balancing.** Building upon the improvements introduced in previous releases, we plan to extend the usage of non-blocking MPI collectives in key stages of the computation, particularly for scalar reduction and data exchange during the self-consistent field (SCF) cycle. This includes further adoption of `MPI_Iallreduce` and non-blocking point-to-point communications where latency hiding can be exploited. Improved load balancing will also be targeted by monitoring computational imbalance and dynamically adapting the domain partitioning in the linear-scaling solver.

- **Taskgroup-based parallelization strategy.** To address the heterogeneous nature of large systems and to maintain high parallel efficiency, we plan to adopt a taskgroup-based parallelization scheme. This approach will allow the subdivision of the global MPI communicator into logical groups, each of which can handle a subset of the computational tasks – for example, assigning a dedicated group to the multi-node Poisson solver. This decoupling is essential to avoid under-utilization for fine-grained tasks and to match the computational granularity to the available hardware resources, espaecially in presence of GPU acceleration.

- **Dynamic mapping of taskgroups and communication balance.** The taskgroup approach introduces new degrees of freedom in the mapping of tasks to nodes. We will develop profiling and benchmarking strategies to guide the optimal size and composition of taskgroups, minimizing communication overheads while maximizing compute utilization. These benchmarks will explore various trade-offs between computational density and inter-group communication, especially for cases where fragment sizes or solver kernels differ substantially.

- **Modular control of concurrency through taskgroup policies.** The new taskgroup model will also guide the organization of concurrency settings – such as OpenMP thread pools, GPU stream allocation, and memory footprint – according to taskgroup roles. This modularization will simplify runtime configuration and allow performance tuning at the level of computational roles, facilitating systematic scalability studies and better integration into complex workflows.

These planned developments aim at ensuring optimal scalability of `BigDFT` across exascale architectures by matching workload characteristics to communication and hardware topology, while preparing the codebase for advanced hybrid workflows.

5.3 Task 3 – Software Engineering

Task 3 encompasses future developments aimed at improving the maintainability, modularity, and portability of the `BigDFT` codebase. These efforts are essential for supporting the upcoming architectural diversification and ensuring the long-term sustainability of the software ecosystem.

- **Unification of buffer management through `f_buffer` abstraction.** A significant line of development involves restructuring how temporary and communication buffers are handled within the `futile` infrastructure library. The goal is to consolidate buffer-related functionalities into a well-defined `f_buffer` object, which abstracts memory allocation, data typing, device-host ownership, and MPI window metadata. This will enable more robust, readable, and error-resilient code in all areas that involve low-level memory access or data exchange, especially across heterogeneous CPU-GPU boundaries.
- **Hourglass interface for backend-agnostic kernel invocation.** In the GPU-enabled Poisson solver module, we plan to generalize the existing backend selection mechanism through a standardized Hourglass interface. This design pattern aims to isolate the computational kernel implementation (bottom of the hourglass) from the calling host code (top of the hourglass) via a minimal and stable intermediate interface. Through SYCL wrappers, this intermediate layer enables the user to switch between backend libraries (such as `cuFFT`, `MKL`, `rocFFT`, or `DBFFT`) without modifying host-level logic. This structure will reduce code duplication and facilitate the deployment of hardware-specific optimizations, while keeping the interface clean and consistent for developers and users alike.
- **Separation of semantics and storage in orbital representations (`liborbs`).** We plan to continue and refactor the `liborbs` module, which is responsible for orbital storage and manipulation. The envisioned structure will separate the mathematical semantics of the orbitals (e.g., basis choice, representation transformations) from the physical data layout and storage. This reorganisation will enable clearer code pathways for variational minimisation, orbital orthogonalisation, or support for localized orbitals. Additionally, it will promote testability and pave the way for further multi-scale extensions of the orbital manipulation pipeline.

Collectively, these software engineering efforts aim to simplify the internal complexity of the code, increase the portability across emerging HPC hardware, and improve the reliability and maintainability of the BigDFT ecosystem in preparation for exascale-readiness.

5.4 Task 4 – New Features and Interoperability

Task 4 focuses on enhancing interoperability between BigDFT and external tools and frameworks, as well as expanding its capabilities for large-scale scientific workflows.

These developments are instrumental for building complex, data-intensive pipelines and enabling the design of exascale-ready lighthouse applications.

- **Development of dynamic, MD-assisted quantum workflows via `remotemanager`.** In recent months, we have demonstrated a first prototype of a hybrid workflow coupling BigDFT (quantum calculations) with the GENESIS molecular dynamics code. These components are orchestrated through the Python-based `remotemanager` library. This framework supports asynchronous, lazy-evaluation workflows that allow new quantum samples to be triggered directly from the trajectory evolution of a long-timescale MD run. Future development will focus on increasing the robustness and generality of this approach, with particular attention to its deployment across different HPC job schedulers and file systems. These developments are a direct continuation of the workflow integration work described in the first-year deliverables (see D1.2 and D2.1).

- **Workflow formalisation for production-grade interoperability.** We will work on turning the current bespoke workflows into formal, reusable and portable entities, expressed either as self-contained Python pipelines or declarative configuration files. This effort aims to standardize BigDFT's role within larger QM/MM or QM+graph analytics applications. It will also allow a more robust interface to simulation platforms that rely on container orchestration (e.g., Apptainer, Singularity) and environment reproducibility.

- **Enhanced I/O layer for interoperability with data-centric platforms.** A further area of planned development concerns the structured representation of BigDFT results for downstream analysis and AI-driven tools. This includes: (i) exporting intermediate and final quantities (e.g., density matrix, Hamiltonian, fragment interaction energies) in standardized formats (e.g., JSON, YAML. MatrixMarket); and (ii) interfacing with Machine learning approaches to understand the reusability of the data.

These activities reflect the strategic direction of BigDFT toward interoperability with multi-code workflows, AI-assisted analytics, and FAIR principles, aligning with the broader vision of the MAX Centre of Excellence.

6 YAMBO

In the following we report the key planned development activities concerning the YAMBO code, presented according to the core Tasks of WP1.

6.1 Task 1 – Node-level Performance

- **Finalization and optimization of OpenACC and OpenMP porting.** The complete development and consolidation of the OpenACC and OpenMP-GPU backends is one of the core goals of YAMBO within MAX . As compared to CUDA-Fortran, which is released as production-ready, the OpenACC and OpenMP-GPU support needs to be extensively tested and profiled for performance (in comparison

with CUDA-Fortran, when possible), and completed in some parts (such as the use of GPU/aware linear algebra, e.g. from HIP Solver, on AMD GPUs).

- **Optimization of the OpenMP parallelism for many-core ARM based architectures.** In parallel to the finalization and optimization of the GPU-porting, we also plan to further optimize the shared-memory OpenMP parallelism in the presence of many-core architectures (notably including, e.g., A64FX and the Rhea chip). The YAMBO mini-app will be used as an optimization vehicle to identify and blue-print possible solutions.

- **Mini-App for Performance and Precision Assessment**. A dedicated mini-app has been developed to isolate the computational kernel of the response function — one of the most time-consuming components of the code, required for a range of calculations including quasiparticle corrections within the GW approximation and optical spectra within the Bethe-Salpeter framework. The primary objective of this mini-app is to restructure the computational workflow to explicitly expose **GEMM-like** operations, facilitating optimization and hardware-specific tuning. In particular, this mini-app will serve as a key tool to evaluate the impact of reduced-precision techniques, such as double-precision emulation, within the Ozaki scheme.

6.2 Task 2 – Parallel Efficiency

- **Extension and optimization of distributed Linear Algebra interfaces** for linear system solvers and diagonalization. The exploitation of distributed linear algebra libraries is key to the parallel performance of some runlevels of YAMBO, such as the BSE solvers or the calculation of response functions in reciprocal space. We plan to further extend the use of distributed linear algebra libraries, especially in the presence of GPU acceleration.

- **Optimization of memory usage and distribution**. Despite the trend in HW development witnessing an increase in the GPU memory, running YAMBO on GPU accelerated machines may reach important memory footprints, posing threats for the completion of calculations in the most severe situations. In view of this, optimizing the distribution of memory in the presence of GPUs is a key aspects that we plan to work on within the scope of MaX-3.

6.3 Task 3 – Software Engineering

- **Modularization enhancement.** Modularization is one of the key strategies embraced by the MAX CoE in order to enhance the long term maintainability of its flagship codes. While important achievements have already been accomplished in the past, the continuous development of the codes, together with the fast evolution of the sources, and a growing set of developers, make modularization a moving target, which needs to be maintained and further optimized.

- **Deployment.** The Spack recipe used for deploying YAMBO will be continuously maintained and updated in line with new releases of both the tool and the YAMBO

code itself. It will soon be integrated into the Spack built-in repository via a pull request. Similarly, EasyBuild recipes have been developed to meet the requirements of the CI/CD working group coordinated by CASTIEL2, and will also be maintained.

- **Automatic Software Documentation.** To improve code maintainability and usability, we plan to develop an automatic documentation workflow by integrating FORD (Fortran Documentation Generator) with Sphinx, a widely adopted documentation system. FORD will be used to extract structured documentation directly from the Fortran source code, while Sphinx will handle the generation of user-friendly, web-based documentation. This combined approach will allow consistency between the source code and its documentation, reduce manual maintenance efforts, and produce high-quality technical resources accessible to both developers and end-users.

- **Continuous Integration: Test Suite Maintenance and Extension.** To support the robustness and reliability of the software, we are extending the Continuous Integration (CI) test suite workflow engine. The test suite workflow is currently managed by a pearl engine. This will be substituted with Python scripting. This effort will simplify automated, reproducible testing across multiple platforms. The ongoing development focuses on improving test coverage, simplifying the integration of new test cases, and enhancing reporting mechanisms to provide rapid feedback to developers.

- **Yambopy test-suite and documentation.** In order to proceed with the official release and publication of the Yambopy code, a complete documentation will be written, both using automatic tools (Sphinx) and including manually prepared tutorials. In addition, the Yambopy test-suite will be created using the `pytest` framework. The Yambo database compatibility tests will be included in the Yambo test-suite.

6.4 Task 4 – New features and Interoperability

- **GW self-consistency.** We have made significant progress in the implementation of the quasi-particle self-consistent GW (QSGW) method in Yambo, following the approach described in Ref. [3]. Initial validation tests revealed that conventional plasmon-pole approximations are insufficient to accurately converge deep-lying electronic states. To address this limitation, we have integrated the QSGW self-consistency module with the recently developed multipole approximation [4], which provides a more reliable dynamical description of the screening. The implementation phase is now complete. Current efforts are focused on exploring and optimizing parallelization strategies to enhance the performance and scalability of the QSGW scheme.

- **Electrostatic embedding of GW and BSE.** Electrostatic embedding within the Polarizable Continuum Model (PCM) framework has been implemented through an interface with the QUANTUM ESPRESSO + ENVIRON library.[2] The new

[2] https://environ.readthedocs.io/en/latest/

functionality is fully parallelized and GPU-enabled. We have successfully validated the implementation by comparing results for small molecules, such as nucleotides, in various solvent environments. Current development efforts focus on extending this scheme to more complex geometries for periodic systems, enabling the inclusion of substrate or encapsulation effects in the calculation of quasiparticle energies and optical spectra of two-dimensional materials.

- **Coupled electron-ion dynamics.** Real time simulations involving a non-adiabatic coupling of electronic and ionic degrees of freedom, as treated within the framework of Ehrenfest dynamics, is one of the long standing goals of recent YAMBO developments. This will be first implemented within the additional hypothesis of small ionic displacements, using the electron-phonon machinery available in YAMBO. Later, this assumption will be removed in favour of a fully-fledged implementation.

- **Convergence accelerators.** Building on the recent development of the W-average technique [5], a successful convergence accelerator for the **k**-point sampling of the GW workflow applied to 2D materials, we have recently developed a blueprint extension of the method to treat metals in 2D and 3D. We plan to implement this new algorithm with YAMBO, as the proper treatment of metallic screening in reduced dimensions is numerically challenging in many-body perturbation theory methods. Given the current interest in 2D metals and semi-metals of the electronic structure scientific community, we expect this approach to have a significant impact in the field.

- **Yambopy new features.** We will include four new main features in the Yambopy code. (i) A revamped and general class to deal with reciprocal space, including high-symmetry pathfinding for plotting k-space dependent functions. This feature will be compatible with both QUANTUM ESPRESSO and YAMBO databases and support any Bravais lattice type. (ii) Full support for analysis and visualization of finite-momentum excitons from Bethe-Salpeter calculations. (iii) Support for calculation, real-space plotting and automatic symmetry analysis (in terms of point group / small group representations and angular momentum) of the exciton wave functions. (iv) Complete set of tools for exciton-phonon coupling calculations, including indirect optical spectra, Raman spectra and exciton lifetimes. (v) As a stretch goal, we plan to include two new reciprocal-space interpolation schemes, based on the tetrahedron method and on the $k \cdot p$ method, to complement the already implemented smooth Fourier method, since the latter is not optimal for nonsymmetric or discontinuous quantities defined in reciprocal space.

7 Targeting new strategic architectures

The computing landscape is undergoing a fundamental transformation that will have a significant impact on high-performance scientific computing. While HPC architectures appeared relatively stable when we first developed our software development plan, we now face rapid changes driven by two major trends.

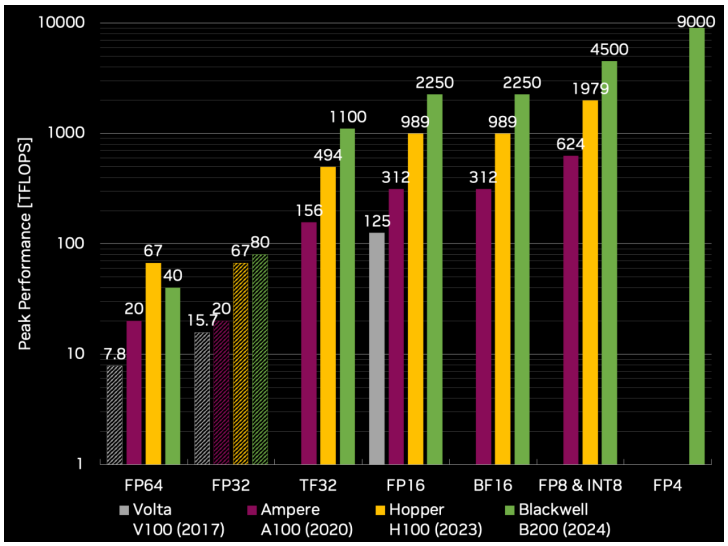

Figure 7: Kindly provided by Massimiliano Fatica, NVIDIA Corp.

First, substantial new investments are reshaping processor development in Europe. The European Chips Act and the DARE initiative aim to establish European sovereignty in microchip technology, leading to a new generation of processors built on the open-source RISC-V instruction set architecture. RISC-V is emerging as a compelling platform for scientific computing, particularly through its scalable vector extensions that provide flexible, wide vector operations crucial for computational workloads.

These vector units offer similar capabilities to traditional SIMD architectures but with enhanced programmability and adaptability. This poses a readiness challenge to MAX flagship codes. It will thus be important for MAX to follow closely the developments and progress of these initiatives. For example, as WP1 we plan to coordinate with the DARE Software technical area and, as soon as possible, support their LLVM-based toolchains and their version of the `flang` compiler. In general, we plan to position ourselves to support these next-generation RISC-V platforms and their advanced vector processing capabilities.

Second, GPU development is increasingly prioritising artificial intelligence applications, also fostered by EuroHPC through initiatives like AI Factories and Gigafactories. This shift has important implications for scientific computing: future GPU architectures from major vendors will likely emphasize single- (or even lower) precision performance while reducing support for double-precision floating-point operations. The historical GPU performance data in Fig. 7, provided by NVIDIA Corp., illustrates this trend clearly. While FP64 performance improved from Volta to Ampere and Hopper architectures, the newer Blackwell generation exhibits reduced double-precision capabilities compared to its predecessor.

These parallel developments — the rise of RISC-V processors and the AI-focused evolution of GPUs — require careful consideration in our software development strategy to ensure continued performance and compatibility across emerging high-performance computing (HPC) platforms. The shift toward AI-optimized GPUs presents both challenges and opportunities for scientific computing. From an energy efficiency standpoint,

traditional double-precision (FP64) and single-precision (FP32) matrix operations using fused multiply-add instructions consume significantly more energy per FLOP than equivalent operations performed through Tensor Core matrix multiplication. The efficiency gap becomes even more pronounced when compared to INT8 operations, which are orders of magnitude more energy-efficient.

This energy advantage, combined with the dominant role of AI workloads in shaping procurement decisions, will likely drive widespread adoption of AI-optimized systems in HPC centers. These systems emphasize single-precision tensor cores over traditional double-precision capabilities, potentially creating substantial performance challenges for scientific applications in materials science, weather modeling, and engineering that rely on high numerical precision.

However, emerging numerical emulation techniques offer a promising solution. Methods based on Ozaki schemes have been for example proposed for performing the FP64 matrix-matrix and matrix-vector operations with the low precision cores of these AI-suited GPU, but also with user-entry ones. A notable example of this using BIGDFT has been recently published [6], and within this report, the QE Sec. 2.1.2 show the preliminary results of a library currently under development by researchers at NVIDIA and other institutions that can preserve the numerical accuracy of double-precision calculations while leveraging the efficiency of single-precision tensor cores. Similar efforts are also planned in different groups.

While early experimentation is essential for encouraging vendor development in this area, the long-term strategy should focus on adopting vendor-provided libraries as they become available. The most valuable contribution that community codes and the MAX Centre of Excellence can make is identifying and documenting the diverse use cases that these libraries need to support.

8 Conclusions

The updates above confirm most of the original plan, including targets, strategies, and methods, with very few significant corrections needed. The main integration concern, as emphasized in Sec. 7, involves adapting all code plans for AI-suited GPUs, which deliver most of their computational power through accelerated operations using lower-precision data types. This adaptation is facilitated by the code's acquired modularity and recent efficiency refactoring, which enables the offloading of code components and allows developers to better understand their code computational characteristics.

Additionally, the supplementary details provided in this update clearly demonstrate how years of adapting and optimizing codes for exascale computing have yielded valuable insights into many critical aspects. For instance, one fundamental lesson concerns the importance of improved communications for parallel efficiency and how dedicated vendor libraries can effectively support this goal. Another common optimization technique, previously presented only in outline, is now illustrated in detail: replacing blocking communications with overlapping computation and communication.

Through years of interaction with the toolchains of numerous EuroHPC clusters, we have also gained important insights into the value of package managers like `Spack` and `EasyBuild`. While this was partially addressed in the original plan, we now provide more detailed support plans, particularly for `EasyBuild`.

A key topic in this update is the acceleration of auxiliary post-processing components and the introduction of newly developed scientific features. This development not only indicates that the main code porting has been largely achieved, but also reflects the growing demand from user communities for comprehensive support of our codes in HPC clusters. While these auxiliary porting efforts and accelerations are crucial for realising our scientific workflows, the programmed efforts must also look beyond this target. We must work toward providing increasingly seamless support for new HPC technologies, making our codes readily available and deployable on these machines, compiled with optimal toolchains, and fine-tuned for performance and efficiency. Through this continued evolution, we ensure that our computational tools remain at the forefront of electronic structure simulations and high-performance computing (HPC), ready to tackle the challenges of tomorrow's most demanding research questions.

References

[1] Gong, X. & Dal Corso, A. An alternative gpu acceleration for a pseudopotential plane-waves density functional theory code with applications to metallic systems. *Computer Physics Communications* **308**, 109439 (2025).

[2] Wittemeier, N., Papior, N., Brandbyge, M., Zanolli, Z. & Ordejón, P. Quantum transport with spin orbit coupling: New developments in transiesta (2025). URL https://arxiv.org/abs/2501.16162. 2501.16162.

[3] van Schilfgaarde, M., Kotani, T. & Faleev, S. Quasiparticle self-consistent gw theory. *Phys. Rev. Lett.* **96**, 226402 (2006).

[4] Leon, D. A. *et al.* Frequency dependence in gw made simple using a multipole approximation. *Phys. Rev. B* **104**, 115157 (2021).

[5] Guandalini, A., D'Amico, P., Ferretti, A. & Varsano, D. Efficient gw calculations in two dimensional materials through a stochastic integration of the screened potential. *npj Comput Mater* **9**, 44 (2023).

[6] Dawson, W., Mohr, S. & et al. J. chem. theory comput. *J. Chem. Theory Comput.* **16** (2020).