D4.2

First report on code profiling and bottleneck identification, structured plan of forward activities

Carlo Cavazzoni, Fabio Affinito, Uliana Alekseeva, Claudia Cardoso, Augustin Degomme, Pietro Delugas, Andrea Ferretti, Alberto Garcia, Luigi Genovese, Paolo Giannozzi, Anton Kozhevnikov, Ivan Marri, Stephan Mohr, and Daniel Wortmann

Due date of deliverable	31/08/2019 (**month 9**)
Actual submission date	31/08/2019
Lead beneficiary	CINECA (participant number 8)
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) / 30/11/2021 (month 36)
Website	http://www.max-centre.eu
Deliverable no.	D4.2
Authors	Carlo Cavazzoni, Fabio Affinito, Uliana Alekseeva, Claudia Cardoso, Augustin Degomme, Pietro Delugas, Andrea Ferretti, Alberto Garcia, Luigi Genovese, Paolo Giannozzi, Anton Kozhevnikov, Ivan Marri, Stephan Mohr, and Daniel Wortmann
To be cited as	C. Cavazzoni et al. (2019): First report on code profiling and bottleneck identification, structured plan of forward activities. Deliverable D4.2 of the H2020 CoE MaX (final version as of 30/08/2019). EC grant agreement no: 824143, CINECA, Bologna, Italy.

Disclaimer

This document's contents are not intended to replace consultation of any applicable legal sources or the necessary advice of a legal expert, where appropriate. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user, therefore, uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors' view.

Contents

1 Executive Summary

This deliverable sets the baseline for the performance and scalability status towards exascale of MAX applications, and at the same time identifies the key bottlenecks that currently preclude MAX codes from efficiently executing a set of selected scientific use cases on future European pre-exascale and exascale systems. It is important to remark that MAX community codes are complex objects that can be configured to run in many different ways, by changing several parameters. Moreover, the required computational resources (time, memory, I/O, etc) connected to different input datasets may span several orders of magnitude, thereby making the codes display very different computational behaviours. Being practically impossible to explore the whole space of possible working conditions and parameters of MAX codes, we have decided to focus our effort on those parameters that are potentially blocking for scientific use cases of interest for future exascale systems. We thus report on code profiling and bottleneck identification activities referring to such use cases. While this does not necessary imply that other scientific use cases may display the same bottlenecks or profiling patterns, the methodology and best practices adopted in this deliverable can be easily extended to all other use cases.

Concerning the profiling, we have decided to consider the simulation wall-time as the main performance metric, since the timing of the whole workflow is what actually impacts the user perception and productivity. We have also decided that profiling and bottleneck identification in particular should be referred as much as possible to well identifiable kernels and modules (e.g.: "the code does not scale primarily because of eigenvectors orthogonalisation"), since this can help both code experts and scientists to review the applications. When a deeper analysis is needed –e.g., instruction or function-level profiling–, we have decided to get in contact with the PoP CoE, since these activities are at the core of their action and expertise. Given the evolution of HPC towards extreme heterogeneity, a relevant advancement of this Deliverable is the systematic inclusion of benchmarks on accelerated architectures.

All above decisions were taken as a result of several discussions during the first months of activity, and finally reviewed in a three-day face-to-face meeting at CINECA (Bologna, IT on July 10-12, 2019) involving people from Work Packages 1, 2, 3, and 4.

In this Deliverable, we report on the profiling and benchmarking campaign performed along the above lines during the first months of MAX on its six flagship codes, QUANTUM ESPRESSO, Yambo, FLEUR, BigDFT, CP2K, and SIESTA. Importantly, we have started to collect benchmark/profiling curated data (including the scientific datasets) in a MAX dedicated repository. On one side, this will allow code developers to automate the collection process (e.g. via AiiDA) and to follow the evolution of the code performance in time. On the other side it will make available the results to the broader community of code users.

2 Introduction

As defined in the description of work, WP4 is responsible for profiling and benchmarking releases of the MAX codes. This is done via a dedicated task, T4.4. The outcome of the activities of this task is functional to many other tasks of other work packages (WP1, WP2, WP3, and WP6) in order to provide feedback on identified code bottlenecks and progresses obtained in terms of performance enhancement with respect of the relevant metric (e.g. scalability, time to solution).

In particular, this report provides the baseline results for the first release of the MAX flagship codes, to be used by all developers across the centre to measure the effectiveness of the solutions adopted to improve code performance. The same information will be used by other WP4 tasks to look for co-design opportunities (e.g. a bottleneck that is linked to a specific hardware feature), or justify a proof-of-concept with a new paradigm or software technology (e.g. if one of the adopted paradigm is found to be responsible for poor performances).

This deliverable is organized as follows: in the first Section we report the selected scientific use cases being used to define the performance and bottleneck baseline for the different codes. The dataset for these use cases are stored in a dedicated MAX Gitlab repository[1] and made available to all MAX developers to be also used in the activities of other WPs. In the second Section we report the results of the many benchmarks we have run along with the identified bottlenecks, and early activities to solve them. In the third Section we discuss on the activities planned to improve the codes towards the exascale (for the given uses cases). Finally, conclusions and lessons learned are reported.

3 Scientific use cases

In this Section we report about the scientific use cases, datasets and profiling goals behind the performance and bottleneck analysis performed in WP4. The cases reported are relevant for future research projects that will run on next generation of European HPC systems. It is therefore of fundamental importance for this community to ensure they will run efficiently and without bottleneck preventing them to run at all.

The datasets are presented in a synthetic way using a template table, where the information available is presented with the same layout regardless of the code. The same layout can be used also in future profiling campaign, making them easy to reproduce and compare.

3.1 List of scientific use cases

The selected scientific use cases are reported as a collection of tables. More detailed information and the actual datasets are stored in the MAX Gitlab repository.

3.1.1 QUANTUM ESPRESSO

For QUANTUM ESPRESSO we identify and collect 4 scientific use cases related to different identified bottlenecks. The first dataset (Table 1) refers to a challenging whole

[1] https://gitlab.com/max-centre/benchmarks

protein DFT electronic structure simulation, meant to test the limits of both QE and current HPC system. Indeed this kind of datasets could represent a valid alternative to LINPACK for system evaluation and assessment, generating enough system load to stress the supercomputer and providing speedup figures on real and relevant scientific use cases. The second dataset (Table 2) has been selected for two reasons: it refers to Car-Parrinello molecular dynamics simulation engine of QE and it is of industrial relevance, since it was setup during an industrial prospect we run with a large European company. The third dataset (Table 3) represents a typical materials science problem, similar to those scientists are running today on high-end HPC system around the world and that is required to run on next generation of supercomputer as well. Finally the fourth case (Table 4) represents a challenging scientific benchmark, already reported in the literature [1], and subject of a comparative study in the past with other codes [2]. These characteristics offer a good opportunity to build a new baseline for the most up-to-date version of QE, to be used for assessing the progress in term of efficiency and bottleneck removal.

Title	Electronic structure of a 5H6V Protein in water
Short description of the scientific use case	Compute ground state property and ab-initio molecular dynamics (Car-Parrinello and Born-Oppenheimer) of a small protein solvated in water. This simulation can be applied to better understand protein-ligand interaction, and improve molecular docking techniques applied in drug design.
Target code	cp.x and pw.x of QE 6.4.1
Description of the input dataset	Atomic position of 5H6V protein plus solvation waters. In total the system contain 7534 atoms of 5 species: N, O, H, C and S. Pseudopotentials are ultrasoft in the Vanderbilt form, the exchange and correlation is PBE. For the DFT problem we use an energy cut-off of 25Ry. It computes 12252 electronic states. It uses 16369 Kleinmann-Bylander projectors.
Target of the profiling campaign / bottleneck	Code scalability and I/O
Target Supercomputer	Marconi (KNL partition)
Target computational resources	Simulation from 128 and 1024 compute nodes (8704 cores and 69632 cores), each simulation lasting approx 3 hours
Link to the dataset (if available)	https://gitlab.com/max-centre/benchmarks/tree/master/Quantum_Espresso/PW/5H6V https://gitlab.com/max-centre/benchmarks/tree/master/Quantum_Espresso/CP/5H6V
Link to the code source	https://gitlab.com/QEF/q-e/-/tags/qe-6.4.1

Table 1: First scientific case for QUANTUM ESPRESSO: Electronic structure of 5H6V protein in water

Title	ZrO2 supercell
Short description of the scientific use case	Compute ground state property and ab-initio molecular dynamics of a ZrO2 supercell under pressure. This dataset is part of an ongoing industrial prospect, and details cannot be disclosed.
Target code	cp.x and pw.x of QE 6.4.1
Description of the input dataset	Atomic position of a ZrO2 supercell. In total the system contain 792 atoms of 2 species: Zr and O. Pseudopotentials are ultrasoft in the Vanderbilt form, the exchange and correlation is PBE. For the DFT problem we use an energy cut-off of 30Ry.
Target of the profiling campaign / bottleneck	Code scalability and load balance
Target Supercomputer	Marconi (KNL partition)
Target computational resources	Simulation from 8 and 256 compute nodes (544cores and 17408cores), each simulation lasting approx. 30minutes.
Link to the dataset (if available)	Industrial prospect, dataset are not public
Link to the code source	https://gitlab.com/QEF/q-e/-/tags/qe-6.4.1

Table 2: Second scientific case for QUANTUM ESPRESSO: ZrO2 supercell

Title	Graphene-Co-Ir
Short description of the scientific use case	Graphene grown on non-commensurate transition metal substrates forms a moire pattern that can be used as templates for molecular adsorption and partial molecule-substrate decoupling at the electronic level, while maintaining the magnetic coupling. We will compute the magnetic ground state of the Graphene/Co/Ir slab. For a recent scientific publication on the subject see Ref. [3].
Target code	QE, pw.x
Description of the input dataset	Atomic position of a 10x10/9x9/9x9 supercell, with a total of 605 atoms with spin polarization (6 k-points, 2668 *2 KS states, 7.7 M G vectors, 256x256x270 FFT grid). LDA exchange and correlation potential. Pseudopotentials are norm conserving with an energy cut-off of 75 Ry.
Target of the profiling campaign / bottleneck	Code scalability (MPI, OpenMP, w/ and w/o GPU support), and I/O
Target Supercomputer	Marconi KNL, Piz-Daint
Target computational resources	Marconi: 144 nodes, using 64 to 72 (hyper-threading) cores per node (with omp_num_threads ranging from 1 to 4 at least). A full scf calculation is expected to last about 5 hours. Piz-Daint: A number of nodes similar to the KNL case, mostly looking at strong scaling
Link to the dataset (if available)	https://gitlab.com/max-centre/benchmarks/tree/master/Quantum_Espresso/PW/GrCoIr
Link to the code source	qe-6.4.1 from: https://gitlab.com/QEF/q-e/-/tags/qe-6.4.1 qe-gpu-6.4.1a1 from: https://gitlab.com/QEF/q-e-gpu/-/releases

Table 3: Third scientific case for QUANTUM ESPRESSO: Graphene-Co-Ir interfaces

Title	Carbon Nanotube CNT10POR8
Short description of the scientific use case	Compute ground state property and ab-initio molecular dynamics (Car-Parrinello and Born-Oppenheimer) of porphyrin functionalised carbon nanotube.
Target code	cp.x and pw.x of QE 6.4.1
Description of the input dataset	Atomic position of CNT10POR8 system. In total the system contains 1532 atoms of 4 species: N, O, H, C. Pseudopotentials are ultrasoft in the Vanderbilt form, the exchange and correlation is PBE. For the DFT problem we use an energy cut-off of 25Ry.
Target of the profiling campaign / bottleneck	Code scalability and I/O
Target Supercomputer	Marconi (KNL partition)
Target computational resources	Simulation from 16 and 1024 compute nodes (1088 cores and 69632 cores), each simulation lasting in approx 3 hours
Link to the dataset (if available)	https://gitlab.com/max-centre/benchmarks/tree/master/Quantum_Espresso/PW/CNT10POR8 https://gitlab.com/max-centre/benchmarks/tree/master/Quantum_Espresso/CP/CNT10POR8
Link to the code source	https://gitlab.com/QEF/q-e/-/tags/qe-6.4.1

Table 4: Fourth scientific case for QUANTUM ESPRESSO: Carbon Nanotube CNT10POR8

3.1.2 Yambo

For Yambo we have identified and collected two scientific use cases related to different bottlenecks. More in details, scalability (both MPI and OpenMP) and profiling tests have been performed considering two systems of physical interest with different computational characteristics. The first use case that we have considered is a defective TiO_2 bulk supercell, having a cell of 72+1 atoms, 8 **k**-points, and a medium range number of plane waves (about 5×10^5 to represent the density). Wavefunctions have been further truncated and represented using 40000 plane waves, while up to 2000 states are included in the sum-over-states. We have used this system to study both the MPI and OpenMP scaling of the code. The second use case that we have considered is a much larger low-dimensional system, a 1D chevron-like Graphene nanoribbon. The system has 136 atoms and still 8 **k**-points, but sits in a much larger supercell, described by means of 3.5 million plane waves (for the density). The sum-over states are performed considering 800 bands (each represented by 4.5×10^5 plane waves). More details of the systems studied and the results targeted are described below.

Title	Defected TiO_2 structures
Short description of the scientific use case	Defects in TiO_2 have a key role in the determination of its electrical conductivity and optical properties. We will perform a GW calculation as implemented in Yambo of a defective TiO_2 bulk supercell. This class of systems (defected TiO_2) is relevant for photocatalytic applications and is currently the subject of a large amount of research. A recent publication on the subject can be found e.g. in Ref. [4].
Target code	Yambo (GW, BSE calculations)
Description of the input dataset	H-defected TiO_2 2x2x3 supercell, with 72 +1 (H) atoms (577 electrons, treated non-spin-polarised). The system is described using 8 **k**-points and to 2000 bands in the sum-over-states. The polarisability cutoff is set to 6 Ry (1317 **G**-vectors).
Target of the profiling campaign / bottleneck	Overall total time-to-solution, strong scaling (and bottleneck spotting), memory handling.
Target Supercomputer	Marconi KNL
Target computational resources	Marconi: from 40 to 320 KNL nodes
Link to the dataset (if available)	https://gitlab.com/max-centre/benchmarks/tree/master/Yambo
Link to the code source	Yambo 4.5 devel version, rev:16810 hash:29d3c00a8 (hosted in a private repo)

Table 5: First scientific case for Yambo: Defected TiO_2 structures

Title	Graphene Nanoribbons
Short description of the scientific use case	One dimensional percursor polymer for graphene (GR) nanoribbon (GNR) with chevron-like shape. Relevant for optical properties (optoelectronic devices), it is a quite large nanostructure made by C and H with al arge number of plane waves involved in the simulations using both QE and Yambo. A recent publication on the electronic and optical properties of this system is in Ref. [5].
Target code	Yambo (GW calculations)
Description of the input dataset	The system contains 136 atoms (C and H) and sits in a large unit cell of 32×66×40 Bohr^3, leading to 3.500.000 (450.000) **G**-vectors for the density (wavefunctions). The system is described using 8 **k**-points and 800 bands in the sum-over states. A cutoff of 3 Ry is used to describe the polarisability (leading to 7423 **G**-vectors).
Target of the profiling campaign / bottleneck	Overall total time-to-solution, strong scaling (and bottleneck spotting), memory handling. Marconi KNL
Target Supercomputer	Marconi KNL
Target computational resources	Marconi: 128 / 256 / 512 / 768 KNL nodes
	https://gitlab.com/max-centre/benchmarks/tree/master/Yambo
Link to the code source	Yambo 4.5 devel version, rev:16810 hash:29d3c00a8 (hosted in a private repo)

Table 6: Second scientific case for Yambo: Graphene Nanoribbons

3.1.3 FLEUR

For FLEUR two use cases have been set up that can be used to investigate different bottlenecks. Both focus on demonstrating the scalability of the main parts of the code, in particular of the Hamiltonian setup and the MPI communication. While the first system is a simple TiO_2 insulator with point defects with two setups of 1078 and 2156 atoms respectively, the second system is a single setup of a 3750 atoms $SrTiO_3$. Despite its larger number of atoms, the strong asymmetry of the second system will require the use of more k-points and hence add the aspect of scalability in the combined k-point and eigenvalue MPI parallelization. It is also planed to use it to investigate possible bottlenecks in some property calculators, e.g. to see conducting defect states.

Title	TiO2 structure with defects
Short description of the scientific use case	Ground state properties
Target code	FLEUR
Description of the input dataset	Two test cases: with 1078 and 2156 atoms.
Target of the profiling campaign / bottleneck	Scalability, Hamiltonian setup, MPI communication
Target Supercomputer	CLAIX2016, CLAIX2018, Hazel Hen, SuperMUC-NG
Target computational resources	CLAIX2016 and Hazel Hen: up to 256 nodes; CLAIX2018 and SupeMUC-NG: up to 512 nodes
Link to the dataset (if available)	1078 atoms: https://gitlab.com/max-centre/benchmarks/tree/master/FLEUR/fleur_big_TiO2_conv 2156 atoms: https://gitlab.com/max-centre/benchmarks/tree/master/FLEUR/fleur_huge_TiO2_conv
Link to the code source	https://www.flapw.de/site/downloads/

Table 7: First scientific case for FLEUR: TiO2 structure with defects

Title	SrTiO3 structure with dislocations
Short description of the scientific use case	Ground state properties, electrical conductivity
Target code	FLEUR
Description of the input dataset	Atom positions of 3750 atoms.
Target of the profiling campaign / bottleneck	Scalability, Hamiltonian setup, MPI communication
Target Supercomputer	CLAIX 2018, Hazel Hen, SuperMUC-NG
Target computational resources	CLAIX 2018: 512 nodes for 1 k-point; Hazel Hen: 1024 nodes for 1 k-point, 4096 nodes for 4 k-points; SuperMUC-NG: 512 nodes for 1 k-point, 2048 nodes for 4 k-points
Link to the dataset (if available)	https://gitlab.com/max-centre/benchmarks/tree/master/FLEUR/fleur_huge_SrTiO3
Link to the code source	https://www.flapw.de/site/downloads/

Table 8: Second scientific case for FLEUR: SrTiO3 structure with dislocations

3.1.4 BigDFT

The physical test system is Uranium dioxide, which, being the most widely used nuclear fuel, is both a technologically important and scientifically interesting material, where hybrid functional DFT calculations could play a more important role if they become less expensive. Aside from these points, our motivation in choosing this system was primarily the fact that it is challenging element to simulate, as it contains a large number of electrons even when using a pseudopotential approach. We used an uranium pseudopotential with 14 electrons and an oxygen pseudopotential with 6 electrons. Furthermore it requires a spin polarized treatment, with non identical spin up and down orbitals. We used 5 different periodic cells covering a wide range of sizes, namely a small cell (12 atoms), a medium sized cell (96 atoms), large cells (324 and 768 atoms) and a very large cell containing 1,029 atoms. Our largest system contained 17,150 Kohn-Sham orbitals.

Title	UO2
Short description of the scientific use case	Ground state properties of a challenging metal oxyde with hybrid functionals
Target code	BigDFT
Description of the input dataset	Uranium dioxide, several cell sizes, large number of electrons. Inputs ranging from 164 to 17150 Kohn-Sham orbitals
Target of the profiling campaign / bottleneck	Scalability of the CPU and GPU implementation for PBE and PBE0 functionals, overlapping between MPI communication and computation. Computational cost evaluation of a hybrid functional.
Target Supercomputer	Piz Daint/Marconi
Target computational resources	Marconi: 400 - 800 KNL nodes; Piz Daint: Up to 1600 nodes (with/without GPU)
Link to the dataset (if available)	https://gitlab.com/max-centre/benchmarks/tree/master/BigDFT/UO2
Link to the code source	http://bigdft.org

Table 9: First scientific case for BigDFT: UO2

3.1.5 CP2K

The aim of the profiling performed on CP2K is to isolate and benchmark performance-critical parts of the code relevant for the scientific cases (Tab. 10, 11). For the scientific case (Tab. 12) the task is to create a base line plane-wave simulation with the SIRIUS back end (already fully GPU accelerated) and track the performance improvements for the duration of the project.

Title	RPA calculations on large condensed phase systems.
Short description of the scientific use case	RPA presents an import post-DFT approach to electronic structure, and CP2K has unique capabilities to compute the RPA energy (and related properties) for large condensed phase systems, featuring both $O(N^4)$ and $O(N^3)$ algorithms. The former is faster for 3D systems up to a few hundred atoms, and currently used in most applications. This $O(N^4)$ algorithm is to target of this benchmark.
Target code	CP2K, RPA functionality
Description of the input dataset	The dataset consists of 128 water molecules in a cubic box, representative of liquid water. The water molecules are described with a high quality basis (correlation consistent triple-zeta basis, 53 basis functions per molecule).
Target of the profiling campaign / bottleneck	These calculations are dominated by tensor contractions, that are implemented as dense linear algebra operations, mostly PDGEMM. In the selected benchmark, this routine dominates the calculation, accounting for 90% on 128 nodes of Piz Daint. The aim of this work is to integrate a highly performing, communication optimal PDGEMM implementation (COSMA) that is available for both GPU and CPU architectures.
Target Supercomputer	Current result will be obtained on Piz Daint, future results will be obtained for the EuroHPC architectures, as well as internationally leading supercomputers in the US.
Target computational resources	This kind of calculation can be scaled to large number of nodes, we aim at 64 – 1024 nodes.
Link to the dataset (if available)	https://github.com/cp2k/cp2k/tree/master/tests/QS/benchmark_mp2_rpa
Link to the code source	https://github.com/cp2k/cp2k

Table 10: First scientific case for CP2K: RPA calculations

Title	Linear scaling DFT calculations for very large systems.
Short description of the scientific use case	CP2K features the capability to perform accurate DFT calculations on systems containing thousands of atoms, using a linear scaling implementation of DFT. This benchmark is an established test case to measure the performance of the underlying software infrastructure. This infrastructure is largely given by a sparse matrix library (DBCSR), which is is particularly optimized to perform sparse matrix - matrix multiplications in parallel for a variety of architectures, including both CPUs and GPUs. Furthermore, the use of this library extends beyond linear scaling DFT, as it finds application in low order correlated wavefunction theory (e.g. $O(N^3)$ RPA).
Target code	CP2K and its DBCSR library, in particular the architecture specific backends (e.g. libcusmm).
Description of the input dataset	The input data consists of a large sample of liquid water, namely 20736 atoms ($\approx$ 7000 molecules). Different atom centered basis sets will be employed.
Target of the profiling campaign / bottleneck	Sparse matrix multiplication, and in particular small matrix multiplication: DBCSR and libcusmm.
Target Supercomputer	see above
Target computational resources	Full simulation runs on 256 nodes. Main target is single node kernel performance (e.g. CUDA, AMD, etc.)
Link to the dataset (if available)	https://github.com/cp2k/cp2k/tree/master/tests/QS/benchmark_DM_LS
Link to the code source	https://github.com/cp2k/cp2k https://github.com/cp2k/dbcsr

Table 11: Second scientific case for CP2K: Linear scaling DFT

Title	Plane Waves DFT calculations
Short description of the scientific use case	Ground state calculation of carbon materials (buckyball) using plane-wave basis set.
Target code	CP2K + SIRIUS
Description of the input dataset	A single molecule of C_{60} in a box. This test covers the cases of large FFT grids and small number of atoms.
Target of the profiling campaign / bottleneck	SIRIUS plane-wave DFT implementation, in particular the MaX-developed SpFFT library.
Target Supercomputer	Piz Daint and future AMD-based supercomputers
Target computational resources	4-64 hybrid nodes of Piz Daint
Link to the dataset (if available)	https://github.com/cp2k/cp2k/tree/matster/tests/QS/benchmark_SIRIUS
Link to the code source	https://github.com/cp2k/ https://github.com/electronic-structure/SIRIUS

Table 12: Third scientific case for CP2K: plane wave DFT calculations

3.1.6 SIESTA

We have chosen as a use case the simulation of a large system of biological interest (DNA), which at the same time provides a rather complete stress-test of the code. Besides the sheer number of atoms and orbitals (up to the order of 100k), which is a relevant figure for the solver stage, the appearance of several atomic species (five: C, H, O, N, P) increases the complexity and thus the relative weight of the Hamiltonian setup phase.

Title	Electronic structure of a section of DNA
Short description of the scientific use case	This simulation can be used to determine the charge profile of DNA and other properties relevant to understand its dynamics.
Target code	Siesta 4.2-rc0 (development version)
Description of the input dataset	A short section of DNA, with around 715 atoms, that can be replicated to obtain larger target systems for scalability test purposes. The largest system foreseen contains around 11400 atoms.
Target of the profiling campaign / bottleneck	Code scalability and I/O, in the initialization, setup of the Hamiltonian, and solver phases.
Target Supercomputer	MareNostrum IV (BSC)
Target computational resources	From 96 to 2304 cores.
Link to the dataset (if available)	https://gitlab.com/max-centre/benchmarks/tree/master/Siesta/BSC-DNA
Link to the code source	http://launchpad.net/siesta (trunk version)

Table 13: First scientific case for Siesta: Electronic structure of a section of DNA

4 Profiling results, bottlenecks, and early actions

4.1 QUANTUM ESPRESSO

4.1.1 Profiling on pw.x

The application pw.x contained in the QUANTUM ESPRESSO suite performs total energy and force calculations with plane waves and pseudopotentials or using the PAW formalism. The methodology is based on a *self-consistent* loop, consisting of the following main computational tasks:

- The iterative diagonalisation of the trial Hamiltonian (H) computing the lowest N_B eigenvalues and eigenfunctions (ψ). N_B must be at least as large as the number of electronic states which are needed to build the self-consistent density, though some post-processing or spectroscopic applications may require to further increase this number. N_B depends upon the number and type of atoms contained in the simulation cell. For periodic systems the Hamiltonian can be split into independent blocks, one per $\mathbf{k}$-point (Bloch vector), on which, at each step, the iterative diagonalisation can be performed concurrently by a distinct MPI group. We will refer to this as *pool* parallelism. The same pool parallelism is applied if the system is spin-polarised, adding a net factor of two to the maximum number of usable pools;
- A sum of the contributions of each wave function to the total density. This sum runs through all occupied bands and $\mathbf{k}$-points;
- A mixing of the input density with the newly computed density;
- The computation of the new Hamiltonian operator.

If not adequately implemented, each of these steps may become a bottleneck for very large systems. The most expensive operation is typically the calculation of $H\psi$ products used in the iterative diagonalisation, involving both *local* and *nonlocal* potentials, respectively computed in the *real-space* and *reciprocal-space* grids. The number N_{proj} of non-local operators (*projectors*) depends upon the number and type of atoms. In order to apply the local and non-local potentials and to compute the charge density, the wave functions need to be transferred from the real-space to the reciprocal-space representation, and vice versa. This is done using parallel 3D FFT transforms. The data distributed on 3D grids represent also the major memory requirement of the program and thus the distribution of FFT 3D grids among MPI tasks is used to distribute the memory as well as the computational workload.

Performance profiling [6] done during the operation of MAX phase 1 has shown that for small- and medium-size calculations (up to a few hundreds atoms in the simulation cell, up to a few thousands for both N_B and N_{proj}) the main bottleneck is constituted by the parallel FFT performance. These findings are, for the FFT part only, confirmed by the results for the use case in Tab. 4 as shown in Fig. 1. We defer the discussion of the linear-algebra term (_diaghg) to the large-size case. Parallel FFT satisfactorily reduces the total wall-time as long as the number of MPI tasks is smaller than or comparable to the linear dimensions of 3D FFT grids. When the number of MPI tasks exceeds this number, the scaling becomes poorer. Actions on this side include:

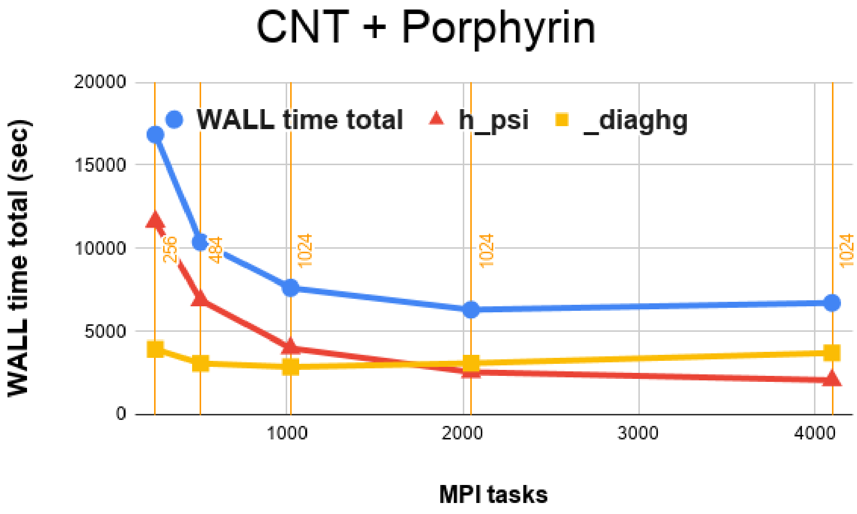


Figure 1: Scaling on use case presented in Tab. 4. Total WALL time and `h_psi` part scale with FFT up to 2048 cores: beyond that, scaling becomes non-optimal. Exact diagonalisation (`_diaghg`) with parallel linear algebra (ScaLAPACK) scales up to a 16×16 BLACS grid (256 MPI tasks). The number of tasks used for parallel linear algebra are indicated by the yellow labels. 4 OpenMP threads per MPI task are used for all runs.

- Improvements on the granularity of the real and reciprocal distribution over MPI tasks, i.e., make the distribution more flexible so that grids can be distributed on more tasks;
- Extension of threaded parallelism in order to exploit more efficiently as many processors as possible in shared memory tasks, as soon as MPI FFT scaling starts to saturate and in case the memory node capabilities limit the number of MPI tasks per node.

We also expect that next accelerated architectures will allow the code to contain the FFT grids of small and medium size systems in one node and to offload FFT and reciprocal space operations directly to the accelerators. With these improvements and the adaptation to accelerators (see next Section), we are confident that the code will be able to exploit efficiently the advancements which will gradually be brought in by the advent of pre-exascale machines.

In large-size computations, where the 3D grids largely exceeds the capacities of a single node, the FFT part scales satisfactorily, but new computational issues are introduced by the growing number N_a of atoms, as the largest of our scientific test cases (Tab. 1) demonstrates. For such case, N_a is already $\mathcal{O}(10^4)$, but with the advent of exascale machines it is foreseeable that users will try to simulate even larger systems.

The timing, scaling, and partition of the computational wall-time for our largest test case are shown in Fig. 2. Although the number of MPI tasks used is already beyond the optimal scalability range for FFT and parallel matrix multiplication, we still have a significant scaling of the `h_psi` sections (540 sec for the 512 MPI × 16 OpenMP vs.

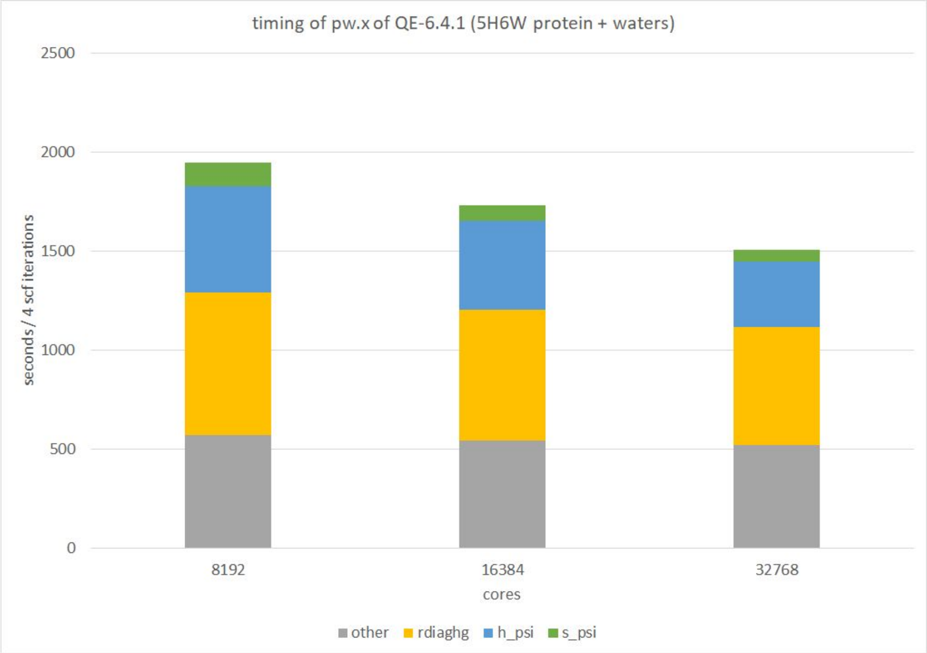


Figure 2: Scalability of pw.x for protein 5H6V case. The parts in yellow (rdiaghg) contain library calls to ScaLAPACK/ELPA exact diagonalisation libraries. The parts in blue (h_psi) contain FFTs and parallel matrix multiplications. The parts in green (s_psi) contain parallel matrix multiplications only.

333 sec for the 2048 MPI × 16 OpenMP). These times are however a small percentage of the total time and improvement on this side will bring only a minor improvement to the overall scalability. Significant portions of the time is instead spent inside exact diagonalisation driver routine (rdiaghg sections of Figs. 2 and 3), which takes 1350 sec with 2048 MPI tasks and 1790 sec with 512 MPI tasks. As anticipated above, this overwhelming cost and poor scaling are caused by the unprecedented number of bands used by the computation.

The most efficient iterative diagonalisation algorithm currently present in the pw.x code is *block Davidson*, which requires repeated exact diagonalisations of a $N_D \times N_D$ matrix with N_D at least two times the number of bands N_B. These exact diagonalisations are performed by the library LAXLib, that uses ScaLAPACK and ELPA to achieve parallelisation. When however N_B exceeds 12000 as in our case, these exact diagonalisations become the dominant term, scaling poorly even if efficient parallel linear-algebra libraries are used. In our calculation, over 25% of the time is spent in performing the parallel exact diagonalisation.

One relevant issue with unmonitored parts for the code has emerged during these profiling sessions: a rather large portion of compute time, tagged as *other* in Fig. 2, could not be assigned to any of the main computational kernels. Profiling tools turned out to be useless: none worked properly, presumably due to the size of the run. It emerged from our analysis that the origin of such bottleneck lies in real-space Ewald sums that were not parallelised at all. The time spent in such part is usually negligible in DFT ab-initio computations (unlike in classical MD) but increases quadratically with N_a and becomes

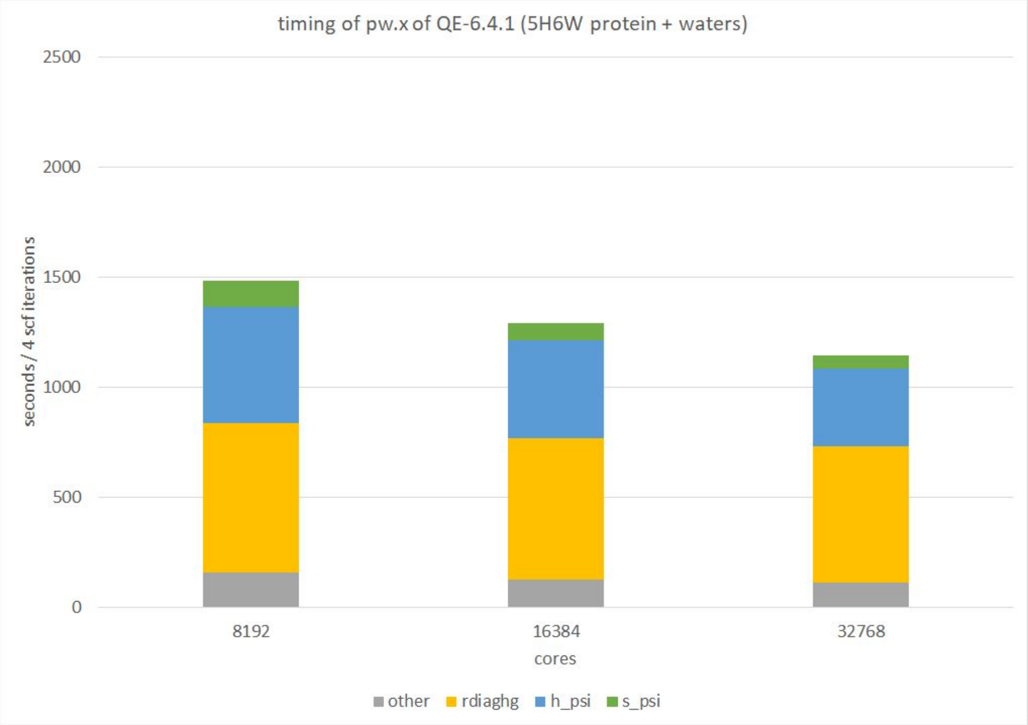


Figure 3: Scalability of pw.x for protein 5H6V case, after the removal of the Ewald sum bottleneck.

large as N_a approaches 10K. As an early activity on the code optimisation, we were able to solve this bottleneck by parallelising the Ewald sums over atoms. This already improves the efficiency of the code at this scale for the benefit of the scientists willing to use it in their next HPC projects (e.g. PRACE or EuroHPC). The improved scalability, with a much smaller *other* section, is reported in Fig. 3.

The calculations of $H\psi$ products (labeled in Fig. 2 as `h_psi`) can also be parallelised on bands, introducing MPI *band groups*, but currently only one of these band groups is used for exact diagonalisation. This forces the code to collect the results from other groups and reduces the number of MPI tasks usable in exact diagonalisation, significantly reducing the advantages (see Tab. 14) of band parallelisation for large-size systems.

tasks	band-groups	h_psi	rdiaghg	s_psi	other	total	calbec
1024	1	854	663	79	–	1596	210
1024	2	561	746	75	186	1569	161
2048	1	668	593	57	–	1450	205
2048	2	485	673	55	205	1418	134

Table 14: Iterative diagonalisation times (sec) for different band parallelism settings for the 5H6V case: The timing of `h_psi` scales down, but the overall scaling is limited by parallel linear algebra (`rdiaghg`) and communication time (included in the *other* section).

The profiling with very large systems has also shown that I/O may become an issue for computations of such size.

- Currently pseudopotentials and restart information (an XML file) are independently read by each MPI task; with thousands of MPI tasks this may cause a significant slow down of reading operations.
- Together with the charge density, the program saves on exit the reciprocal space components of the wave functions. In test cases with more than a thousand wave functions, the termination phase takes a significant time.

A final possible source of bad scaling for giant calculations is the large number of projectors. The usage of projectors involves operating with matrices with dimensions $N_{proj} \times N_B$. The cost of building these matrices is usually negligible and so far not monitored by clocks, but may become sizable for large-size calculation. The memory footprint of these matrices may also become a problem.

4.1.2 Profiling of `pw.x` on GPUs

Since version 6.4.1 of QUANTUM ESPRESSO, `pw.x` can be used on systems equipped with GPUs. As the compilation for GPUs requires a specific setup and linking to architecture specific libraries, the code is distributed GPU-ready via a dedicated repository together with specific routines. Users willing to compile and use `pw.x` on GPU-enabled architectures may find the code at https://gitlab.com/QEF/q-e-gpu .

The most computation-intensive kernels executed with GPU acceleration are the parallel linear algebra and, at variable levels depending upon the system size, the FFT routines. If the memory of a node is sufficient to store all 3D grid data, 3D FFTs are performed directly with GPU-specific libraries using devices on one node. When instead it is necessary to distribute 3D data among MPI tasks, the GPU acceleration is used to perform local 1D and 2D FFT operations and data are then scattered via MPI in order to perform the net 3D FFT operation. In the most common setup of GPU devices, this requires frequent data transfer between device and host memory. In order to improve the computational efficiency, local FFT operations and MPI data scattering are thus done concurrently on different batches of wave functions; while a batch is processed by the GPU, other batches are scattered via non-blocking MPI.

The benchmarking of `pw.x` on GPUs has been done running the scientific test case described in Tab. 3. The size of this test case is such to require the 3D distribution on at least 60 MPI tasks per pool. Parallel linear algebra is performed using GPU-specific libraries on a single GPU. The number of bands (here over 2600) is in fact close to the limit that fits into the device memory of current Piz Daint nodes. For this reason, it is necessary to use the Davidson algorithm limiting the leading dimension of the Hamiltonian matrix representation in the iterative space to its minimal value $N_D = 2 \times N_B \simeq 5000$. We have run the calculations from scratch to full solution. This requires 48 successive iterative diagonalisation loops. The Hamiltonian is divided into 12 irreducible blocks for which iterative diagonalisation may be run independently. The different parallel setting are summarised in Tab. 15.

As can be seen in Fig. 4, the time-to-solution scales down increasing the number of pools. The pool scalability, in principle linear, is impaired because the Hamiltonian

MPI tasks	Pools	tasks per pool	time to solution (secs)
72	1	72	26774
144	1	144	25656
144	2	72	17363
240	4	60	10349

Table 15: Different setups used to run the scientific use case on Tab. 3 on the cluster PiZ-Daint@CSCS with GPUs.

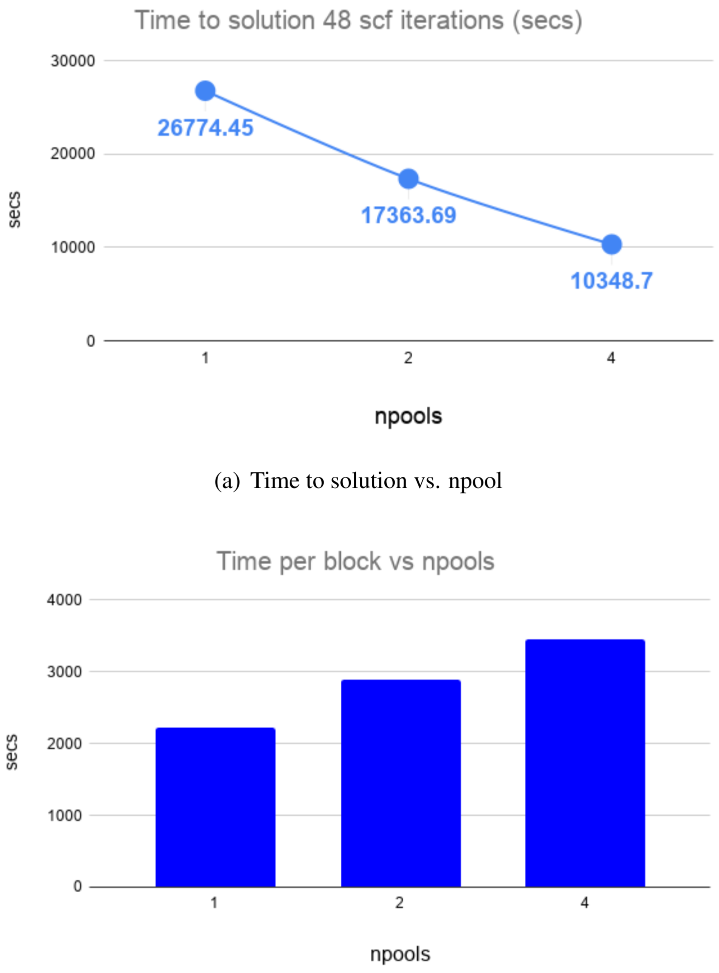
blocks require a widely variable number of iterations. In this scenario, distributing the blocks among pools levels the average time per block of each pool up to the most expensive one. When the number of pools is equal to the number of blocks, all pools are constrained by the slowest block.

As we have a moderately large number of bands, the iterative space of the Davidson eigensolver is over 5000×5000, and the exact diagonalisations are thus computationally important. This part of the computation, labeled as `_diaghg` in Fig. 5, is performed using specific routines for accelerated linear algebra on GPUs. For this reason, this part is almost independent on the number of MPI tasks per pool. Overall the exact diagonalisation time slightly increases with the number of MPI tasks because of the increased communication costs needed to build the representation of the Hamiltonian in the iterative space.

The other part of the iterative diagonalisation time, labeled as `h_psi` in Fig. 5 , uses instead MPI parallelism on 3D data on real and reciprocal space performing 3D FFT operations or matrix vector products as those computed in `vloc_psi` and `calbec`. The behaviour of these routines with respect to the number of MPI tasks per pool is presented in Fig. 6. While `calbec` shows some scaling as the number of tasks per pool increases, `vloc_psi` (whose task includes two 3D FFT operations) is completely saturated at a value around 72 tasks per pool.

4.1.3 Profiling on `cp.x`

The application `cp.x` contained in the QUANTUM ESPRESSO suite performs the canonical Car-Parrinello ab-initio molecular dynamic simulation, sharing most of the modules, subroutines and parallelisation strategy with `pw.x`. In particular, considering the most time consuming kernels, the FFT kernel is the same and is used in the same way in both codes. This kernel was the object of a heavy refactoring in the previous phase of the MAX project, and its scalability was improved significantly for both codes, especially for large atomic systems running on many thousands of nodes. In fact from the performance analysis on both codes presented in this chapter, the FFT was not the main source of performance problems for large-size systems. Instead, it is evident that linear algebra constitutes now the most critical kernel for both codes. Here `cp.x` and `pw.x` share the same low level linear algebra drivers, but differ in the way the eigenstates are computed. In `pw.x` they are obtained by iterative diagonalisation of the Hamiltonian matrix (see Section above), while in `cp.x` they are first obtained by minimising the Car-Parrinello Lagrangian, then propagated adiabatically together with the ions. While they are propagated they are kept orthogonal by applying an orthogonality constraint on the electronic



(a) Time to solution vs. npool

(b) Time to solution per block vs npool

Figure 4: Average time per pool evaluated with different pool distributions.

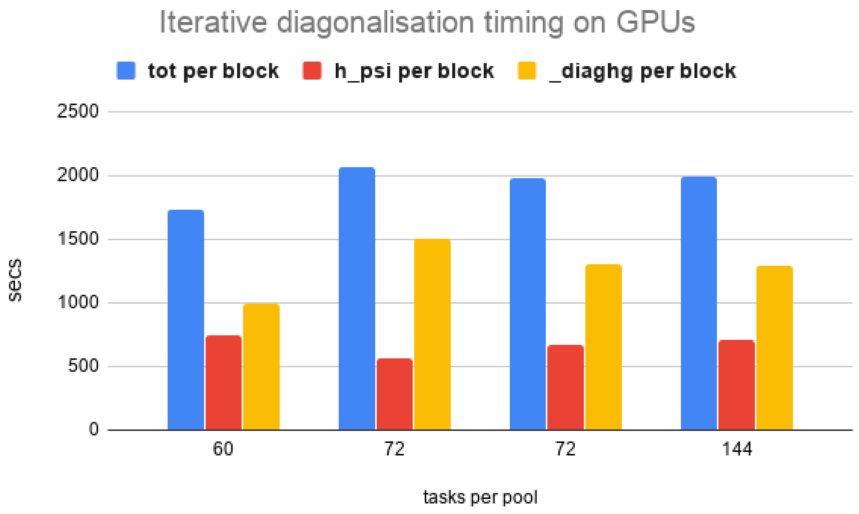


Figure 5: The iterative diagonalisation times for the different adopted setups. The time per block is decomposed in `h_psi` and `_diaghg`, this latter is the exact diagonalisation in the iterative space. The 2 different setups at 72 tasks per pool correspond respectively to rows 1 and 3 of Tab. 15, the difference between the two is mostly due to the deviations from linearity of pools parallelism.

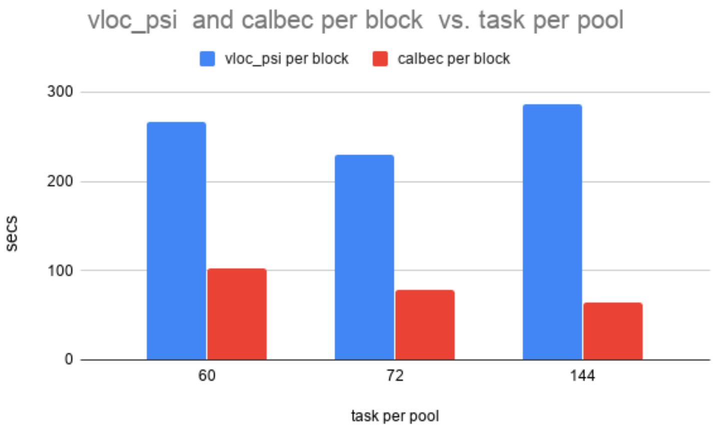


Figure 6: Total times per block of general MPI parallel routines `vloc_psi` and `calbec`, the value at 72 tasks per pool is obtained as average of setups 1 and 3 in Tab. 15.

degree of freedom. This operation is performed in the *ortho* kernel, which implements an iterative constraint refinement, by applying an exact diagonalisation and a sequence of matrix multiplications. on matrices of $N_B \times N_B$ size, to solve the constraint equation.

For the scientific use case ZrO_2, used in a prospect for a service required by an industrial user, we performed several runs for different combinations of parallelisation parameters. The results are shown in Fig. 7. There we reported the total loop time (main_loop) and the time taken by the *ortho* kernel. While for a relatively small number of cores (i.e. 512) the *ortho* time is almost negligible, at 16000 cores the time taken by the *ortho* kernel represents more than 50% of the total loop time, and its scalability profile is flat (i.e. it does not scale) from 512 to 16000 core. Analysing the code, we traced back the cause to a sub-optimal parallelisation of matrix multiplications, with doubly distributed matrices: over the linear algebra *ortho* group, and over the band group. In order to accelerate this scientific use case and make the code more efficient for large core counts (e.g. a system partition of roughly 1 PFlop/s peak performance), we need to improve the *ortho* kernel as the first priority.

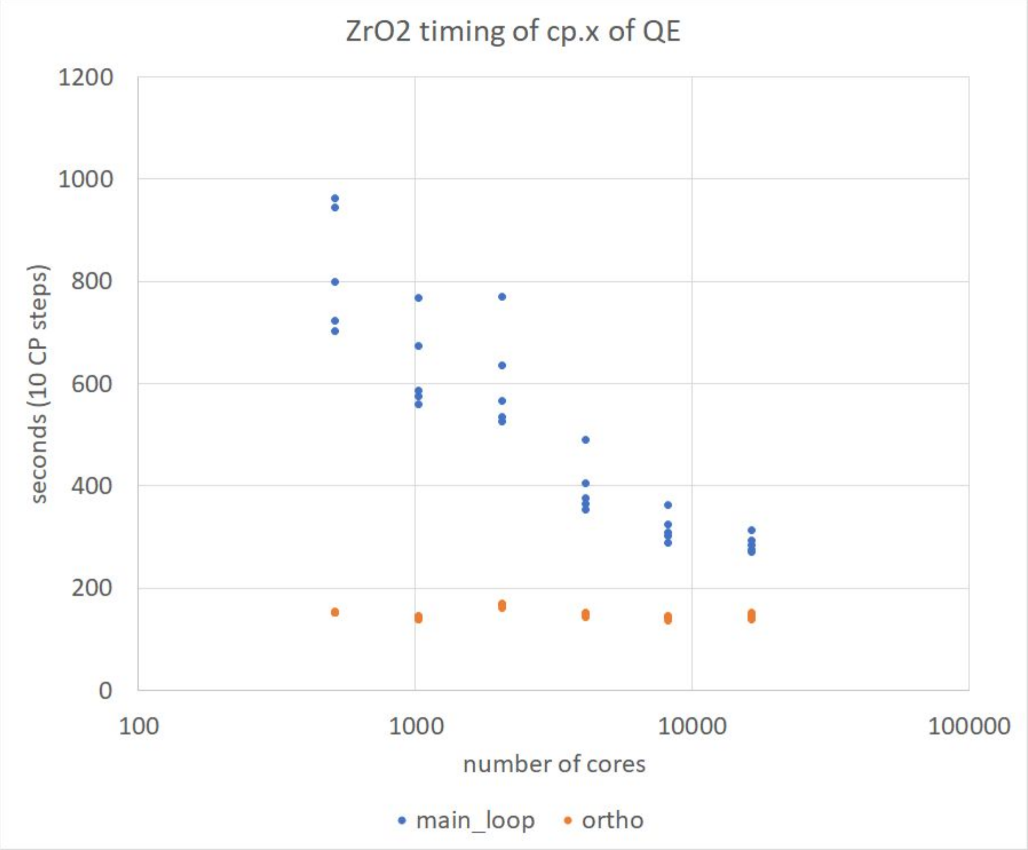


Figure 7: Scalability of `cp.x` for ZrO_2 case. For each number of cores, we tested different numbers of task group, from 1 to 16. The best number of task group for each number of cores give the best performance and scalability for the code. We also plotted the profile of the orthogonalisation kernel, which is almost flat. This is not an issue with small number of cores, since its time is much less than the rest of the code, but prevent the code to scale above 10000 cores.

4.2 Yambo

4.2.1 Profiling on Yambo: the GW workflow

Yambo is an ab-initio code for calculating quasiparticle corrections and optical properties within the framework of many-body perturbation theory. Yambo relies on the Kohn-Sham (KS) wavefuntions generated by the DFT pw.x code of the QUANTUM ESPRESSO package through the interface *p2y* (PWscf to Yambo interface). Quasiparticle (QP) energies are obtained within the GW approximation for the self-energy. The calculation of QP corrections is made by the following main tasks:

(i) Calculation of dipole matrix elements. The dipole matrix element between the states $|n\mathbf{k}\rangle$ and $|m\mathbf{k}\rangle$ (typically one occupied and one empty) is defined as $\mathbf{r}_{nm\mathbf{k}} = \langle n\mathbf{k} \mid \mathbf{r} \mid m\mathbf{k}\rangle$, and is computed within periodic boundary conditions using the relation $[\mathbf{r}, H] = \mathbf{p} + [\mathbf{r}, V_{nl}]$, leading to:

$$\mathbf{r}_{nm\mathbf{k}} = \frac{\langle n\mathbf{k} \mid \mathbf{p} + [\mathbf{r}, V_{nl}] \mid m\mathbf{k}\rangle}{\epsilon_{n\mathbf{k}} - \epsilon_{m\mathbf{k}}} \tag{1}$$

where $\epsilon_{n\mathbf{k}}$ are KS energies. Computationally, the calculation of dipoles involves a reduction over space degrees of freedom ($\mathbf{G}$, here) for every $\mathbf{k}$-point and nm pairs (usually $N_\mathbf{k} \times N_\text{occ} \times N_\text{empty}$ reductions to compute). Both wavefunctions and β projectors are needed, making the kernel both I/O and memory intense. Dipole matrix elements can be evaluated using several approaches (i.e. *G-space v*, *shifted grids*, *Covariant*, and *R-space x* approach, see Ref. [7] for more details). Profiling and benchmarking tests have been performed using the *G-space v* procedure.

(ii) Calculation of density response function in the independent particle approximation, that is:

$$\begin{aligned}\chi^0_{\mathbf{G}\mathbf{G}'}(\mathbf{q},\omega) \quad &= \quad \frac{f_s}{N_\mathbf{k}\Omega} \sum_{nm\mathbf{k}} \rho_{nm\mathbf{k}}(\mathbf{q},\mathbf{G}) \rho^\star_{nm\mathbf{k}}(\mathbf{q},\mathbf{G}') \\ &\times \Big[\frac{f_{m\mathbf{k}}(1 - f_{n\mathbf{k}-\mathbf{q}})}{\omega - (\epsilon_{m\mathbf{k}} - \epsilon_{n\mathbf{k}-\mathbf{q}}) - i\eta} - \frac{f_{m\mathbf{k}}(1 - f_{n\mathbf{k}-\mathbf{q}})}{\omega - (\epsilon_{n\mathbf{k}-\mathbf{q}} - \epsilon_{m\mathbf{k}}) + i\eta} \Big]\end{aligned} \tag{2}$$

where $f_{m\mathbf{k}}$ is the occupation function and

$$\rho_{nm\mathbf{k}}(\mathbf{q},\mathbf{G}) = \langle n\mathbf{k}| e^{i(\mathbf{q}+\mathbf{G})\cdot\hat{\mathbf{r}}} |m\mathbf{k} - \mathbf{q}\rangle, \tag{3}$$

represents one of the core quantities computed by the Yambo code. The calculation of $\rho_{nm\mathbf{k}}(\mathbf{q},\mathbf{G})$ need to have a large number of wavefunctions in memory and involves FFTs. Importantly, for each $\mathbf{q}$ and ω, χ^0 is a dense matrix in the $\mathbf{G}, \mathbf{G}'$ indexes (running up to several thousands), making the kernel very memory intense.

Next, the reducible response function χ is calculated within the random phase approximation as:

$$\chi_{\mathbf{G}\mathbf{G}'}(\mathbf{q},\omega) = \sum_{\mathbf{G}''} \Big[I - \chi^0(\mathbf{q},\omega) v(\mathbf{q}) \Big]^{-1}_{\mathbf{G},\mathbf{G}''} \chi^0_{\mathbf{G}''\mathbf{G}'}(\mathbf{q},\omega) \tag{4}$$

Here v is the Fourier transform of the bare Coulomb potential. The solution of this equation is equivalent to a dense linear system and can be handled with parallel linear algebra techniques.

(iii) Calculation of the electronic self-energy $\Sigma_{n\mathbf{k}}$. It is composed of the exchange (x) and correlation (c) parts, $\Sigma_{n\mathbf{k}}(\omega) = \Sigma^x_{n\mathbf{k}} + \Sigma^c_{n\mathbf{k}}(\omega)$. The exchange term is simply the Fock potential of the Hartree-Fock method and assumes the form:

$$\Sigma^x_{n\mathbf{k}} = -\sum_m \int_{BZ} \frac{d\mathbf{q}}{(2\pi)^3} \sum_{\mathbf{G}} v(\mathbf{q}+\mathbf{G}) \mid \rho_{mm\mathbf{k}}(\mathbf{q},\mathbf{G}) \mid^2 f_{m(\mathbf{k}-\mathbf{q})} \tag{5}$$

while the correlation part of the self-energy is given by:

$$\begin{aligned}\Sigma^c_{n\mathbf{k}}(\omega) &= i\sum_m \int_{BZ} \frac{d\mathbf{q}}{(2\pi)^3} \sum_{\mathbf{G},\mathbf{G}'} \frac{4\pi}{\mid \mathbf{q}+\mathbf{G} \mid^2} \rho_{nm\mathbf{k}}(\mathbf{q},\mathbf{G}) \rho^\star_{nm\mathbf{k}}(\mathbf{q},\mathbf{G}') \\ &\times \int d\omega' G^0_{m\mathbf{k}-\mathbf{q}}(\omega-\omega') \epsilon^{-1}_{\mathbf{G}\mathbf{G}'}(\mathbf{q},\omega') \end{aligned} \tag{6}$$

where $\epsilon^{-1}_{\mathbf{G}\mathbf{G}'}(\mathbf{q},\omega) = \delta_{\mathbf{G}\mathbf{G}'} + v(\mathbf{q}+\mathbf{G})\chi_{\mathbf{G}\mathbf{G}'}$ and G^0 is the non-interacting Green's function [8]. In order to avoid the inversion of large matrices for many frequencies, yambo adopts the so-called plasmon-pole approximation (PPA) for the GW self-energy. Within PPA, the $\epsilon^{-1}_{\mathbf{G}\mathbf{G}'}$ function is approximated with a single pole function and the frequency integral in Eq. (6) done analytically. When implemented, the evaluation of the GW-PPA self-energy is quite CPU intensive.

4.2.2 Defective TiO$_2$ structure: MPI and OpenMP scaling

As a first step, we have considered a defective $2 \times 2 \times 3$ TiO$_2$ bulk supercell with an interstitial H impurity. The system counts 577 electrons. DFT calculations have been performed using pw.x and adopting norm-conserving pseudopotentials with a kinetic energy cutoff (for wavefunctions) of 80 Ry. The electronic structure data generated by pw.x have been processed by p2y for conversion to the Yambo internal format. QP corrections have been finally calculated with Yambo at the GW-PPA level. Tests have been performed to verify the efficiency of both MPI and OpenMP parallelisation schemes. Results are reported below.

MPI scaling. These tests aim at the assessment of the Yambo scaling with respect to the number of MPI tasks. The number of tasks was defined by changing the number of KNL nodes from 40 to 320, using 32 tasks-per-node. This corresponds to a number of MPI tasks that ranges from 1280 to 10240. In all calculations we imposed 2 OpenMP threads per task, in order to take full advantage of the KNL computational resources. The results are shown in Fig. 8 and Tab. 16.

As can be seen from the table, the scaling of the main computational kernels (Dipoles, χ^0, Σ^x, Σ^c) is quite good, with parallel efficiencies typically larger than 75% at 240 KNL nodes, with peaks at 90% or more for χ^0 and Σ^c. At the same time, the overall wall-time does not show the same behaviour. In fact, the total wall-time decreases from 5433 sec (using 1280 MPI tasks) to 1987 sec (10240 MPI tasks). The most significant decrease happens for the runs up to 5120 MPI tasks. For larger number of MPI tasks the total wall-time is almost constant. This happens despite the consistent decrease of the time taken to perform each of the main kernels of the codes with the number of MPI tasks, up to 10240, as can be seen in Fig. 8 and Tab. 16.

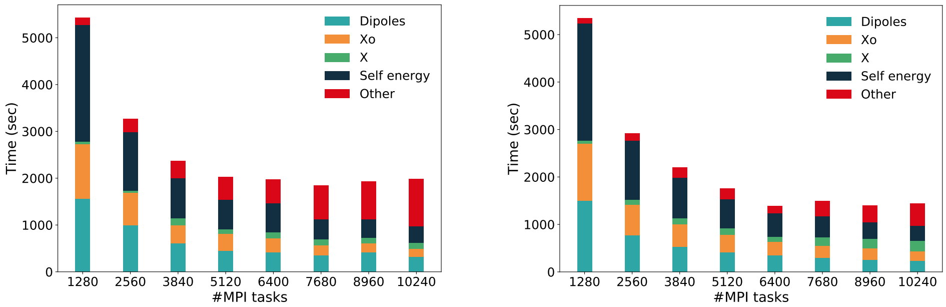


Figure 8: Defected rutile: Execution times for the tests performed with the `yambo` code to verify the efficiency of the MPI parallelisation. Left panel: timing using a pristine version of the code; Right panel: timing after bottlenecks have been addressed. The time for each of the main tasks of the code is given separately. The total time taken to perform other tasks is labeled as "Other". Timing data are explicitly reported in Tab. 16.

For larger number of MPI tasks, the time taken by other tasks than the calculation of dipoles, χ^0 and Σ, represented in red in Fig. 8 and labeled as "Others", dominates the performance of the code. This is the main bottleneck identified by means of this use-case. The "Other" time is mainly due to the initialisation step, before any actual calculation, connected e.g. to the parallel launch of the job as well as the setup of MPI communicators and IO. In particular, after a deeper investigation, we have identified a bottleneck related to job launching (20-30 seconds at 200 KNL nodes, 6400 MPI tasks), perhaps connected to the use of the SLURM scheduler, and a more relevant problem connected to the creation of folders by `Yambo` (of the order of 5-6 minutes). We have re-written the I/O initialisation of `Yambo` and were able to solve the latter issue. The updated data are given in the right panel of Fig. 8.

The MPI parallelism can be distributed in different ways, exploring the parallelism on **k** or **q** points, valence or conduction bands (v,c), or **g** vectors. The distribution is governed by the parameters `X_CPU` and `X_ROLES="q.k.c.v.g"` for the polarisability, `DIP_CPU` and `DIP_ROLES="k.c.v"` for dipoles and `SE_CPU` and `SE_ROLEs="q.qp.b"` for the self energy. The use of different number of tasks leads to changes in the MPI distribution of the different levels of parallelism. Depending on the system, some choices may be significantly better in optimising the performance of the code than others. This leads to small fluctuations in the reported time for each run. A possible improvement could be achieved by the implementation of an algorithm to determine, for a given total number of MPI tasks, and a given system (meaning a given number of bands, **k** and **q** points, **g** vectors), the optimal parallel distribution of MPI tasks for each run-level.

OpenMP scaling. In `Yambo`, the OpenMP parallelism of χ^0 (Dipoles) is governed by the input variable `X_Threads` (`DIP_Threads`). By default these variables are set to zero and controlled by the `OMP_NUM_THREADS` environment variable. The OpenMP parallelism for the self-energy is governed by the variable `SE_Threads`. In order to test the OpenMP implementation of `yambo`, we have concurrently increased the `X_Threads`, `DIP_Threads` and `SE_Threads` variables from 1 to 40. Tests have been performed by

Pristine version

#MPI	#threads	dipoles	χ^0	χ	Σ^x	Σ^c	wall-time
1280	2	1568	1159	53	30	2464	5433
2560	2	997	691	49	15	1232	3268
3840	2	606	385	148	16	848	2369
5120	2	451	361	100	8	619	2029
6400	2	414	301	133	12	606	1979
7680	2	349	221	120	7	427	1846
8960	2	414	196	123	7	381	1928
10240	2	322	165	134	6	344	1987

Fixed version

#MPI	#threads	dipoles	χ^0	χ	Σ^x	Σ^c	wall-time
1280	2	1499	1199	69	30	2441	5347
2560	2	766	651	109	15	1231	2920
3840	2	532	471	133	12	834	2200
5120	2	412	371	133	8	608	1762
6400	2	344	294	99	7	495	1393
7680	2	294	260	180	7	427	1495
8960	2	256	243	196	7	345	1400
10240	2	231	201	225	6	306	1438

Table 16: Defected rutile: Tests have been performed using 32 MPI tasks per node and 2 threads. Times are given in seconds. In all considered cases, I/O χ and I/O WF operations require only a few seconds, and therefore are not reported in the table.

fixing the number of MPI task to 480 (60 nodes, 8 MPI tasks per node). The distribution of the MPI tasks on the different levels of parallelisms has been kept constant during the test, that is we have always imposed in the input the multilevel MPI structure:

```
X_CPU="1.1.30.8.2"
X_ROLES="q.k.c.v.g"
DIP_CPU="1.60.8"
DIP_ROLES="k.c.v"
SE_CPU="1.4.120"
SE_ROLEs="q.qp.b"
```

for the polarisability, dipoles, and self energy, respectively.

The results obtained are reported in Fig. 17 and in Tab. 17, where two horizontal dashed lines divide the plot in three parts. In particular, the first dashed line separates the no-hyperthreading regime (number of threads smaller than 8) from the hyperthreading regime (number of threads greater than 8), while the second dashed line identifies the transition between a regime where the wall-time decreases with increasing the number of threads and a regime where the wall-time increases with increasing the number of threads (note that the KNL technology behind Marconi A2 allows for the use of 4 threads per core, for a total of 272 virtual cores). By fully exploiting the OpenMP parallelism, we can reduce the wall-time of about a factor five. In particular, we observe that the dipoles, χ^0, and Σ^c routines show an excellent OpenMP scalability. At the contrary we recorded a non-monotonic trend in the time associated with the calculation of the

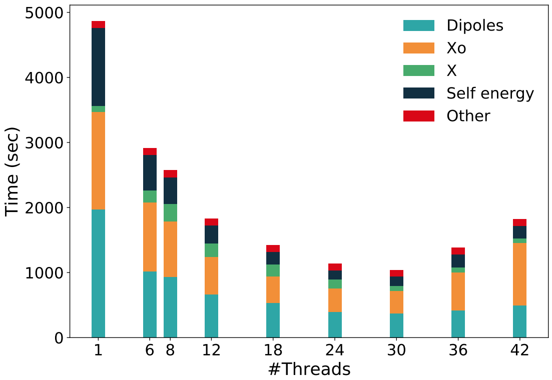


Figure 9: Defected rutile: Execution times for the tests performed with the yambo code to verify the efficiency of the OpenMP parallelisation. The time for each of the main tasks of the code is given separately. The total time taken to perform other tasks is labeled as "Other".

reducible polarisability χ; this behaviour will be carefully analysed in the near future.

4.2.3 Chevron-like polymer: MPI scaling

The computational study of this use case has been mainly focused on the use of large KNL partitions for a single GW run. While performing the tests, we have set the number of MPI tasks per node equal to 8 and the number of OpenMP threads equal to 8, while increasing the number of used KNL nodes (Marconi A2 @ CINECA) from 128 to 768, for a total of 1024 to 6144 MPI tasks (8192 to 49152 cores). The results of the timing footprint of the tests are reported in Fig. 10 (top panel). Two main bottlenecks can be identified from the results of these runs. As for the defected TiO_2 structure, the "Other part", mostly related to the initialisation process, becomes massive (up to 26 minutes of initialisation for the largest partition). The fixed version of the code significantly improves on this issue, as visible in the lower panel of Fig. 10.

More importantly, this use case also evidences a second, more tricky, bottleneck, related to the parallel redistribution of the χ matrix after the solution of the Dyson equation (cast as a dense linear system) has been computed. This is shown as a green contribution to the columns in the left panel of Fig. 10. While the problem can be easily worked around when the χ matrix is not split across $\mathbf{G}, \mathbf{G}'$ indexes, the problem is less trivial when such distribution is switched on. In passing by we also remark that this issue is tightly related to the implementation and usage of parallel IO via HDF5 (already present in yambo). The definition and validation of a general purpose solution is ongoing and requires more investigation.

4.2.4 Intra-node profiling on Yambo: GPUs

Recently, an extensive activity to port Yambo on GPUs has been put in place. This has addressed the kernels computing dipoles, Hartree-Fock, linear response, GW and Bethe Salpeter equation (BSE). Technically, the porting has been achieved by taking advantage

#MPI	#threads/ MPI	#threads/ node	dipoles	χ^0	χ	Σ^x	Σ^c	wall time
480	1	8	1974	1493	92	12	1191	4869
480	6	48	1019	1062	184	5	542	2913
488	8	64	935	849	271	5	407	2576
480	12	96	665	576	209	6	273	1832
480	18	144	532	407	187	7	186	1422
480	24	192	400	356	134	5	141	1140
480	30	240	370	353	70	4	144	1039
480	36	288	419	586	77	4	197	1384
480	42	366	491	968	66	9	182	1819

Table 17: Defected rutile: Tests have been performed using 60 nodes and 8 MPI tasks per node. Times are given in seconds. For all the considered cases, I/O χ and I/O WF operations require only few second, and therefore are not reported in the table.

of CUDA Fortran and exploiting both cuf-kernels and CUDA libraries (cublas, cufft, and cusolver). Currently Yambo counts about 62 CUDA Fortran cuf kernels. Benchmarks have been performed on PizDaint@CSCS (nodes equipped with NVIDIA Tesla P100 GPUs) and Galileo@CINECA (nodes with NVIDIA Tesla V100 GPUs). The system considered is a poly-acetylene chain, i.e. an organic polymer with the repeating unit $(C_2H_2)_n$. Simulations have been performed by adopting a version of Yambo compiled using the PGI compiler for both the CPU and GPU cases. A snapshot of the profiling tool applied to the GPU-ported version of Yambo is shown in Fig. 11.

Results obtained on **Piz Daint** (XC50 partition) are reported in Fig. 12. Here, we compared the execution time for the irreducible polarisability χ^0, the Hartree-Fock, and the self-energy routines for calculations performed on both CPUs and GPUs (from 2 to 8 nodes). The obtained results point out a 5 to 10× speedup in time to solution for the ported kernels. Noticeably, the use of CUDA Fortran for GPU porting has a rather small impact on the code because the accelerated parts are well-localised in only few subroutines. Remarkably, the profiling performed to optimise the performance on GPUs allowed us to obtain also some optimization when running on CPUs, as for instance the one concerning the routine FFT_setup, where a few minutes, non-scaling, time has been reduced to a few seconds.

Data concerning the intra-node profiling of Yambo on Galileo[2] are reported in Tab. 18. Here results are divided in two sets concerning GW (upper part of the table) and BSE (lower part) simulations. Also in this case the results evidence a significant speedup on the time-to-solution, for both GW and BSE.

Typically, for GPU-accelerated systems the recommended usage model for our software is to use one MPI task per card, setting the number of OpenMP threads to the maximum available on the host in order to best exploit also its computational capabilities. Since currently available GPUs (say NVIDIA Volta) have a quite large computational power per card, with a few MPI tasks one is already able to address and exploit a very

[2] Note that Galileo, on a single node, mounts 2 NVIDIA V100 GPUs cards (testing purposes). Results of Tab. 18 compare the performance obtained on the full node (2*8-core Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz (Haswell)) with the data obtained also exploiting the two above-mentioned V100 cards.

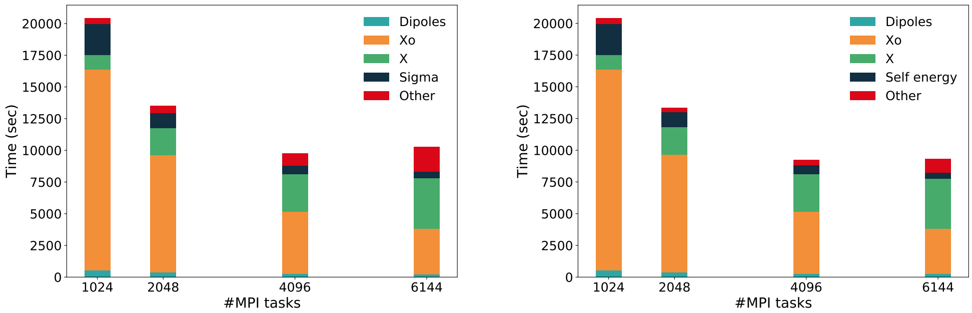


Figure 10: Chevron polymer: Execution times for the tests performed with the `yambo` code to verify the efficiency of the MPI parallelisation. Left panel: timing using a pristine version of the code; Right panel: timing after some of the bottlenecks have been addressed. The time for each of the main tasks of the code is given separately. The total time taken to perform other tasks is labeled as "Other". Note that the data corresponding to 1024 MPI tasks are the same in the two cases since we do not expect major changes introduced by the fix.

	GW					
	$\mathrm{FFT}_{\mathrm{setup}}$	$\mathrm{Xo}_{\mathrm{TOT}}$	$\mathrm{X}_{\mathrm{TOT}}$	HF	Σ	Wall-Time
CPU	69.00	389.00	3.00	52.00	222.00	765.00
GPU	0.21	19.44	2.50	0.63	5.47	38.00

	BSE					
	$\mathrm{FFT}_{\mathrm{setup}}$	$\mathrm{Xo}_{\mathrm{TOT}}$	$\mathrm{X}_{\mathrm{TOT}}$	$\mathrm{BSE}_{\mathrm{kernel}}$	$\mathrm{BSE}_{\mathrm{diago}}$	Wall-Time
CPU	2.24	376.30	2.73	67.75	1.22	477
GPU	0.25	17.22	2.37	11.69	0.74	39

Table 18: Full node vs two NVIDIA V100 GPUs on Galileo. Times are given in seconds. In the upper part we report the times for the FFT setup, the irreducible and reducible polarisability, the Hartree-Fock, the self-energy and the wall-time for both CPU and GPUs. Similarly, the FFT setup, the irreducible and reducible polarisability, the kernel BSE, the BSE diagonalisation and the wall time are considered for BSE calculations.

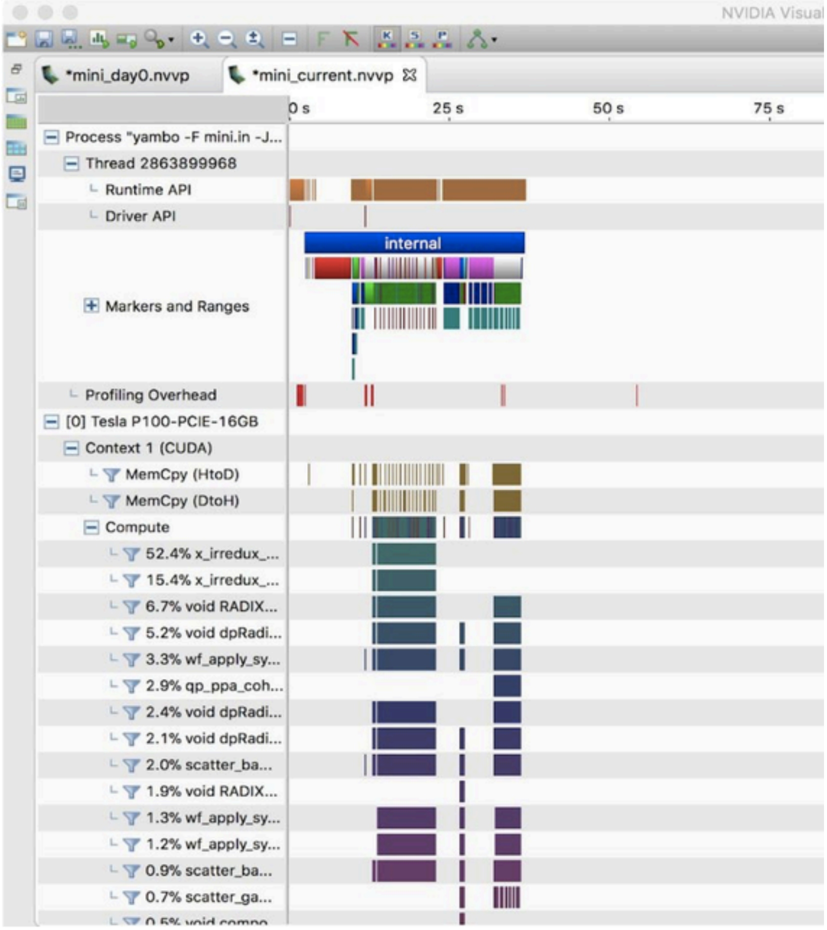


Figure 11: Profiling of the GPU porting of `Yambo` performed on Piz Daint (XC50 partition) using NVIDIA tools. Currently the code counts about 62 cuf (CUDA Fortran) kernels, no custom kernels, plus the use of cufft, cublas, and cusolver libraries.

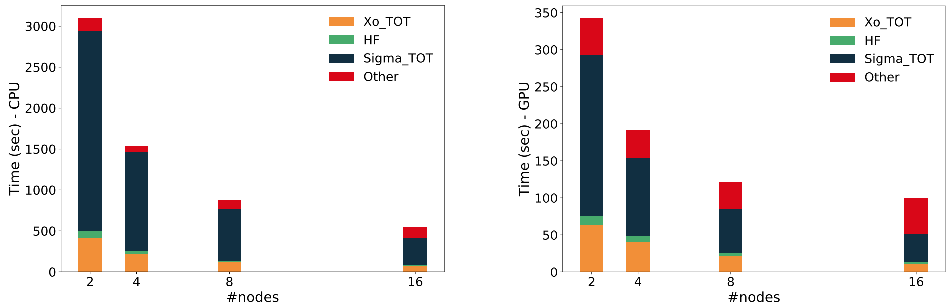


Figure 12: Full socket (left panel) vs GPU (right panel) timing for poly-acetylene on Piz Daint (XC50 partition). The time for each of the main tasks of the code is given separately. The total time taken to perform other tasks is labeled as "Other".

large computational partition. Given the very effective scalability of `Yambo` at both the MPI and OpenMP level (especially in the limit of small-medium number of tasks), the main issue threatening the use of `Yambo` on GPU-accelerated machines is the memory footprint. Current GPUs typically have 16 to 32 GB RAM per card, while `Yambo` can be very memory hungry (especially if the memory is distributed only over a few MPI tasks).

In this respect, testing the GPU porting of the code against larger systems would be very important (and is a currently ongoing activity). So far, besides poly-acetylene, we have also tested a larger system from the `Yambo` benchmark suite, namely the N7 graphene nanoribbon of Ref. [9]. Already on a single node with two NVIDIA TITAN V cards, results are very positive, meaning that the calculations went through and the performance is very good. Moreover, these preliminary tests have highlighted a bottleneck connected to a computational kernel not ported on GPUs (the calculation of the cut-off Coulomb potential in reciprocal space). A porting of this kernel will be addressed as soon as possible.

4.3 FLEUR

The performance of the DFT Code FLEUR [10] was substantially optimized during the first phase (2015-2018) of the MAX project. MPI parallelisation of the code was extended, shared memory parallelism was added, memory access was re-optimized, interfaces to optimized libraries were implemented. Those improvements not only drastically changed time-to-solution (up to 4 times) but also provided possibility to simulate much larger unit cells with over 1000 atoms [11]. Thus, the last FLEUR version of the first phase of the MAX project (MAX Release 3) will be the base for the performance optimization process during the phase two.

4.3.1 Performance of the FLEUR MAX Release 3

FLEUR is a full-potential all-electron DFT code, an implementation of the full-potential linearized augmented plane wave method (FLAPW). A usual run of the code consist of many self-consistency iteration cycles; the main parts of one cycle are: i) generation of potential, ii) setup of the Hamiltonian and overlap matrices, iii) solving the generalized eigenvalue problem, i.e. diagonalisation, iv) calculation of the new charge density. When several (or many) k-points need to be calculated, several (or many) independent matrices need to be set up and diagonalised. Number of k-points needed is usually inversely proportional to the size of a unit cell, e.g. for a cubic unit cells with about 1000 atoms one k-point can be sufficient.

MPI parallelisation of the FLEUR code consists of two layers: the first one for the k-points and the second one for the distributed set up and diagonalisation of the matrices. Since eigenvalue problems for different k-points are independent and show nearly ideal scaling, only one k-point is calculated in the test cases chosen for profiling. All self-consistency iterations are computationally identical, that is why for the performance tuning purposes we only run one iteration step.

The most time-consuming parts of the code are matrix setup, diagonalisation, and new charge generation, as one can see from the scaling plot of a test case TiO2 with 1078 atoms (Fig. 13). These three parts scale cubically with the system size. The generation of potential, which scales quadratically, stops playing a substantial part for unit cells of this size. Setup of the matrices ("HS setup", green) and solving the generalized eigenvalue problem ("Diagonalisation", blue) together are responsible for more that 95% of the computing time. These measurements were done on the CLAIX 2016 (Intel Broadwell, see Tab. 19) supercomputer at RWTH Aachen University, the plot starts with 16 nodes due to the memory requirements.

	CPU	# cores/ node	performance/ core, GFlops	memory/ node, GB	memory bandwidth, GB/s
CLAIX 2016	Intel Broadwell E5-2650v4	24	35	128	120
CLAIX 2018	Intel Skylake Platinum 8160	48	67	192	180

Table 19: Hardware systems used to perform the benchmark calculations with FLEUR.

It is apparent form the scaling diagram that the overall scalability of the code is

	Matrix setup	Diagonalisation	New charge	Total
CPI	0.58	0.38	0.61	0.48
Performance, GFlops	7.7	20.6	5.8	12.1

Table 20: Performance counter measurements done by LIKWID, average values per core. Code: FLEUR, hardware: CLAIX 2016, one node (24 cores). Test case: CuAg 256 atoms.

impeded mostly by the matrix setup part. The new charge generation part, though only takes less than 5% of the execution time, still can influence the total performance. To better understand factors affecting the performance, those two parts were investigated in more details (the diagonalisation part is outsourced to external libraries, in this case ELPA, and shows a good scaling behaviour). Using a smaller test case (CuAg, 256 atoms) with the similar scaling behaviour, performance counters were measured and traces were collected during the parallel execution of the code.

Performance counters for the matrix setup and new charge generation regions as well as for the diagonalisation part and total run were measured while running the code on one node with 4 MPI processes, 6 OpenMP threads per MPI process (Tab. 20). The values for CPI (cycles per instruction) and performance (number of double precision floating point operations per second) show that the code is quite performing, but still have a room for the improvement, especially in the new charge part.

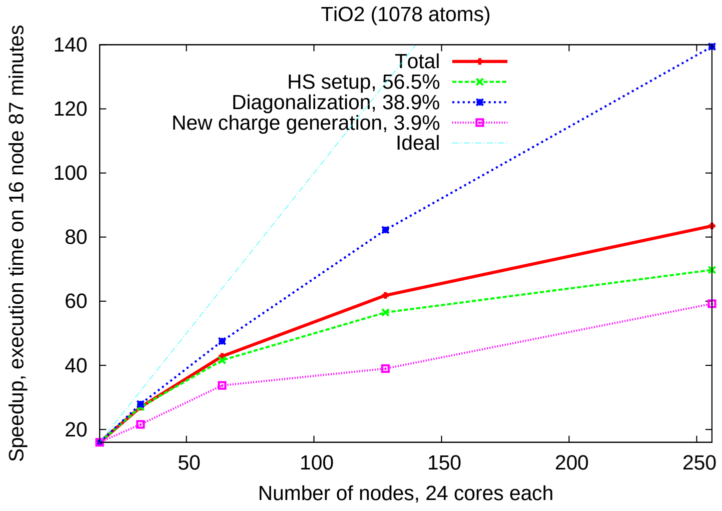


Figure 13: Strong scaling of the test case TiO2 (1078 atoms, 1 k-point, 1 self-consistency iteration step). The plot starts with 16 nodes, the percentage on the legend refers to this run. The total scaling (red) is compared with the scaling of the main parts (green, blue, magenta) and with the ideal scaling (light blue). Measurements were done on the CLAIX 2016 supercomputer at RWTH Aachen University (Tab. 19).

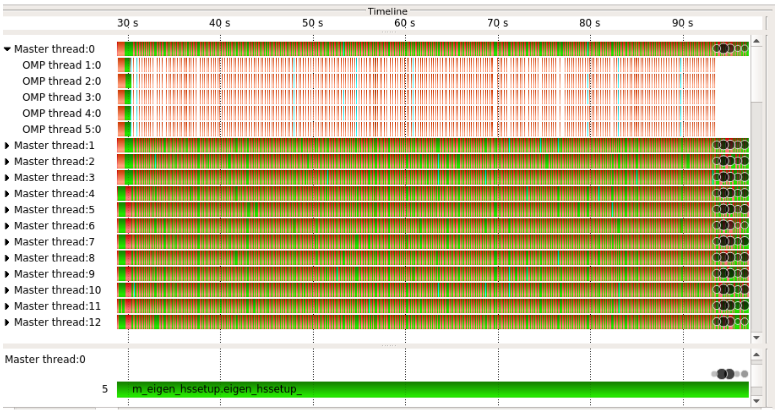


Figure 14: Trace of the parallel execution of the FLEUR code, matrix setup region. Only 12 of 32 MPI processes are shown, one the first MPI process is shown with its worker OpenMP threads.

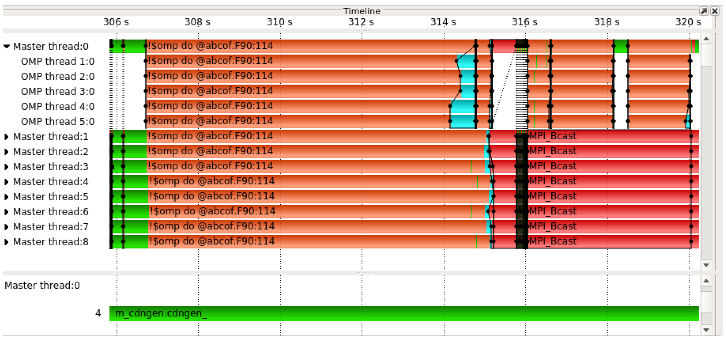


Figure 15: Trace of the parallel execution of the FLEUR code, new charge density region. Only 8 of 32 MPI processes are shown, one the first MPI process is shown with its worker OpenMP threads.

Traces for the Matrix setup and new charge generation regions are shown on Figs. 14 and 15 respectively. This was a parallel run on 8 nodes, 4 MPI processes per node, 6 OpenMP processes per MPI process. To save the place, only several MPI processes are shown, and only the first together with its worker OpenMP threads. We can see a decent load balance for the matrix setup part and a significant serial part in the new charge region, which can explain differences in CPI and performance values for these code parts.

4.3.2 New data layout

The compact way of storing the matrix, so-called packed storage was applyed in FLEUR in the MAX Release 3 and all preveous versions. At the time of implementation (i.e. several decades ago), when memory resources was scarce, it was a very reasonable optimization. Since the matrices are hermitian, only half of it was calculated and stored via one-dimentional array.

This strategy is not the best nowadays. In modern HPC systems, not the amount but the access to the memory introduces the main bottleneck. The most performant algo-

# nodes	16	32	64	128	256
MaX R3 (Packed storage), s	2966.62	1759.26	1141.23	839.98	680.23
Develop (Full matrix), s	1458.07	815.11	494.78	340.09	266.99
Speedup	2.03	2.16	2.30	2.47	2.55

Table 21: Execution time of the matrix setup. Two version are compared: packed storage (from the MAX release 3) vs. full matrix storage (the develop brunch, not released yet). Test case: TiO2 1078 atoms. Hardware: CLAIX2016

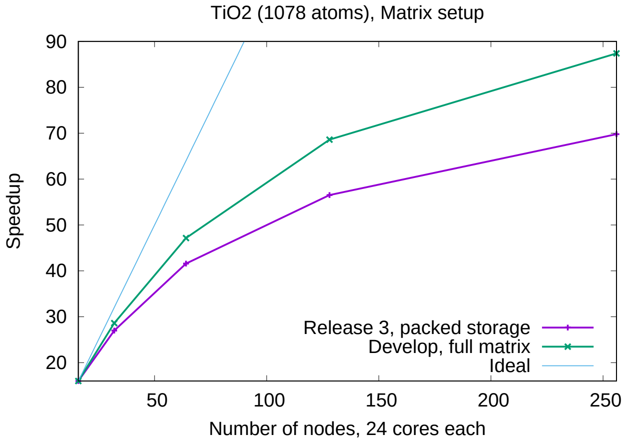


Figure 16: Strong scaling of the matrix setup part. Two version are compared: packed storage (from the MAX release 3) vs. two-dimentioanl storage (the develop brunch, not released yet). Test case: TiO2 1078 atoms. Hardware: CLAIX2016

rithms are now those which are able to use and reuse the data as many times as possible, reducing the streaming of the data from the main memory to the CPUs. Hence, storing data explicitly as matrices and allowing therefore the utilization of the standard optimized routins (such as BLAS2 and BLAS3) should provide better performance.

Restructuring the data layout in this way indeed improved performance considerably (Tab. 21): execution time became more than two times shorter. Also the scalability of this part of the code became more efficient (Fig. 16). The difference in the performance of the whole run can be seen on the Fig. 17, where the execution time of the one self-consistency iteration of the test case TiO2 (1078 atoms) is plotted. With this optimized version of the FLEUR the simulations of a larger variant of this test case (2156 atoms) became possible, which is also shown on this plot. Unfortunately, only three configurations were possible on this machine: it starts with 128 nodes because of the memory requirements and 512 nodes are nearly the whole machine.

After the second stage of the CLAIX machine, CLAIX 2018 (Tab. 19) became available, we repeated the calculations with these two test cases (Fig. 18) together with a new one: SrTiO3 with dislocations (STO).

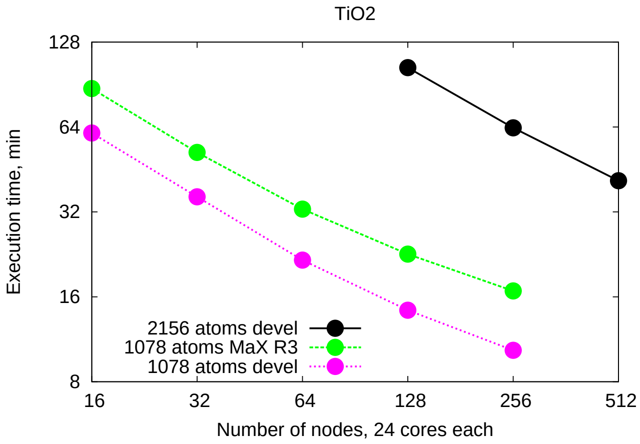


Figure 17: Scaling behaviour of the test case TiO2 (1 k-point, 1 self-consistency iteration). For the smaller setup (1078 atoms), the performance improvement of the current development version compared to the MAX Release 3 is shown (green and magenta). The simulation of a bigger test case (2156 atoms) became possible. Hardware: CLAIX 2016.

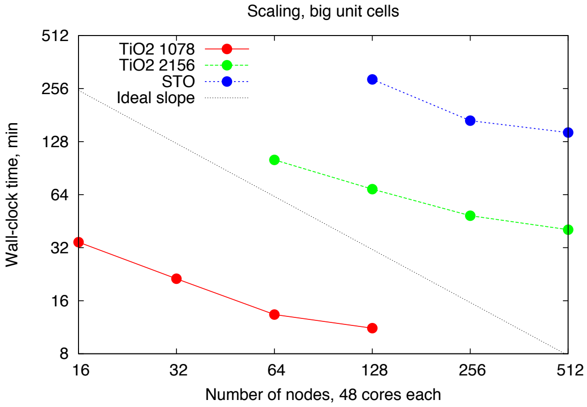


Figure 18: Scaling behaviour of big test cases: TiO2 (1078 atoms, 2156 atoms) and STO (3750 atoms). Hardware: CLAIX 2018.

4.4 BigDFT

4.4.1 Uranium-dioxyde benchmarks - GPU

Thanks to a highly efficient GPU implementation of a novel wavelet based algorithm for the evaluation of the exact exchange, we succeeded in reducing the cost of hybrid functional calculations in systematic basis sets by nearly one order of magnitude. As a consequence, hybrid functional calculations with our method are only about three times more expensive than a GGA functional calculation. This is a price that most scientists should be ready to pay for the significantly improved accuracy offered by such functionals. We expect that this development will pave the way for a very widespread use of hybrid functionals in combination with systematic basis sets. This in turn will greatly increase the predictive power of density-functional calculations and make them even more popular. Moreover, from a HPC viewpoint, the usage of such methods will enable extensive usage of Petaflop machines for electronic structure calculations communities, on the brink of the exascale era.

The calculation of the exact exchange energy E_X requires a double summation over all the N occupied orbitals

$$E_X[\hat{F}] = -\frac{1}{2}\sum_{i,j,\sigma} f_{i,\sigma} f_{j,\sigma} \int \mathrm{d}\mathbf{r}\,\mathrm{d}\mathbf{r}'\, \frac{\rho^{\sigma}_{ij}(\mathbf{r})\,\rho^{\sigma}_{ji}(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}\ , \tag{7}$$

where we have defined $\rho^{\sigma}_{ij}(\mathbf{r}) = \psi^{*}_{j,\sigma}(\mathbf{r})\,\psi_{i,\sigma}(\mathbf{r})$. The diagonal ($i = j$) contribution to E_X exactly cancels out the Hartree electrostatic energy $E_H[\rho]$. We explicitly specify the (collinear) spin degrees of freedom with the index $\sigma =\uparrow,\downarrow$, together with the occupation number $f_{i,\sigma}$. The action of the Fock operator $\hat{D}_X$ to be added to the KS Hamiltonian directly stems from the E_X definition:

$$\hat{D}_X\,|\psi_{i,\sigma}\rangle = -\sum_{j} f_{j,\sigma}\,|\hat{V}^{\sigma}_{ij}\psi_{j,\sigma}\rangle\ , \tag{8}$$

where we have defined the set of operators $\hat{V}^{\sigma}_{ij}$ with integral kernels

$$\langle\mathbf{r}|\,\hat{V}^{\sigma}_{ij}\,|\mathbf{r}'\rangle = \int \mathrm{d}\mathbf{r}''\frac{\rho^{\sigma}_{ji}(\mathbf{r}'')}{|\mathbf{r}-\mathbf{r}''|}\delta(\mathbf{r}-\mathbf{r}')\ , \tag{9}$$

that is the solution of the Poisson's equation $\nabla^2 V^{\sigma}_{ij} = -4\pi\rho^{\sigma}_{ij}$. In a KS-DFT code which searches for the ground state orbitals, one has to repeatedly evaluate, during the SCF procedure, for a given set of $\psi_{i,\sigma}(\mathbf{r})$, the value of E_X as well as the action of the corresponding Fock operator $\hat{D}_X$ on the entire set of occupied orbitals.

To give an idea of the computational workload of our calculations we have written in Tab. 22 the number of evaluations of the Poisson solver that the calculation of E_X and $\hat{D}_X$ (for a *single* wavefunction iteration) requires for the systems considered in these benchmarks.

The system we used to test our GPU accelerated runs is the Piz-Daint supercomputer in the Swiss national supercomputing center (CSCS), Lugano. Each run was performed using 1 MPI process per node, and 8 OpenMP threads per MPI process.

During the exact-exchange computation, with communication properly overlapped, each of the GPUs spends 80% of the time computing, and reaches 40 GFlop/s of sustained

N_a	12	96	324	768	1029
# ψ_i^σ	164	1432	5400	12800	17150
# ρ_{ij}^σ	6 658	513 372	7 292 700	81 926 400	73 539 200

Table 22: Number of Poisson solver evaluations per self-consistent iteration required to calculate the exact exchange energy and operator on the different systems used in the study. The number of atoms N_a as well as the number of KS orbitals ψ_i is indicated.

double precision performance. Memory throughput on the GPU is on average more than 165 GBps, over 65% of the peak theoretical bandwidth of the GPU, which is the limiting factor here. The CPU performance for the same run without acceleration on PIZ DAINT is 6.4 GFlops per CPU (using 8 OpenMP processes per CPU).

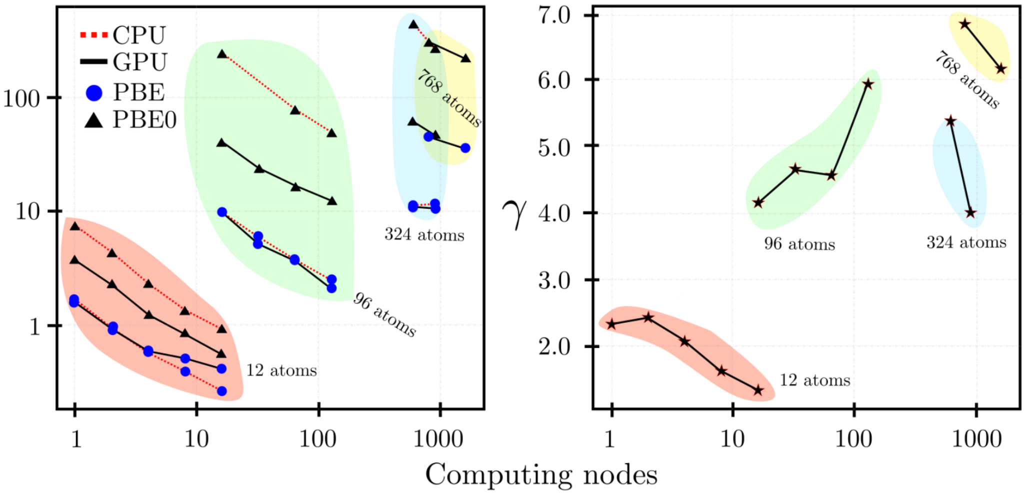


Figure 19: Timings (s) for PBE and PBE0 with and without GPU acceleration per iteration for different cells of UO_2 as a function of computing nodes on Piz-Daint. Right panels represent the ratio for a SCF iteration between a PBE0 and a PBE calculation.

As shown in Fig. 20 the GPU accelerated version scales well up to 3 orbitals/node. When going to two or even one orbital per node the degradation of the scalability is due to the fact that computation time is not high enough to overlap the communications. For one of the large systems used in this study (768 atoms, i.e. 12800 orbitals), the 3200 node run (4 orbitals/node) was 75% quicker than the 1600 nodes run. Even larger sizes (1029 atoms, resulting in 17150 orbitals, on up to 4288 nodes) were reached during this study, although network issues arose and prevented a good scalability. These limits, although only reached on sizes that are currently considered as uncommon for scientific production work, could be overcome by adding a wavelet based compression/uncompression step before each communication. This transformation is already implemented on the GPU in BIGDFT, and currently used only before entering the exact exchange step. On some systems this could reduce communication sizes by a factor of four. The new version of GPU-accelerated supercomputers like PIZ-DAINT or JEAN ZAY at IDRIS in France, provide a huge bump in computing power on the GPU, while the network was

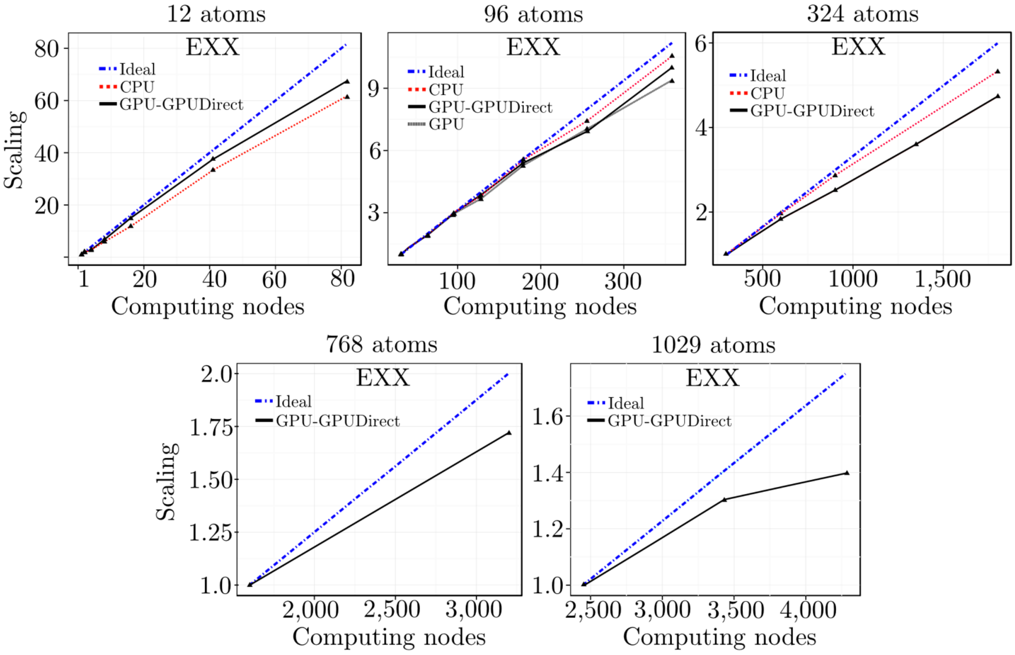


Figure 20: Strong parallel scaling of the exact exchange implementation in BigDFT for different systems (from a very small, 12 atom cell, up to a 1,029 atom cell). The ideal curve is depicted in blue, CPU, GPU and GPU-GPUDirect are included for small cells.

only slightly improved. This means that this optimization is even more important.

Another example of submission of the same systems in CINECA's Marconi KNL computer can be found in the notebook as the URL [12], where the AiiDA plugin is used to run the computations.

4.4.2 Bench Submission through AiiDA

We have developed the AiiDA [13] plugin for BigDFT code and we have inserted it in the development version of the library *PyBigDFT*, under stabilisation in the context of WP3. The plugin allows to integrate AiiDA effortlessly in preexisting BigDFT notebooks, enabling asynchronous deported execution on supercomputers as well as database tracking of results. To validate its technological readiness and usefulness we have ran some of the performance benchmarks with such a plugin. As an illustration we show a code snippet associated to the submission of the UO2 bench on Marconi.

```
from BigDFT import Datasets as D, Calculators as C
#instantiate the dataset of the runs to be performed
benchData=D.Dataset(label='bench',input=inp,posinp=posinpfile, queue_name='
    knl_usr_prod', account='Max_devel_0', async=True)
#append jobs following MPI and OMP
for nodes in list_of_nodes:
    for mpi in list_of_mpi_per_node:
        for omp in list_of_threads_per_process:
            code=C.AiidaCalculator(code="bigdft@marconi",num_machines=nodes,
                mpiprocs_per_machine=mpi,omp=omp, walltime=...)
            id={'name':jobname, 'nodes':nodes,'mpi':mpi,'omp':omp}
```

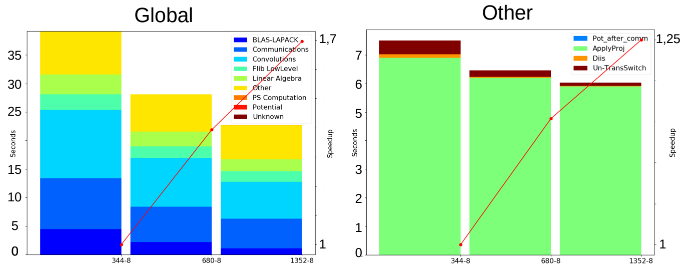


Figure 21: Scaling of a 3 iterations PBE run with UO2_3 input (5400 orbitals) on Marconi system, with 43-169 nodes, 8 MPI processes/node, 8 OMP threads/process - Global View on the left, Subsection Other on the right.

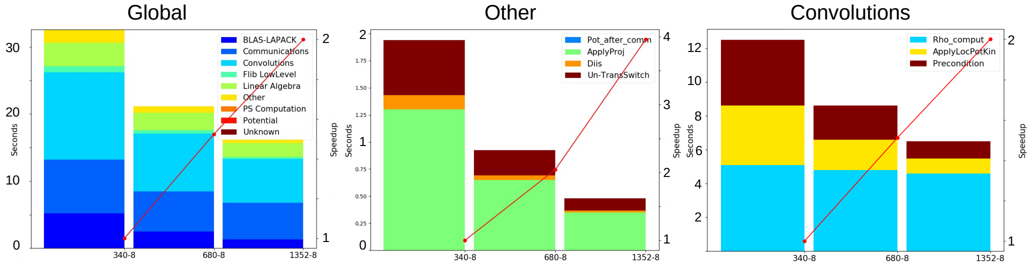


Figure 22: Scaling of a 3 iterations PBE run with UO2_3 input (5400 orbitals) on Marconi system, with pre-application of pseudopotentials with 43-169 nodes, 8 MPI processes/node, 8 OMP threads/process - Global View, Subsection Other, Subsection Convolutions.

```
            benchData.append_run(id=id,runner=code)
#submit and run the jobs remotely
benchData.run()
```

After completion, execution and performance data can be analyzed with *PyBigDFT* analysis tools, providing highly useful input on BigDFT's behavior. Time spent inside each section and subsection, imbalance at each level, and time spent inside routines can be inspected directly from the notebook used for submission.

4.4.3 Uranium-dioxyde benchmarks - KNL

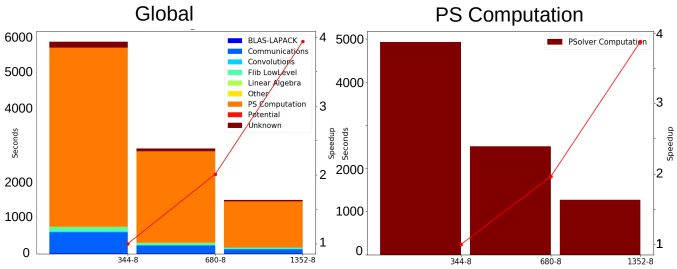


Figure 23: Scaling of a 3 iterations PBE0 run with UO2_3 input (5400 orbitals) on Marconi system, with 43-169 nodes, 8 MPI processes/node, 8 OMP threads/process - Global View, Subsection Poisson Solver.

UO2_3 PBE (Fig. 21) : Identifying bottlenecks in this run is simple, as every part of BigDFT is traced separately and each portion of the code can be isolated. Here, the "Other" section seems to not offer good scaling. Zooming on this section indeed reveals that the subsection "ApplyProj", corresponding to the application of the nonlocal pseudopotential, does not scale well on this particular run. This is due to a default choice in BigDFT to recompute pseudopotential application each time, without storing the resulting value in memory, to save memory, as this part is usually very small compared to convolutions. As this computation is local, it does not scale. An option in input file can be used to calculate the pseudopotentials once and for all and store the results, at the cost of memory occupancy (3GB of overhead per MPI process in this case, 24GB per node). As memory was not an issue in this case, we were able to run the experiments again and were able to get rid of this bottleneck. Fig. 22 shows the new runs, showing a maximum performance improvement of 30% in the computation phase. The convolutions section is now again the most costly, and the "Rho_comput" can now be seen as the more problematic section. This part handles the computation of the charge density. Convolutions are usually the most costly part of PBE execution, and the convolutions involved in these computation will be integrated in the general convolution library libconv, using autotuning strategies to select the most optimized version of each convolution kernel according to its parameters and the underlying platform. This work is currently part of WP2.

UO2_3 PBE0 (Fig. 23): Computation of the exact exchange part is the most expensive part in these experiments, as expected. Scaling of this section is near-perfect, thus improving computational throughput of the Poisson Solver is critical. Using GPUs yields much better performance than CPUs and even KNL processors, as seen on the experiments reported previously.

These benchmarks were reproduced on other larger datasets, with up to 800 KNL nodes on the Marconi system at Cineca, confirming the results previously obtained and the overall behavior of the application. Once the `AiiDA` plugin released, BigDFT will be able to provide simple and reproducible notebooks to perform benchmarking tasks and analysis on any platform it is installed on. This will allow performance comparison, bottleneck inspection and optimization on any new platform with very little effort.

4.5 CP2K

CP2K is a program to perform simulations of solid state, liquid, molecular and biological systems. It has a rich set of functionalities which allows to run different kinds of DFT and post-DFT simulations. CP2K relies on the performance of external parallel linear algebra libraries, such as ScaLAPACK and DBCSR (a library to perform sparse matrix-matrix multiplications). The aim of this work is to isolate and benchmark performance-critical parts of the CP2K code relevant for the scientific cases (Tab. 10, 11). For the scientific case (Tab. 12) the task is to create a base line plane-wave simulation with the SIRIUS back end (already fully GPU accelerated) and track the performance improvements for the duration of the project.

4.5.1 RPA calculations with CP2K

Evaluation of independent particle response function $\chi^{KS}(\mathbf{r}, \mathbf{r}', \omega)$ for each frequency point ω is the most expensive part of RPA calculations, accounting for up to 90% of the run time. Algorithmically this is expressed as a sequence of matrix-matrix multiplications, where the matrices represent the projections of the Kohn-Sham electron-hole pairs $\psi_c^\dagger(\mathbf{r})\psi_b(\mathbf{r})$ onto the auxiliary basis functions which are used to expand $\chi^{KS}(\mathbf{r}, \mathbf{r}', \omega)$ in spatial domain. The projection matrices are rectangular with a large imbalance between the number of rows (corresponding to the number of electron-hole pairs) and number of columns (corresponding to the number of auxiliary basis functions). For the RPA test of 128 water molecules this number are 17408 auxiliary basis functions and 3473408 electron-hole pairs. In CP2K the parallel matrix-matrix multiplication is done with ScaLAPACK which has a highly volatile performance depending on the internal parameters such as block sizes of a 2D block-cyclic distribution or BLACS grid dimensions. A much more stable performance can be obtained with the communication avoiding algorithm for matrix multiplications implemented in the COSMA library [14]. The work in in progress to implement PDGEMM wrapper for COSMA and interface it with the CP2K code. In the Tab. 4.5.1 we show the preliminary results for the matrix-matrix multiplication benchmark with the dimensions corresponding to the 128 water molecules test case.

Water molecules: 64			
Matrix sizes (m,n,k): 8704, 8704, 868352			
	ScaLAPACK (CPU)	COSMA (CPU)	COSMA (GPU)
Block sizes: 3392x2176 MPI grid: 128x2	1.92s	1.95s	1.65s
Block sizes: 32x32 MPI grid: 128x2	4.16s	2.2s	1.95s
Block sizes: 32x32 MPI grid: 16x16	13.83s	2.1s	1.8s

Water molecules: 128			
Matrix sizes (m,n,k): 17408, 17408, 3473408			
	ScaLAPACK (CPU)	COSMA (CPU)	COSMA (GPU)
Block sizes: 13568x4352 MPI grid: 128x2	21.5s	22.95s	15.1s
Block sizes: 32x32 MPI grid: 128x2	31.73s	26.75s	17.84s
Block sizes: 32x32 MPI grid: 16x16	116.72s	25.70s	17.55s

Table 23: Benchmark of the parallel matrix-matrix multiplication which accounts for the $\sim$90% of the RPA execution time. The cases of 64 and 128 water molecules are considered. The CPU runs were performed on multicore nodes of Piz Daint containing 36 CPU cores (2x 18-core Intel Browadwell). GPU runs were performed on the hybrid nodes of Piz Daint containing 12-core Intel Haswell + NVIDIA P100 card. COSMA library is much less sensitive to the MPI grid dimensions and block sizes; this is because COSMA is working with its own matrix storage layout to which the input matrices are converted.

4.5.2 Linear scaling calculations

In the linear scaling regime the costly diagonalisation of the Hamiltonian is avoided and replaced by the evaluation of the density matrix using the expansion of the Fermi operator. This leads to a sequence of sparse matrix-matrix multiplications which become a bottle neck in this type of calculations. In CP2K the sparse matrix multiplications are done using the DBCSR library [15] (formerly a part of CP2K code base, now a standalone reusable library). To get the optimal performance DBCSR is using auto-tuning, i.e. for a given CP2K Gaussian basis set it optimizes several parameters of matrix-matrix GPU multiplication kernels for all possible m, n, k triplets arising from this particular basis. Although the auto-tuning will give the best performance for the selected Gaussian basis, it has a significant drawback: for the unknown m, n, k triplets for which no auto-tuning was performed (i.e. when a user picked a new basis set) the GPU back end was not generated and the CPU back end will be called instead. This will result in a significant drop of performance if the user picked a basis for which DBCSR was not auto-tuned. This is also a problem for HPC centers who maintain the CP2K installation: there is no way to know for which basis they need to auto-tune and compile CP2K+DBCSR.

To solve this problem we have been working on the automatic determination of the optimal GPU matrix-matrix multiplication kernel parameters for the arbitrary m, n, k

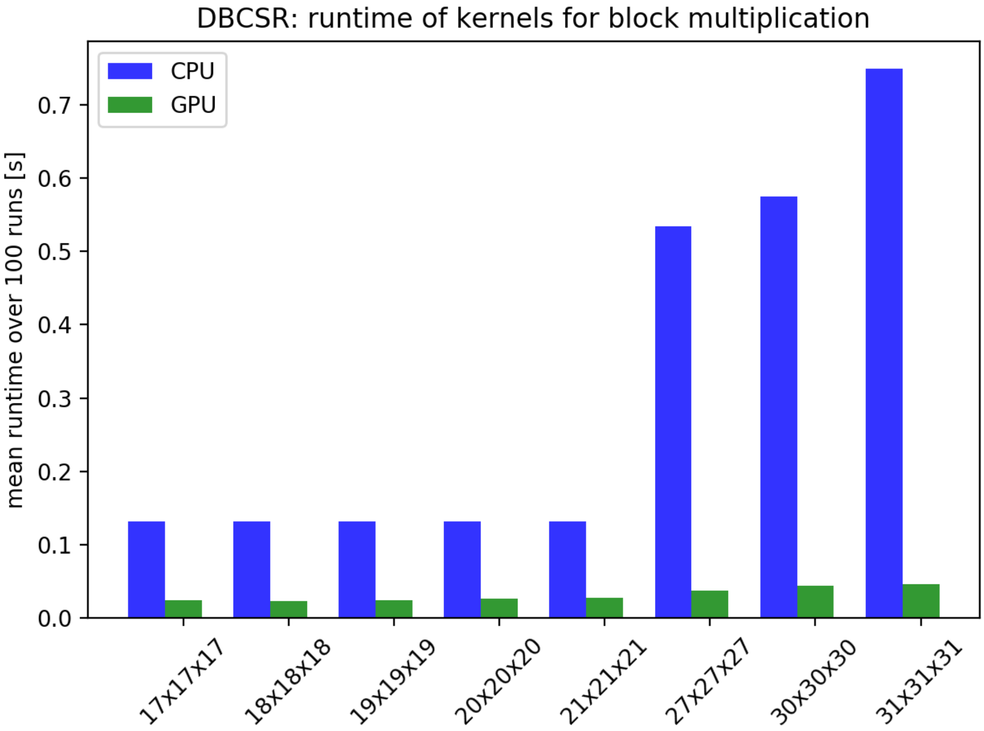


Figure 24: Using the machine learning in combination with JIT allows to add new GPU kernels with optimized parameter sets to the library without an additional cost of auto-tuning. This graph shows the comparison of the performance of the new GPU kernels added to the library thanks to JIT-ing with the performance of the fallback CPU kernels that would be used without these developments. Speedups easily exceed 10x for larger block sizes.

triplets using the machine learning technique. The model of the GPU kernel parameters is initially fit on a training dataset and then is used to generate GPU kernels for any m, n, k triplets with infered parameters. These kernels are then compiled on the fly at the first invocation of the library using CUDA's Just-in-Time (JIT) compilation capabilities. The preliminary results are shown on the Fig. 24.

The new version of the DBCSR library with JIT capabilities was tested on the linear scaling DFT run for $\approx$ 7000 water molecules on 256 nodes of Piz Daint, using an accurate, molecularly optimized TZVP basis. The full application performance was improved from 1150 sec. to 540 sec. in time-to-solution metric (>2x speedup).

4.5.3 Plane-wave pseudo potential calculations

As part of the ongoing project activities CP2K code was interfaced with the SIRIUS library [16] which implements plane wave pseudopotential and augmented plane wave full-potential methods of the DFT. The primary objective of this development is to provide an easy validation and verification mechanism for the Gaussian basis set of CP2K

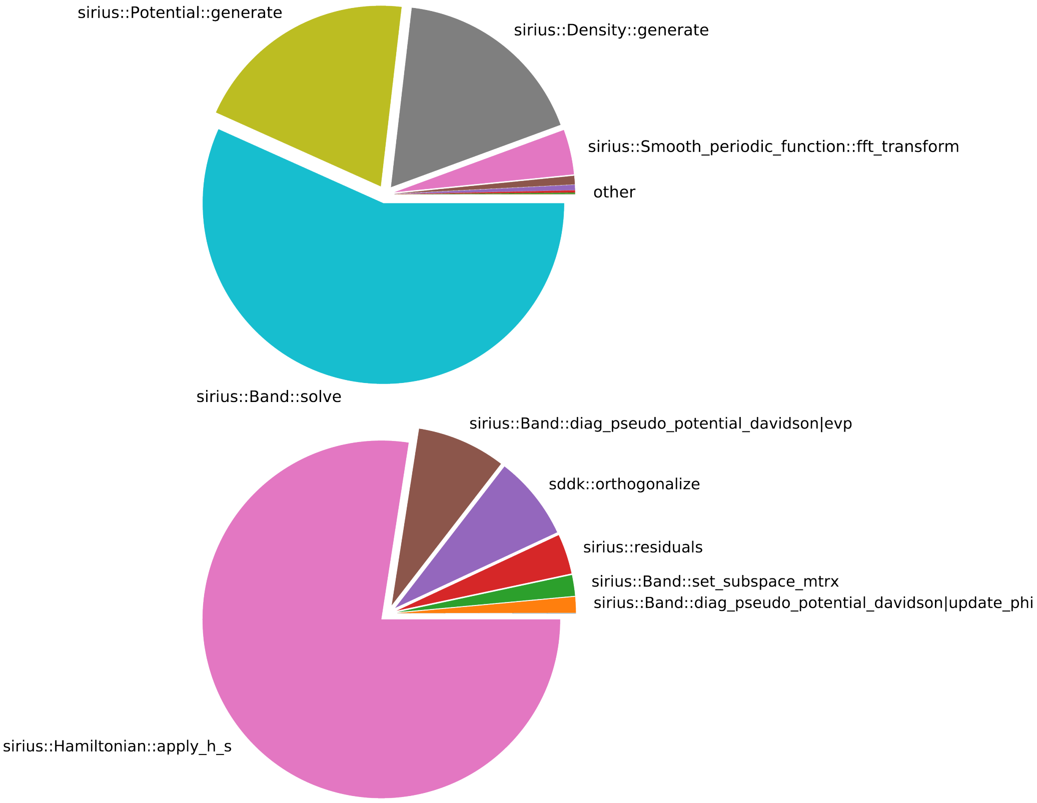


Figure 25: Distribution of time in the CP2K+SIRIUS plane-wave pseudopotential ground state calculation of C_{60} molecule in the box. Top: diagonalization of the Hamiltonian and generation of density and potential are the main time consumers. Bottom: application of the Kohn-Sham Hamiltonian is the most expensive part of the iterative diagonalization process. FFT transformation of the wave-functions which are done during application of the Hamiltonian and charge density summation are the bottle neck for this type of calculations.

by enabling the reference plane-wave ground state calculation inside CP2K using the same input file and the same pseudopotential. As a benchmark we picked a single C_{60} molecule in a large box and run a DFT ground state on 4–16 nodes to establish the baseline performance values. The results are collected in Tab. 4.5.3

SIRIUS library measures the execution time of its individual components and provides a comprehensive timing report at the end of the run. This allows for a quick and lightweight analysis of the library performance, as, for example, shown in Fig. 25.

Number of nodes	Time to solution²
4	548.7s
9	305.4s
16	213.1s

Table 24: Time to solution for the plane wave pseudopotential DFT ground state of C_{60} molecule done with CP2K+SIRIUS. The runs were performed on the hybrid nodes of Piz Daint containing 12-core Intel Haswell + NVIDIA P100 card.

4.6 SIESTA

During the reporting period we have profiled the performance of SIESTA. We have used the powerful performance analysis tools developed at BSC [3] that afford a very detailed understanding of the code's behavior while affecting its execution only minimally [4]. The profiling work was done on MareNostrum IV[5] at the Barcelona Supercomputing Center. Each node of this machine contains 48 Intel Platinum 8160 @ 2.1 GHz CPU cores.

In Fig. 26 we show the timings and parallel efficiency (with the efficiency of 96 cores defined as 1.0) for a specific large use case, namely a short DNA strand containing about 11,000 atoms. We performed two geometry iterations, each with five iterations of the main SCF loop. As can be seen, the overall scaling is not good, with an efficiency of only 30% for 2300 cores. The main culprits for the performance degradation, taking into account CPU usage, are the initialisation stages (state_init and Setup_H0), with an efficiency of 7% and 3%, respectively, which take more than half of the overall CPU time when running on 2300 cores. The routine state_init is called before processing each new geometry, and Setup_H0 is called at the beginning of each scf loop. Even though these routines are only called once [6] , their scaling is bad enough to considerably affect the overall performance, and are thus bottlenecks that should be addressed.

We turned first to the routine nlefsm, part of Setup_H0. This routine computes the matrix elements of the non-local part of the pseudopotential, involving products of overlaps between Kleinman-Bylander (KB) projectors and orbitals. Orbitals are fully distributed in SIESTA, but KB projectors are handled implicitly by all processes via their positions, and in fact some of the same overlaps are computed several times by different processes. This is unavoidable and leads to reduced scaling. In addition, this routine, in its first call, was computing the interpolation tables for the overlaps as they were requested (further calls just use the tables). Hence, a reduced scaling was coupled with heavy CPU use in the initialization call, leading to a severe bottleneck.

Rather than just marking it as such and include it in the list of actions to be taken later on, we felt that it would be important to fix it immediately, since its presence could mask other possible bottlenecks, and would also unnecessarily inflate the CPU time of further tests. The solution was actually quite straightforward: to pre-compute the needed interpolation tables in a more favorable setting (in fact, in a fully distributed way: routine initMatel). Then the initial call to nlefsm just uses the tables and takes very little CPU time.

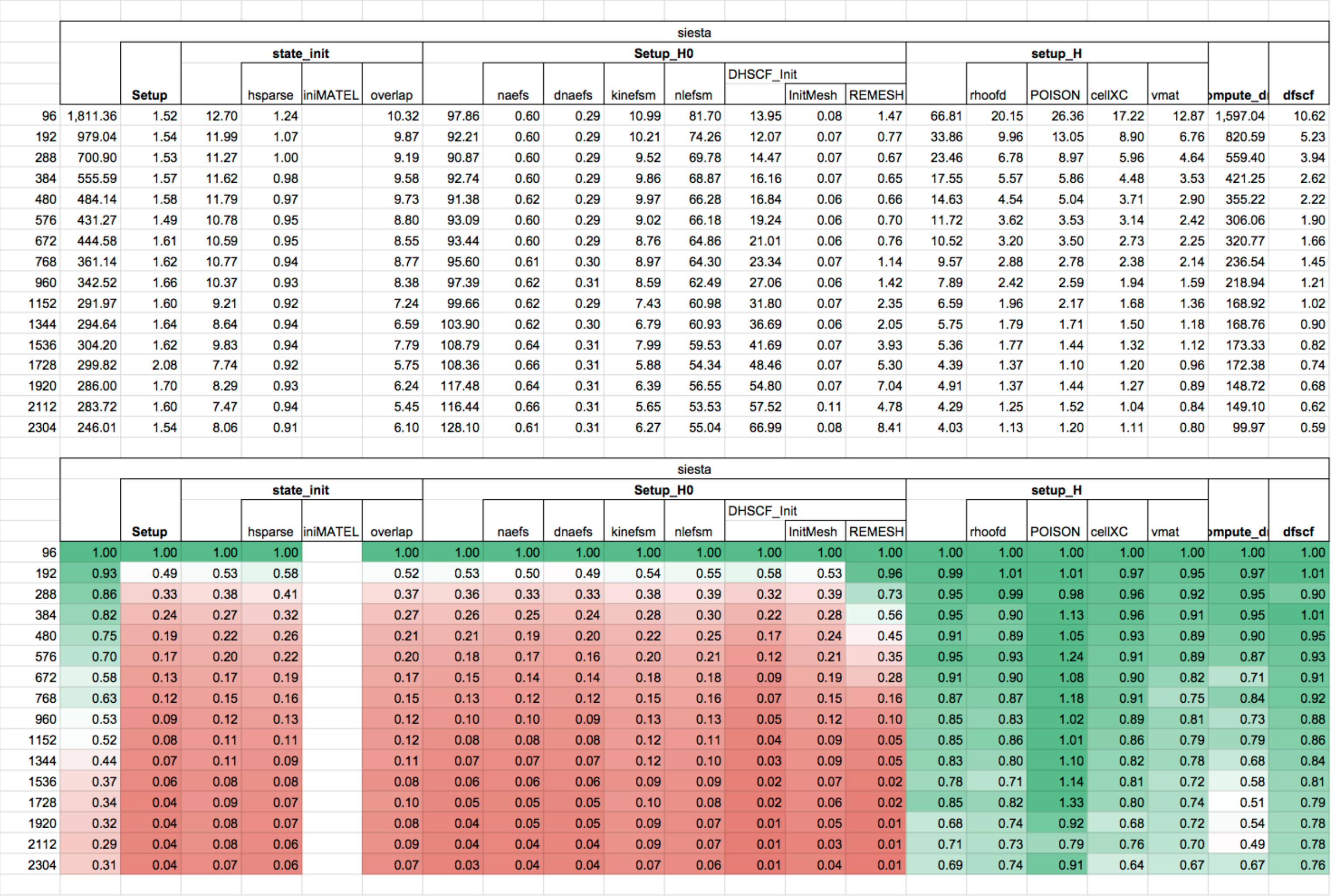
For the same reasons of clarity and economy, we also fixed the bottleneck in the dhscf_init routine, which sets up the data structures in the real-space grid used for computing the charge density and the matrix elements of the effective potential. Here the problem was that the pre-scheduling of the needed communications, implemented some years ago using graph-coloring techniques, became very costly for large numbers of processors. Asynchronous communications have been used instead. Finally, a number of

[3] https://www.bsc.es/discover-bsc/organisation/scientific-structure/performance-tools

[4] It often happens that a detailed profiling of a code leads to a considerable slow down, which might affect the code execution itself and thus bias the profiling. With the BSC tools this is not the case.

[5] https://www.bsc.es/marenostrum/marenostrum

[6] This is true for one single SCF calculation. In the case of a geometry optimization or Molecular Dynamics, the initialisation routines are called more than once.



	siesta																				
			state_init				Setup_H0								setup_H						
												DHSCF_Init									
		Setup		hsparse	iniMATEL	overlap		naefs	dnaefs	kinefsm	nlefsm		InitMesh	REMESH		rhoofd	POISON	cellXC	vmat	ompute_d	dfscf
96	1,811.36	1.52	12.70	1.24		10.32	97.86	0.60	0.29	10.99	81.70	13.95	0.08	1.47	66.81	20.15	26.36	17.22	12.87	1,597.04	10.62
192	979.04	1.54	11.99	1.07		9.87	92.21	0.60	0.29	10.21	74.26	12.07	0.07	0.77	33.86	9.96	13.05	8.90	6.76	820.59	5.23
288	700.90	1.53	11.27	1.00		9.19	90.87	0.60	0.29	9.52	69.78	14.47	0.07	0.67	23.46	6.78	8.97	5.96	4.64	559.40	3.94
384	555.59	1.57	11.62	0.98		9.58	92.74	0.60	0.29	9.86	68.87	16.16	0.07	0.65	17.55	5.57	5.86	4.48	3.53	421.25	2.62
480	484.14	1.58	11.79	0.97		9.73	91.38	0.62	0.29	9.97	66.28	16.84	0.06	0.66	14.63	4.54	5.04	3.71	2.90	355.22	2.22
576	431.27	1.49	10.78	0.95		8.80	93.09	0.60	0.29	9.02	66.18	19.24	0.06	0.70	11.72	3.62	3.53	3.14	2.42	306.06	1.90
672	444.58	1.61	10.59	0.95		8.55	93.44	0.60	0.29	8.76	64.86	21.01	0.06	0.76	10.52	3.20	3.50	2.73	2.25	320.77	1.66
768	361.14	1.62	10.77	0.94		8.77	95.60	0.61	0.30	8.97	64.30	23.34	0.07	1.14	9.57	2.88	2.78	2.38	2.14	236.54	1.45
960	342.52	1.66	10.37	0.93		8.38	97.39	0.62	0.31	8.59	62.49	27.06	0.06	1.42	7.89	2.42	2.59	1.94	1.59	218.94	1.21
1152	291.97	1.60	9.21	0.92		7.24	99.66	0.62	0.29	7.43	60.98	31.80	0.07	2.35	6.59	1.96	2.17	1.68	1.36	168.92	1.02
1344	294.64	1.64	8.64	0.94		6.59	103.90	0.62	0.30	6.79	60.93	36.69	0.06	2.05	5.75	1.79	1.71	1.50	1.18	168.76	0.90
1536	304.20	1.62	9.83	0.94		7.79	108.79	0.64	0.31	7.99	59.53	41.69	0.07	3.93	5.36	1.77	1.44	1.32	1.12	173.33	0.82
1728	299.82	2.08	7.74	0.92		5.75	108.36	0.66	0.31	5.88	54.34	48.46	0.07	5.30	4.39	1.37	1.10	1.20	0.96	172.38	0.74
1920	286.00	1.70	8.29	0.93		6.24	117.48	0.64	0.31	6.39	56.55	54.80	0.07	7.04	4.91	1.37	1.44	1.27	0.89	148.72	0.68
2112	283.72	1.60	7.47	0.94		5.45	116.44	0.66	0.31	5.65	53.53	57.52	0.11	4.78	4.29	1.25	1.52	1.04	0.84	149.10	0.62
2304	246.01	1.54	8.06	0.91		6.10	128.10	0.61	0.31	6.27	55.04	66.99	0.08	8.41	4.03	1.13	1.20	1.11	0.80	99.97	0.59

	siesta																				
			state_init				Setup_H0								setup_H						
												DHSCF_Init									
		Setup		hsparse	iniMATEL	overlap		naefs	dnaefs	kinefsm	nlefsm		InitMesh	REMESH		rhoofd	POISON	cellXC	vmat	ompute_d	dfscf
96	1.00	1.00	1.00	1.00		1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
192	0.93	0.49	0.53	0.58		0.52	0.53	0.50	0.49	0.54	0.55	0.58	0.53	0.96	0.99	1.01	1.01	0.97	0.95	0.97	1.01
288	0.86	0.33	0.38	0.41		0.37	0.36	0.33	0.33	0.38	0.39	0.32	0.39	0.73	0.95	0.99	0.98	0.96	0.92	0.95	0.90
384	0.82	0.24	0.27	0.32		0.27	0.26	0.25	0.24	0.28	0.30	0.22	0.28	0.56	0.95	0.90	1.13	0.96	0.91	0.95	1.01
480	0.75	0.19	0.22	0.26		0.21	0.21	0.19	0.20	0.22	0.25	0.17	0.24	0.45	0.91	0.89	1.05	0.93	0.89	0.90	0.95
576	0.70	0.17	0.20	0.22		0.20	0.18	0.17	0.16	0.20	0.21	0.12	0.21	0.35	0.95	0.93	1.24	0.91	0.89	0.87	0.93
672	0.58	0.13	0.17	0.19		0.17	0.15	0.14	0.14	0.18	0.18	0.09	0.19	0.28	0.91	0.90	1.08	0.90	0.82	0.71	0.91
768	0.63	0.12	0.15	0.16		0.15	0.13	0.12	0.12	0.15	0.16	0.07	0.15	0.16	0.87	0.87	1.18	0.91	0.75	0.84	0.92
960	0.53	0.09	0.12	0.13		0.12	0.10	0.10	0.09	0.13	0.13	0.05	0.12	0.10	0.85	0.83	1.02	0.89	0.81	0.73	0.88
1152	0.52	0.08	0.11	0.11		0.12	0.08	0.08	0.08	0.12	0.11	0.04	0.09	0.05	0.85	0.86	1.01	0.86	0.79	0.79	0.86
1344	0.44	0.07	0.11	0.09		0.11	0.07	0.07	0.07	0.12	0.10	0.03	0.09	0.05	0.83	0.80	1.10	0.82	0.78	0.68	0.84
1536	0.37	0.06	0.08	0.08		0.08	0.06	0.06	0.06	0.09	0.09	0.02	0.07	0.02	0.78	0.71	1.14	0.81	0.72	0.58	0.81
1728	0.34	0.04	0.09	0.07		0.10	0.05	0.05	0.05	0.10	0.08	0.02	0.06	0.02	0.85	0.82	1.33	0.80	0.74	0.51	0.79
1920	0.32	0.04	0.08	0.07		0.08	0.04	0.05	0.05	0.09	0.07	0.01	0.05	0.01	0.68	0.74	0.92	0.68	0.72	0.54	0.78
2112	0.29	0.04	0.08	0.06		0.09	0.04	0.04	0.04	0.09	0.07	0.01	0.03	0.01	0.71	0.73	0.79	0.76	0.70	0.49	0.78
2304	0.31	0.04	0.07	0.06		0.07	0.03	0.04	0.04	0.07	0.06	0.01	0.04	0.01	0.69	0.74	0.91	0.64	0.67	0.67	0.76

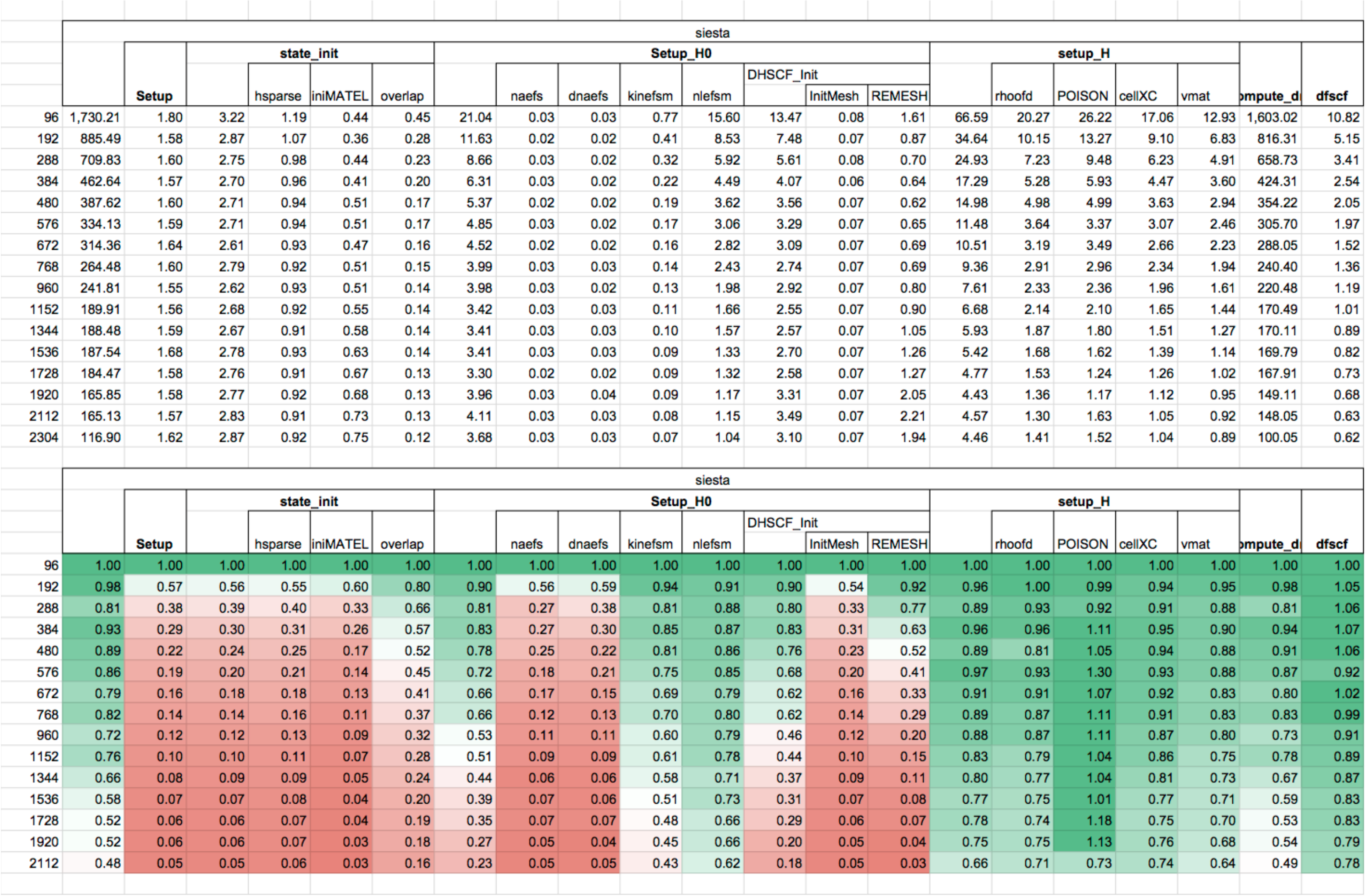
Figure 26: Timings and parallel efficiency of the reference version of the code, showing in red sections of reduced scalability.

smaller changes, mostly involving distribution of work that previously was done serially in naefs, dnaefs, and kinefsm, complete the preliminary work done up to now.

Fig. 27 shows the new timings and parallel efficiency. While several red sections of reduced scalability are still shown, their associated CPU times are now quite small. Overall, the optimisation achieved in this preliminary stage of the analysis has reduced considerably the initialization effort in the program, something which is intrinsically useful but also very relevant for the operation of SIESTA as a "callable quantum engine", as foreseen in the WP1 work-package in MAX. A graphical representation of this can be seen in the trace in Fig.28 : the top panel shows the original trace, with an oversized initialization section followed by two groups of five SCF steps. In the bottom panel the initialization section has been reduced almost to zero on the scale of the whole calculation. In Fig.29 we show a zoom of the initialization part, where the big improvements are even more noticeable. As can be seen, some routines have virtually vanished in the optimised version of the code.

Some more tweaks to a few routines are still possible, but they are deferred to the next stage. What the preliminary analysis and early fixes have uncovered is that the overall scalability of the program in this profiling run is mostly limited by the solver stage, which takes almost 90% of the CPU time but shows an efficiency of a bit less than 50% when running with 2100 cores. This is actually not a bad relative result: matrix diagonalisation, as implemented in distributed form in the ScaLAPACK library, is known to exhibit a severe performance degradation for large numbers of processes. Our calculations are done with the PEXSI solver, which circumvents diagonalisation by constructing directly the density-matrix using combined pole-expansion and selected-inversion algorithms. It exploits the sparsity of the H and S matrices in SIESTA, resulting in an improved complexity scaling as a function of system size N, namely $N^{(0,5+D/2)}$, where D is the dimensionality of the system. As can be seen in Ref. [17], the PEXSI solver is dramatically more efficient and scalable than ScaLAPACK for this kind of system.

Most of the remaining 10% of the CPU time is taken by the routines called to compute the Hamiltonian during the SCF cycle. These exhibit a linear scaling with system size, and their parallelisation is quite efficient, around 90% for about 1000 processes, and around 70% for 2100 MPI processes. This strong dependence on the solver is demonstrated in Fig. 30 , where we compare the scaling of the initial version of the code with the optimized one. The big improvements are clearly visible. Moreover, we see that the scaling of the optimized version is virtually the same as the scaling of the solver part alone, demonstrating that this section represents the big remaining bottleneck.



	siesta																				
			state_init				Setup_H0								setup_H						
												DHSCF_Init									
		Setup		hsparse	iniMATEL	overlap		naefs	dnaefs	kinefsm	nlefsm		InitMesh	REMESH		rhoofd	POISON	cellXC	vmat	ompute_d	dfscf
96	1,730.21	1.80	3.22	1.19	0.44	0.45	21.04	0.03	0.03	0.77	15.60	13.47	0.08	1.61	66.59	20.27	26.22	17.06	12.93	1,603.02	10.82
192	885.49	1.58	2.87	1.07	0.36	0.28	11.63	0.02	0.02	0.41	8.53	7.48	0.07	0.87	34.64	10.15	13.27	9.10	6.83	816.31	5.15
288	709.83	1.60	2.75	0.98	0.44	0.23	8.66	0.03	0.02	0.32	5.92	5.61	0.08	0.70	24.93	7.23	9.48	6.23	4.91	658.73	3.41
384	462.64	1.57	2.70	0.96	0.41	0.20	6.31	0.03	0.02	0.22	4.49	4.07	0.06	0.64	17.29	5.28	5.93	4.47	3.60	424.31	2.54
480	387.62	1.60	2.71	0.94	0.51	0.17	5.37	0.02	0.02	0.19	3.62	3.56	0.07	0.62	14.98	4.98	4.99	3.63	2.94	354.22	2.05
576	334.13	1.59	2.71	0.94	0.51	0.17	4.85	0.03	0.02	0.17	3.06	3.29	0.07	0.65	11.48	3.64	3.37	3.07	2.46	305.70	1.97
672	314.36	1.64	2.61	0.93	0.47	0.16	4.52	0.02	0.02	0.16	2.82	3.09	0.07	0.69	10.51	3.19	3.49	2.66	2.23	288.05	1.52
768	264.48	1.60	2.79	0.92	0.51	0.15	3.99	0.03	0.03	0.14	2.43	2.74	0.07	0.69	9.36	2.91	2.96	2.34	1.94	240.40	1.36
960	241.81	1.55	2.62	0.93	0.51	0.14	3.98	0.03	0.02	0.13	1.98	2.92	0.07	0.80	7.61	2.33	2.36	1.96	1.61	220.48	1.19
1152	189.91	1.56	2.68	0.92	0.55	0.14	3.42	0.03	0.03	0.11	1.66	2.55	0.07	0.90	6.68	2.14	2.10	1.65	1.44	170.49	1.01
1344	188.48	1.59	2.67	0.91	0.58	0.14	3.41	0.03	0.03	0.10	1.57	2.57	0.07	1.05	5.93	1.87	1.80	1.51	1.27	170.11	0.89
1536	187.54	1.68	2.78	0.93	0.63	0.14	3.41	0.03	0.03	0.09	1.33	2.70	0.07	1.26	5.42	1.68	1.62	1.39	1.14	169.79	0.82
1728	184.47	1.58	2.76	0.91	0.67	0.13	3.30	0.02	0.02	0.09	1.32	2.58	0.07	1.27	4.77	1.53	1.24	1.26	1.02	167.91	0.73
1920	165.85	1.58	2.77	0.92	0.68	0.13	3.96	0.03	0.04	0.09	1.17	3.31	0.07	2.05	4.43	1.36	1.17	1.12	0.95	149.11	0.68
2112	165.13	1.57	2.83	0.91	0.73	0.13	4.11	0.03	0.03	0.08	1.15	3.49	0.07	2.21	4.57	1.30	1.63	1.05	0.92	148.05	0.63
2304	116.90	1.62	2.87	0.92	0.75	0.12	3.68	0.03	0.03	0.07	1.04	3.10	0.07	1.94	4.46	1.41	1.52	1.04	0.89	100.05	0.62

	siesta																				
			state_init				Setup_H0								setup_H						
												DHSCF_Init									
		Setup		hsparse	iniMATEL	overlap		naefs	dnaefs	kinefsm	nlefsm		InitMesh	REMESH		rhoofd	POISON	cellXC	vmat	ompute_d	dfscf
96	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
192	0.98	0.57	0.56	0.55	0.60	0.80	0.90	0.56	0.59	0.94	0.91	0.90	0.54	0.92	0.96	1.00	0.99	0.94	0.95	0.98	1.05
288	0.81	0.38	0.39	0.40	0.33	0.66	0.81	0.27	0.38	0.81	0.88	0.80	0.33	0.77	0.89	0.93	0.92	0.91	0.88	0.81	1.06
384	0.93	0.29	0.30	0.31	0.26	0.57	0.83	0.27	0.30	0.85	0.87	0.83	0.31	0.63	0.96	0.96	1.11	0.95	0.90	0.94	1.07
480	0.89	0.22	0.24	0.25	0.17	0.52	0.78	0.25	0.22	0.81	0.86	0.76	0.23	0.52	0.89	0.81	1.05	0.94	0.88	0.91	1.06
576	0.86	0.19	0.20	0.21	0.14	0.45	0.72	0.18	0.21	0.75	0.85	0.68	0.20	0.41	0.97	0.93	1.30	0.93	0.88	0.87	0.92
672	0.79	0.16	0.18	0.18	0.13	0.41	0.66	0.17	0.15	0.69	0.79	0.62	0.16	0.33	0.91	0.91	1.07	0.92	0.83	0.80	1.02
768	0.82	0.14	0.14	0.16	0.11	0.37	0.66	0.12	0.13	0.70	0.80	0.62	0.14	0.29	0.89	0.87	1.11	0.91	0.83	0.83	0.99
960	0.72	0.12	0.12	0.13	0.09	0.32	0.53	0.11	0.11	0.60	0.79	0.46	0.12	0.20	0.88	0.87	1.11	0.87	0.80	0.73	0.91
1152	0.76	0.10	0.10	0.11	0.07	0.28	0.51	0.09	0.09	0.61	0.78	0.44	0.10	0.15	0.83	0.79	1.04	0.86	0.75	0.78	0.89
1344	0.66	0.08	0.09	0.09	0.05	0.24	0.44	0.06	0.06	0.58	0.71	0.37	0.09	0.11	0.80	0.77	1.04	0.81	0.73	0.67	0.87
1536	0.58	0.07	0.07	0.08	0.04	0.20	0.39	0.07	0.06	0.51	0.73	0.31	0.07	0.08	0.77	0.75	1.01	0.77	0.71	0.59	0.83
1728	0.52	0.06	0.06	0.07	0.04	0.19	0.35	0.07	0.07	0.48	0.66	0.29	0.06	0.07	0.78	0.74	1.18	0.75	0.70	0.53	0.83
1920	0.52	0.06	0.06	0.07	0.03	0.18	0.27	0.05	0.04	0.45	0.66	0.20	0.05	0.04	0.75	0.75	1.13	0.76	0.68	0.54	0.79
2112	0.48	0.05	0.05	0.06	0.03	0.16	0.23	0.05	0.05	0.43	0.62	0.18	0.05	0.03	0.66	0.71	0.73	0.74	0.64	0.49	0.78

Figure 27: Timings and parallel efficiency of the version of the code with improved initialization routines.

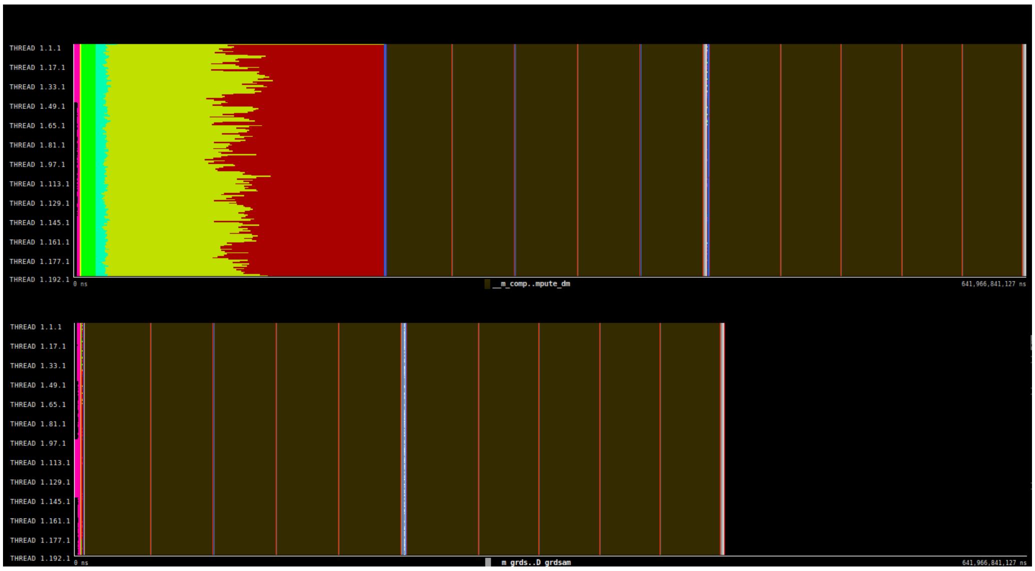


Figure 28: Traces of parallel execution of the code: top panel, reference version; bottom panel: version with fixes to the initialization bottlenecks.

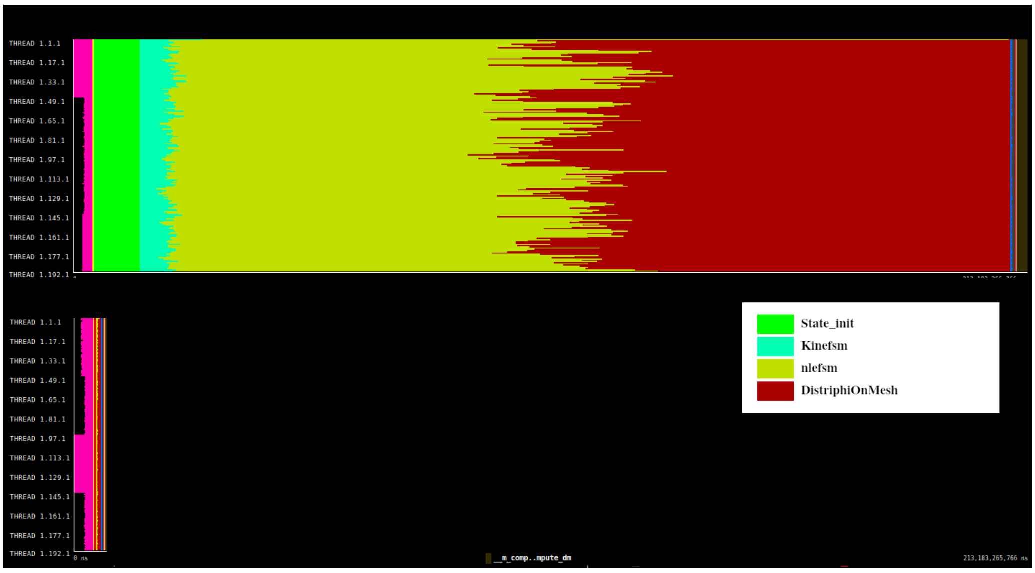


Figure 29: Traces showing a zoom of the initialization part: top panel, reference version; bottom panel: improved version.

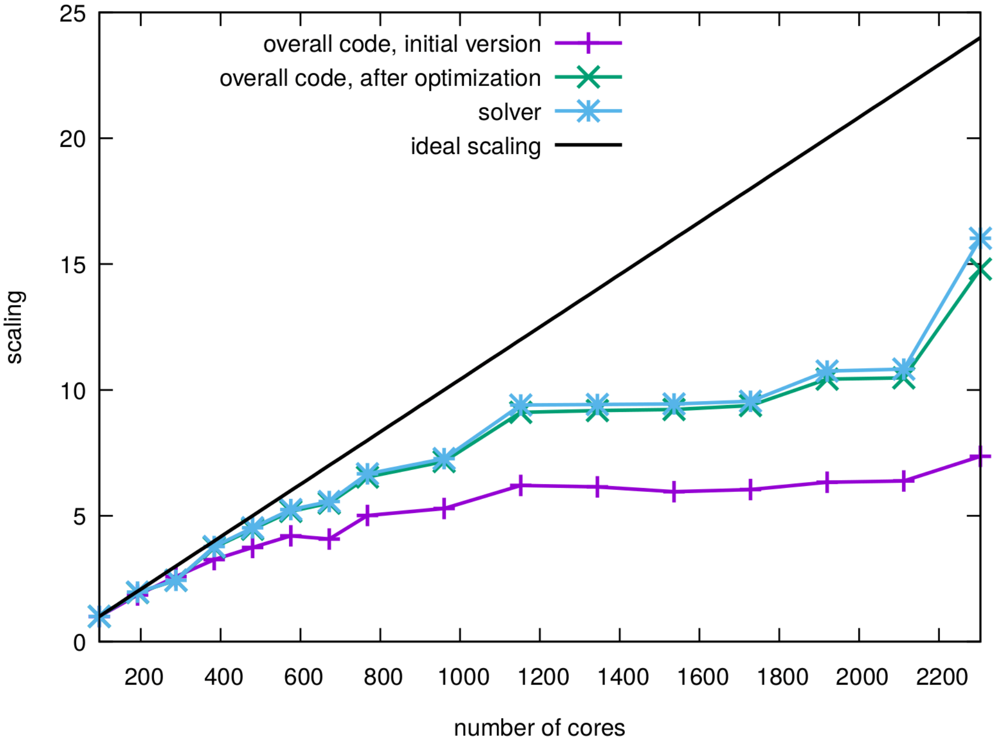


Figure 30: Scaling of the initial and the optimized code, together with the scaling of the solver part alone. The data is taken from Fig. 26 and Fig. 27 .

5 Structured plan of forward activities

5.1 QUANTUM ESPRESSO

Optimise multi-threading and accelerator offloading. A general optimization of the usage of multi-threading and the offloading of massive parallel parts to accelerators will allow to improve the performances for those use cases in which each diagonalisation pool may be contained within a single node. In these cases the efficient offloading to accelerators will make FFT and parallel linear algebra more efficient, avoiding communication bottlenecks. Multi-threading will be used to share the work done on bands (those contained within each band group) and projectors. This optimization will be crucial for the sustainability of high throughput computations in exascale machines and will also allow to use efficiently the code in the new generations of workstations equipped with accelerator devices.

Improve the scalability of iterative eigensolvers. Profiling of very large use cases demonstrates that the Davidson iterative diagonalisation turns to be a major bottleneck. Davidson operates on the whole block of eigenvectors at the same time. With a moderate number of eigenstates this is in fact an advantage with respect to methods where eigenstates must be accessed sequentially like the CG diagonalisation, but requires an exact diagonalisation of $N_D \times N_D$ matrices with $N_D \geq 2N_B$, number of bands. For this reason we are currently considering alternative methods which allow to deal with smaller-size blocks of eigenstates. Methods currently under scrutiny are RMM-DIIS [18], ParO [19] and PPCG [20].

Rationalization of I/O. The profiling the large-size use cases indicates that the I/O policy currently adopted in QUANTUM ESPRESSO, both when reading and when writing, may become inadequate when the number of MPI tasks is very large. Planned actions include:

- Read input, restart (XML) files, and pseudopotential files on a single MPI task, broadcast data to all other tasks;
- Examine the possibility to save large binary records (wavefunctions and charge density) in single precision;
- Consider using parallel HDF5, reorganizing the way distributed data (wavefunctions and charge density) is written in order to minimise the number of tasks and the communication load needed to re-distribute read data across tasks, while still keeping portability with respect to the number of MPI tasks.

Improve scalability and performance of CP code. Car-Parrinello (CP) code for large systems with thousands of atoms (see Fig. 7) displays a bottleneck in the orthogonalisation of the electronic wave functions, which was not unexpected given the size of the matrices involved (more than 10000 × 10000). As anticipated in the previous chapter, analysing the code we trace back the cause to a sub-optimal parallelisation of matrix multiplications, with doubly distributed matrices: over the linear algebra *ortho* group, and band group. Moreover we identify a few other places, where atoms (usually less than one thousands in number) were replicated across processors (with few atoms, this is certainly the best strategy, since atoms are not the most relevant data structure in CP simulations), with the computation on ultra-soft pseudo-potential projectors replicated as well. For large systems distributed on tens of thousands of processor, even a small fraction of the computation left undistributed will start to dominate.

Concerning the CP code, for the next months of MAX projects within the activity of WP1, we plan to:

- improve the parallelisation of the *ortho* kernel;
- optimise the computation of ultra-soft pseudo-potential projectors.

5.2 Yambo

I/O efficiency, including parallel I/O. While collecting the benchmark data shown in this report, mostly taken on the KNL partition of Marconi@CINECA, we have observed a sensible degradation of the I/O performance (and in turn of the wall-time) when HDF5-based parallel I/O was in use. Yambo has recently implemented this feature and we still need to investigate extensively the reasons of the performance loss. More generally, the overall efficiency of I/O will be addressed and analysed explicitly.

Distributed data structures. Strictly related to the previous issue is the definition of the parallel layout for distributed data. This is crucial to Yambo, which deals with rather large datasets (think e.g. to the spatial and frequency dependent response function $\chi^0(\mathbf{G}, \mathbf{G}', \omega)$, where the number of $\mathbf{G}$ could be of the order of tens of thousands or more, or the Bethe Salpeter kernel represented on the space of transitions). Several options are currently available in Yambo (stripe distribution for full or triangular matrices or ScaLapack /BLACS grids), mostly dependent on the required processing of the data (including I/O dumping). Accurate evaluation of the efficiency and scalability of the conversion from one distribution to another is needed.

MPI load imbalance. The manybody methods implemented by Yambo are usually simple to parallelise in terms of computation (in view of the large amount of computation to be performed), while tend to suffer because of load imbalance. At the moment the scheduling of MPI tasks is static, while the one of OpenMP threads, most of the time working at the same level, is dynamical. If needed, more advances strategies to address load balancing (eg a master-slave approach) will be considered and evaluated.

Alternative approaches for portability on accelerated machines. So far, Yambo portability on GPU-accelerated machines has been based on the use of CUDA Fortran. While being very numerically efficient and easy to adopt, this is limited to NVIDIA GPUs and is a proprietary extension of Fortran (currently implemented only by the PGI compiler). In the scenario, where multiple accelerator HWs and multiple SW stacks are going to appear on the market (besides NVIDIA and AMD, accelerated HW is expected also from Intel), we plan to implement and evaluate alternative porting strategies. Among these, directive-based porting (OpenACC and OpenMP 5.0) and/or explicit CUDA support.

Memory footprint on GPU-accelerated machines. Present results for the numerical and parallel performance of Yambo ported on GPUs are very encouraging. Nevertheless, one aspect that can be threatening to Yambo use cases is the control of memory footprint, especially in situations where one MPI task per GPU is used, i.e. when data distribution can be limited. A detailed investigation of Yambo memory usage on GPUs, and more generally on accelerators, is currently of uttermost importance.

5.3 FLEUR

Scalability of the code. After the fundamental data restucturing of the matrix setup part, the detailed investigation of the performance and scalability should be repeated. We do not expect the appearance of the load unbalance in the matrix setup part, but this assumption needs to be verified. The performance of this part will be studied in details, in particular, algorithmically different constituents (e.g. spherical and non-spherical matrix set up) will be investigated separately in order to identify the promising optimization strategies. We also plan to improve the load balance and hence the scalability of the new charge part. An additional issue is a tendency that every newer supercomputer architecture has a lower machine balance (i.e. the relation of the memory bandwidth to the peak performance) than a previous one (e.g. the machine balance of the CLAIX 2018 cluster, Intel Skylake, is more than twice less than that of the CLAIX 2016, Intel Broadwell), which makes codes even more memory-bound.

Possible communication bottlenecks with large benchmarks. The STO system, our so far biggest benchmark, has a special geometry so that several k-points need to be calculated. Although the almost ideal scaling over many k-point was many times seen with smaller unit sets, it should be verified for simulations with such a large unit cell. The necessity to communicate eigenvectors of several large dense complex matrices (340k X 340k in this case, 2 TBytes each) could lead to a bottleneck which would then need to be eliminated.

Performance fluctuations. While performing the calculations with large (>1000 atoms) benchmarks, we observed large fluctuations of the performance for the computationally intensive code parts which lead to the increase of the wallclock execution time up to three times. Performance counter measurements have shown that in these cases the CPU frequencies of some nodes were dropped down to the half of their usual values. We never observed such behaviour for the smaller benchmarks and it should be investigated further.

Offloading onto GPU. Matrices set up and diagonalisation are the two most computationally intensive parts of the code and are independent for different k-points. This and the fact that only 15% of the eigenvectors are needed makes them ideal candidates for the offloading onto an accelerator.

5.4 BigDFT

AiiDA **plugin.** To perform these benchmarks, an AiiDA plugin has been developed for BigDFT, allowing for a seamless integration of AiiDA in BigDFT's notebook workflows. This work will be stabilized and released in the next weeks.

Performance prediction. For large-scale calculations, the number of node hours which are needed to perform a benchmark may easily become important. Users and, more importantly, developers, may thus need a large amount of computational resources to identify bottlenecks and verify that the proposed solutions provide a relevant improvement in the performances. To obtain good performances on such use-cases, it is common to try several algorithmic alternatives, to relax synchronizations as much as possible to allow the applications to opportunistically use resources, or to rely on automatic scheduling and load-balancing techniques. The efficiency of such approaches is difficult to assess and highly depends on the target system. Furthermore, the corresponding implementations are quite complex and thus quite error-prone. It is not uncommon in dynamic and highly optimized approaches to obtain efficient implementations that suffer from rare non-deterministic deadlocks or failstop errors which are extremely difficult to narrow and debug. Another argument against direct experiments is that they provide only limited experimental control, hindering the reproducibility of experiments.

Hence, both the performance and the correctness of these systems must be considered. This induces very different techniques: most performance evaluations reason about representative or average executions, while correctness must be evaluated over all possible executions. Performance assessment through pure theoretical analysis often mandates stringent and ultimately unrealistic assumptions regarding the underlying platform and/or the application, which is why most research results come from empirical experiments. Virtualisation and emulation techniques can be used to run real applications on virtual machines (similarly to in-vitro experiments elsewhere). This approach greatly increases the ability to control experiments, but the process is technically very demanding, leading to increased costs in terms of time and labor. Also, the systems built to this end are typically so complex that it is difficult to assess their correctness.

Performance simulation is an appealing alternative to study such systems. It consists in the prediction of the system's evolution through numerical and algorithmic models. The SimGrid framework [21] constitutes a notable framework to perform such experiments. This versatile scientific instrument has been used for simulation studies in Grid Computing, Cloud Computing, HPC, Volunteer Computing and P2P Systems. It has also been shown to be both more realistic and more scalable than its major competitors, thus lowering the boundaries between research domains.

BigDFT has already been partly ported on top of SimGrid and very promising simulation results have been obtained [22]. In the future, we plan to include in our benchmark calculations simulation of the performances of the same bench datasets that are considered in the MAX repository. If this techniques proves successful, it would not represent a simple technical improvement, but a radical epistemic shift; the developer will be able to test and validate the performances of a run at the exascale level without the need for a huge amount of computational resources for the validation of the approach.

Libconv library. The main bottleneck identified in BigDFT is computation costs, and for non-hybrid functionals, mainly convolutions. Libconv is an auto-tuned, BOAST-based, convolution library meant to replace most of the BigDFT convolution kernels and provide optimized versions for each architecture and parameter value. During generation of the library, each kernel is generated for each set of parameters dozens of times with various optimizations applied, and benchmarking is performed on each one. The best kernels are then selected and used in the resulting library, with brokers generated to automatically select the best version for every call. A generic interface has been designed to facilitate integration of the library in DFT codes, with BigDFT as the first user.

5.5 CP2K

Interface with COSMA library. Preliminary results of the communication optimal algorithm for matrix-matrix multiplication implemented in COSMA library shows a good performance in the standalone PDGEMM tests. The work will be continued in the following directions:

- Use the PDGEMM wrapper for COSMA inside CP2K and test the full application in the production runs
- Port COSMA to AMD/ROCm
- Performance optimisation of internal COSMA functions related to the parallel and local data remapping
- Interface COSMA with SIRIUS library

DBCSR library. DBCSR library already has the highly-optimised Intel CPU and NVIDIA GPU back ends. The work will be continued to create the AMD/ROCm and forthcoming EuroHPC architecture back ends. We also investigate the possibility of transfer learning to enable the optimal generation of GPU kernel parameters on the new architectures without running the auto-tuning and machine learning on them and using only the knowledge of the current GPU architecture for which DBCSR is optimised.

Interface with SIRIUS library. SIRIUS is already ported to NVIDIA GPU architecture and can take the full advantage of the GPU nodes. The work will be continued to improve performance of the SIRIUS library, in particular, its internal FFT3D implementation will be replaced by a newly developed SpFFT library. Another action that we plan to accomplish in this project is a port of SIRIUS to the AMD/ROCm architecture.

5.6 SIESTA

From our analysis of this scientific use case, it follows that the most relevant bottlenecks of the “core” part of SIESTA (that devoted to setting up the Hamiltonian) have been resolved on a pure MPI level. Further improvements in efficiency should come from hybrid MPI/OpenMP parallelization. SIESTA contains some OpenMP parallelism, but it is far from being complete. The reason is the presence of heavy data indirection in the code, resulting from indexing of the sparse data structures that are essential to the operation of the code. Rather than working at the loop level, it will be advantageous to explore task-based parallelism, as supported by newer versions of OpenMP. The other major area of focus for further improvement of the performance and scalability of SIESTA is the solvers, which take the Hamiltonian and Overlap matrices and compute the density matrix, which encapsulates the information about the electronic structure. In fact, all the solvers currently in use in SIESTA are provided by external libraries. Ideally, it is the developers of these libraries (and programmers working on their optimization for specific architectures) who should be concerned with portable performance. Client codes like SIESTA, respecting the proper interfaces, should be able to simply link to the appropriate version of the library. For example, the PEXSI library is being continuously improved: a new scheme needing fewer poles was recently introduced; a new level of parallelism in the determination of the chemical potential is now available, and new symbolic factorization code, with better parallel-scaling properties, is in the process of being made the default. SIESTA will directly benefit from these developments.

Some of the solver libraries used in SIESTA fall directly under the purview of MAX: this is the case of the CheSS and O(N)-OMM linear-scaling libraries. They can only be used for systems with a gap separating occupied and unoccupied states, but their favorable scaling makes them quite important. For CheSS, it is planned within WP3 to improve SIESTA's efficiency through the on-the-fly generation of a system-optimized basis set of lower cardinality and hence narrower eigenvalue spectrum, thus reducing the number of polynomial terms needed in the expansion of the Fermi operator. For the O(N)-OMM solver (which was the original solver in SIESTA), the optimization work (also scheduled within WP3) is to leverage the increased performance of the DBCSR sparse-matrix multiplication library, which has been extracted from CP2K and in fact belongs to the MAX family of modules. For the other solvers in use, our task will be to monitor continuously the performance of existing and upcoming versions of the libraries in different regimes (size of the system, number of eigenvectors sought (if relevant)) and, crucially, on different architectures. This is a major effort. As an example, we have started to evaluate the usage of GPUs at the solver stage, relying on libraries providing GPU support for the linear algebra operations used. We did the tests on the CTE-Power block of MareNostrum at the BSC, whose nodes contain 40 IBM Power9 cores and 4 Volta Nvidia GPUs. We tested first the ESSL-CUDA library, not portable to non-IBM systems, but readily available and offering complete code compatibility, as well as "transparent" CPU and GPU interoperability. The initial results of the benchmarks do not show dramatic speedups, and suggest that more control over the data movement might be necessary. The next stage involves using the MAGMA library, which offers more control, as well as being more portable. This kind of profiling work for hybrid architectures and programming models will be an important part of our future actions in WP4. For the reasons of data indirection mentioned above, it is likely that most of the performance gains will come for the solver stage, and less so for the routines involved in the setup of the Hamiltonian during the SCF loop.

6 Conclusion and lessons learned

We decided to base our analysis on scientific cases of interest, rather than directly on application requirements, since this would have implied a deeper understanding of the constraints in the hardware evolution, algorithms, languages and paradigms, which are still a matter of debate even in the HPC field. We have therefore decided to focus the scope of our performance analysis on the identification of bottlenecks that would limit a selected class of scientific cases. Such use cases have been chosen as representative of prototype calculations that may exploit future exascale systems. This approach results to be pragmatic and effective in focusing and directing our effort for codes re-factoring and optimisation.

As a byproduct, the identification, description and recording of several effective scientific cases is a first important result of this work, which has built an extremely valuable repository for the users' community. Along with the use cases we now also have a baseline version of the MAX codes and a quite significant database of performance and profiling results, all stored in the MAX Gitlab repository. All these data will be used by WP4 as well as by all other MAX WPs to asses and validate the WP outcome and progress. It is also a remarkable fact that, even if the use of AiiDA framework to help collecting the benchmark results in a semi-automatic way was foreseen only for later activities, many of the profiling runs were already performed using AiiDA plug-ins. We think that automatising the collection of these data will greatly improve the productivity of HPC experts and developers.

Nevertheless, the lack of resources to run large benchmarks, profiling and debugging campaigns clearly arose as a very critical issue for European HPC as a whole. A single run that lasts, say, 1 hour on 100K compute cores burns 100K CPU/hours, and to stay at the forefront of code development and exploit next generation machines, we need to run many of these large scale test runs. We think, and this is a request to EuroHPC that we will ensure to present in the proper discussion groups, that at lest 5%, (10% would be ideal) of all EuroHPC systems should be dedicated to the above activities, otherwise the availability of European exascale software will be at risk. For this deliverable, we relied mostly on the availability of CPU time guaranteed by HPC centres and MAX developers on their own or national budget. This cannot be taken as granted, and without a committed budget all these activities are at risk. This does not concern only MAX, but rather all CoEs as well as exascale software development in general.

As the main result of this deliverable we report the profiling analysis performed on all MAX codes for the selected scientific use cases, the identification of bottlenecks that should be solved or mitigated along the path towards exascale, and, even if not foreseen at first, some early activities to improve the performance of the codes. The collected database of benchmarks results is then complete and will become the consolidated baseline against which we will assess and compare code refactoring in future MAX activities. Here it is important to remark, as anticipated in the executive summary, that bottlenecks and future improvements are expected to be more and more specific to different classes of use cases. In this conditions it will be much harder for our applications to obtain single improvements effective across all working conditions. MAX applications are in fact frameworks with many capabilities that trigger different internal algorithms and data distributions which may display quite different computational patterns. Nevertheless an improvement, unless for specific cases, it is expected to improve or at least not worsen the performance of the codes under other working conditions. It may also happen that the removal of a deep bottleneck could trigger a complete refactoring of the application, which will then require actions and efforts at the community level even beyond MAX.

As a lesson learned in preparing this deliverable, we observe that when running scientific cases that push both the code and the HPC system to their limits in terms of resources and scalability (inside and outside the node), a lot of problems may occur concerning the stability of the HPC system itself (actually reliability is one of the 10 most important issues spotted out for exascale computing). The profiling tools (usually meant to work under controlled conditions on a fraction of the resources of the system) may fail to save traces and logs or to process them *post mortem*. The codes running under conditions (e.g. thousands of nodes) never tested before may display unexpected or unobserved bugs. Even if this is sometimes frustrating from the side of the developers, it could be extremely helpful to improve the overall robustness of the future exascale European systems and software stacks. Then we figured out, and we propose, to include MAX scientific cases among the codes and datasets to be used in assessing EuroHPC systems and, why not, already in the procurement phase.

References

[1] Giannozzi, P. *et al.* QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials. *Journal of Physics: Condensed Matter* **21**, 395502 (2009).

[2] Quantum espresso vs vasp. URL https://www.nsc.liu.se/~pla/blog/2013/12/18/qevasp-part3/.

[3] Avvisati, G. *et al.* Orbital symmetry driven ferromagnetic and antiferromagnetic coupling of molecular systems. *Nano Lett.* **18**, 2268–2273 (2018).

[4] Atambo, M. *et al.* Electronic and optical properties of doped tio$_2$ by many-body perturbation theory. *Phys. Rev. Mater.* **3**, 045401 (2019).

[5] Denk, R. *et al.* Probing optical excitations in chevron-like armchair graphene nanoribbons. *Nanoscale* **9**, 18326 (2017).

[6] Baroni, S. *et al.* First release (R1) of the MAX codes (2016). https://drive.google.com/file/d/0BzCqlvvD4LOiTGlHOVVZVlZOV0U/view.

[7] Sangalli, D. *et al.* Many-body perturbation theory calculations using the yambo code. *J. Phys: Condens. Matter* **31**, 325902 (2019).

[8] Marini, A., Hogan, C., Grüning, M. & Varsano, D. yambo: An ab initio tool for excited state calculations. *Comput. Phys. Commun.* **180**, 1392 – 1403 (2009).

[9] Denk, R. *et al.* Exciton dominated optical response of ultra-narrow graphene nanoribbons. *Nature Commun.* **5**, 4253 (2014).

[10] Blügel, S. & Bihlmayer, G. Full-potential linearized augmented planewave method. In Grotendorst, J., Blügel, S. & Marx, D. (eds.) *Computational Nanoscience: Do It Yourself! - Lecture Notes*, vol. 31, 85 (NIC Series, 2006). ISBN: 3-00-017350-1.

[11] Alekseeva, U., Michalicek, G., Wortmann, D. & Blügel, S. Hybrid parallelization and performance optimization of the fleur code: New possibilities for all-electron density functional theory. In Aldinucci, M. (ed.) *Euro-Par 2018, LNCS 11014*, 735–748 (Springer International Publishing AG, 2018).

[12] Uranium dioxyde benchmark with bigdft. URL https://gitlab.com/max-centre/benchmarks/blob/master/BigDFT/UO2/bench_marconi_uo2_3/benchmarking-UO2-3.ipynb.

[13] Pizzi, G., Cepellotti, A., Sabatini, R., Marzari, N. & Kozinsky, B. AiiDA: automated interactive infrastructure and database for computational science. *Comp. Mat. Sci.* **111**, 218 – 230 (2016).

[14] COSMA library. URL https://github.com/eth-cscs/COSMA.

[15] DBCSR library. URL https://github.com/cp2k/dbcsr.

[16] SIRIUS library. URL https://github.com/electronic-structure/SIRIUS.

[17] Lin, L., García, A., Huhs, G. & Yang, C. SIESTA-PEXSI: massively parallel method for efficient and accurate ab initio materials simulation without matrix diagonalization. *Journal of Physics: Condensed Matter* **26**, 305503 (2014).

[18] Kresse, G. & Furthmüller, J. Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set. *Phys. Rev. B* **54**, 11169–11186 (1996).

[19] Dai, X., Liu, Z., Zhang, X. & Zhou, A. A parallel orbital-updating based optimization method for electronic structure calculations (2015). arXiv:1510.07230.

[20] Vecharynski, E., Yang, C. & Pask, J. E. A projected preconditioned conjugate gradient algorithm for computing many extreme eigenpairs of a hermitian matrix. *Journal of Computational Physics* **290**, 73 – 89 (2015).

[21] Casanova, H., Legrand, A. & Quinson, M. Simgrid: A generic framework for large-scale distributed experiments. In *Tenth International Conference on Computer Modeling and Simulation (uksim 2008)*, 126–131 (2008).

[22] Bédaride, P. *et al.* Toward better simulation of mpi applications on ethernet/tcp networks. In Jarvis, S. A., Wright, S. A. & Hammond, S. D. (eds.) *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation*, 158–181 (Springer International Publishing, Cham, 2014).