JCTC Journal of Chemical Theory and Computation

pubs.acs.org/JCTC Article

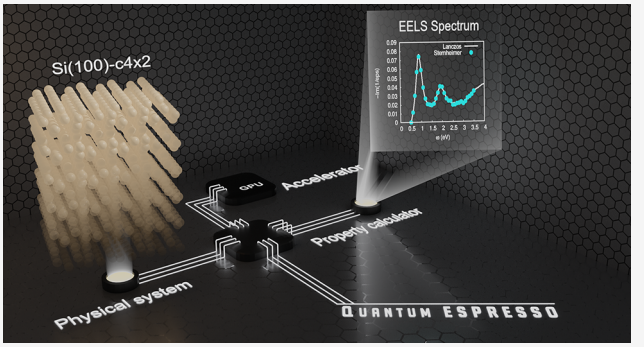
QUANTUM ESPRESSO: One Further Step toward the Exascale

Ivan Carnimeo,* Fabio Affinito, Stefano Baroni, Oscar Baseggio, Laura Bellentani, Riccardo Bertossa, Pietro Davide Delugas, Fabrizio Ferrari Ruffino, Sergio Orlandini, Filippo Spiga, and Paolo Giannozzi

Cite This: https://doi.org/10.1021/acs.jctc.3c00249 Read Online

ACCESS | Metrics & More | Article Recommendations

ABSTRACT: We review the status of the QUANTUM ESPRESSO software suite for electronic-structure calculations based on plane waves, pseudopotentials, and density-functional theory. We highlight the recent developments in the porting to GPUs of the main codes, using an approach based on OpenACC and CUDA FORTRAN offloading. We describe, in particular, the results achieved on linear-response codes, which are one of the distinctive features of the QUANTUM ESPRESSO suite. We also present extensive performance benchmarks on different GPU-accelerated architectures for the main codes of the suite.



1. INTRODUCTION

High-performance computing (HPC) is approaching the *exascale*, that is, 10^{18} floating-point operations per seconds (flops). This means that calculations that took hundreds of hours 30 years ago could now be performed in tens of seconds, at least in principle. HPC has thus become a strategic asset for industrial and technological development of countries. Several exascale and pre-exascale machines are in the preproduction state or already fully operational. For such a class of machines, graphics processing unit (GPU) acceleration has become a *de facto* standard, and almost all of the first five entries in the Top500[1] list of supercomputers are currently based on GPU acceleration.

This context has proved to be particularly fertile for molecular and material sciences that have evolved in parallel with the advances in computer science. Nowadays, most of the main codes for molecular and material modeling are accelerated or are in the process of being ported to accelerated architectures. In this respect, the QUANTUM ESPRESSO software suite[2−5] can boast a long experience: the first accelerated working version dates back to several years ago (2017), and the release qe-6.4 (March 2019) was the first one to be officially distributed by the QUANTUM ESPRESSO FOUNDATION[6] having a GPU counterpart for the most important core functionalities. This first porting phase, covering only the main self-consistent code PWSCF, is described in ref 2.

Since then, a great effort has been devoted to the improvement of the GPU version of PWSCF and to the porting of the other codes of the suite to GPU-accelerated architectures. The latest release of QUANTUM ESPRESSO, namely, qe-7.2, enables GPU execution of the linear-response codes: PHONON,[7,8] turboEELS,[9,10] turboLanczos,[11,12] HP,[13−15] and of the molecular-dynamics code CP.[16]

The aim of this work is thus dual: on the one hand, we want to disclose some important developments done in the GPU porting of QUANTUM ESPRESSO since the first article[2] was published. On the other hand, we also want to provide more detailed information—that is missing in the literature—about performances of the codes of the suite on the current state-of-art HPC supercomputers, highlighting advantages, drawbacks, and the most effective parallelization schemes for GPU execution.

The structure of this paper is as follows: Section 2 describes the main new developments of the codes, together with the general philosophy and the technical approach followed for the porting; Section 3 presents selected benchmark tests on GPU for some of the main codes (PWSCF, PHONON, turboEELS, CP) of the QUANTUM ESPRESSO suite; Section 4 contains our conclusions.

2. CODE DEVELOPMENT

2.1. General Philosophy. Our approach has been developed within the *separation of concerns*[2] philosophy: ideally, method developers in science departments and

Special Issue: Electronic Structure Theory Packages of Today and Tomorrow

Received: March 5, 2023

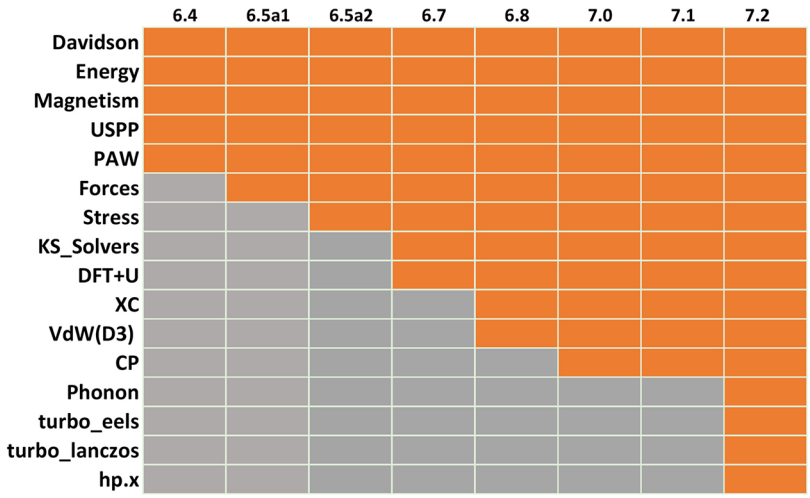


Figure 1. Progression of GPU porting over the different versions of the QUANTUM ESPRESSO suite. In gray and orange are the nonported and ported features, respectively.

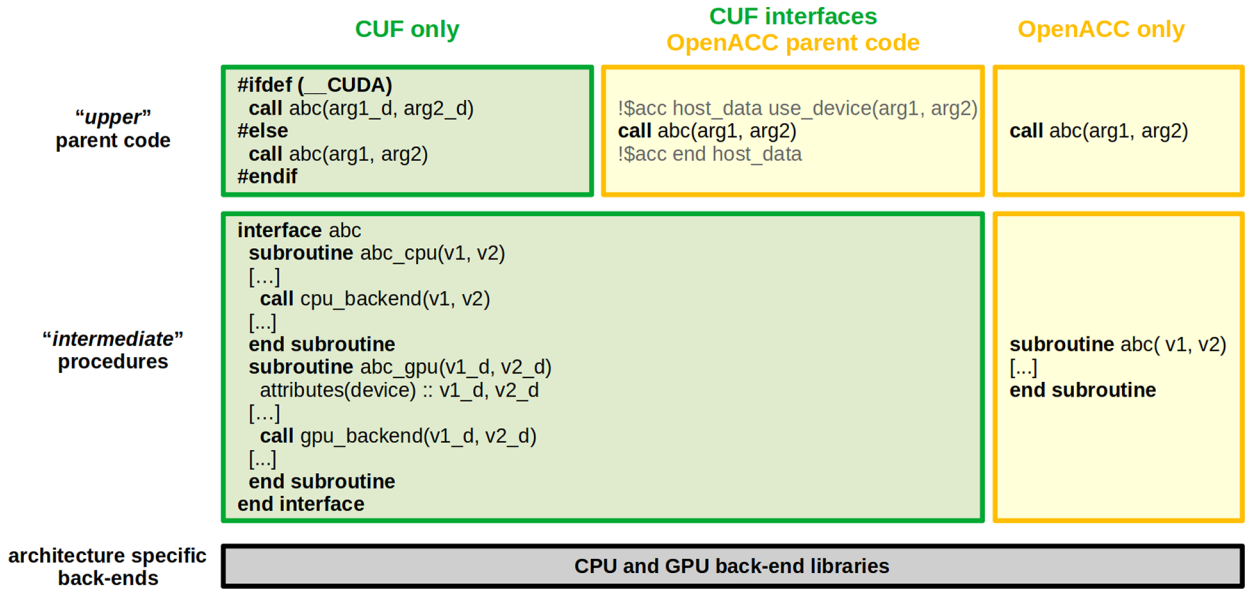


Figure 2. Comparison between CUDA FORTRAN and OpenACC codes.

research laboratories should be concerned with the calculation of physical properties, disregarding architectural details, whereas scientists and research engineers in IT departments and HPC centers should focus on low-level mathematical and system libraries. Separation of concerns is the overarching guideline for the action of the EU MAX Centre of Excellence (CoE) for HPC Applications,[17] whose mission is to foster the porting of important community codes for quantum materials modeling to heterogeneous architectures. The MAX way to separation of concerns is to refactor community codes into a software stack of conceptually distinct—though in practice partially overlapping—components. The core of the code is a *quantum engine* whose main purpose is to perform *Hamiltonian builds*, i.e., the application of the Hamiltonian operator to molecular/Bloch orbitals and related operations, and to solve the quantum-mechanical equations that determine them and their response to external perturbations. The quantum engine is complemented by various *property calculators*, designed to evaluate specific properties and to simulate specific processes of molecular and extended systems. Both the quantum engine and the property calculators leverage a number of modules and mathematical and system libraries. Modules are homogeneous software components that share the same coding style and naming conventions and may share global variables with other modules, with the quantum engine and property calculators. Modules are not designed for extended portability and their adoption in third-party software in general requires the adoption of at least some of the internal data structures of QUANTUM ESPRESSO. Domain-specific mathematical libraries address various general-purpose mathematical operations (e.g., three-dimensional fast Fourier transforms, linear algebra based on both factorization and iterative techniques, minimization and extrapolation to self-consistency, etc.). Ideally, libraries should not rely on any global variables but trade data with the calling program units only through well-designed public application programming interfaces (APIs). Although domain-specific libraries are specialized for and take advantage of the specific features of plane-wave electronic-structure codes, they can be easily adopted by third-party codes without major concerns about their internal data structure. Finally,

https://doi.org/10.1021/acs.jctc.3c00249
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

system libraries perform various low-level tasks, including the offload of data to hardware accelerators, the management of inter- and intranode data communications, etc. System libraries play a key role in our sustainable model of software development and maintenance. By abstracting as much as possible data management and communications, it is possible to maintain a large code base largely independently of the underlying hardware architecture, thus avoiding or dramatically limiting code duplication and freeing the developers of high-level software layers from the need to operate with hardware-specific directives.[2]

2.2. GPU Porting. In Figure 1 a brief summary of the progress of the porting of the QUANTUM ESPRESSO suite over subsequent releases is shown, starting from release qe-6.4 (March, 2019) to the last qe-7.2 version (March 2023).

Earlier versions of the suite were accelerated using a programming model fully based on the CUDA FORTRAN language, that, on the one hand, provided significant speed-ups with respect to the nonaccelerated counterpart but, on the other hand, intrinsically required duplication of code and variables, on host and device sides (see, for example, the pseudocode in Figure 2). As a consequence, for each accelerated portion of the native FORTRAN code, a number of "_gpu.f90" files were created that included GPU counterparts of the original subroutines and modules.

Figure 3 shows a steady increase in the number of lines of "_gpu.f90" files with an increasing number of CUDA FORTRAN

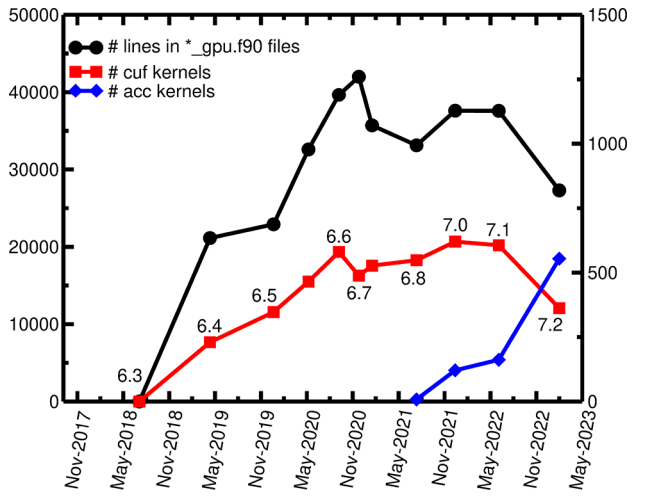


Figure 3. Estimate of CUDA FORTRAN and OpenACC directives in the QUANTUM ESPRESSO codes over the years, where the reference release version has been specified along the red line. Black line refers to the left *y*-axis, red and blue lines refer to the right *y*-axis, and "acc kernels" includes both OpenACC kernels and parallel directives.

("cuf") kernels, until the end of 2020. As the number and size of the "_gpu.f90" files increased, the maintenance burden also increased accordingly, hampering the porting of new features. Furthermore, as mentioned in the previous section, QUANTUM ESPRESSO is a community code where people with different backgrounds are encouraged to contribute. Since developers with no or little experience on GPU programming tend to add features only to the CPU pathway of the codes, the two pathways easily tend to diverge, and it becomes very difficult to keep them aligned in the long term and to identify and fix bugs.

For these reasons, we switched to an alternative porting model, mainly based on OpenACC, which also exploits OpenACC/CUDA FORTRAN interoperability. The main advantage of this approach, as briefly shown in Figure 2, is that in this case host and device copies of the variables are managed by directives and are referenced as the same variable in the parent code, thus allowing us to have a unique source code for both CPU and GPU compilation and execution. In this way, the general structure of the code tends to remain consistent even in the case of features added only on the CPU side. Notably, CUDA FORTRAN has not been completely removed but it has been retained in those cases for which it led to a clear advantage. For example, it can be useful when the differences between CPU and GPU architectures can be better exploited using different algorithms, e.g., in case of large loops batched on the cache size or Fast Fourier Transforms (FFT), where in the CPU execution one band at time is processed, whereas the heavy internal parallelism of GPU encourages us to process many bands at once. Another example is for FORTRAN interfaces (see Figure 2), where the "device" attribute allows us to trigger specific host or device procedures (there is not an equivalent within the OpenACC framework). The latter case is especially helpful when general libraries call system-specific backends of numerical libraries, which are specialized for host or device execution. For example, the *abc* interface referenced in Figure 2 can be representative of FFTXlib or LAXlib, which internally call different host or device specific numerical backends (e.g., cuFFT or cuSOLVER, referenced as *cpu_backend* and *gpu_backend*) to respectively perform FFTs and solve eigenproblems. Also, inside UtilXlib, different MPI backends can be linked. OpenACC/CUDA FORTRAN interoperability has also been exploited to progressively substitute CUDA FORTRAN portions with OpenACC directives, avoiding the rewriting of an entire new code from scratch. After the adoption of this approach in release qe-6.8 (beginning 2021) the number of lines of code contained in the duplicated "_gpu.f90" files dropped down as the number of OpenACC directives increased, even with an increased number of ported features (compare Figures 3 and 1). For example, in the latest qe-7.2 release, the linear-response codes (PHONON, turboEELS) are ported to GPU, but the number of lines in the "_gpu.f90" files decreased by more than 10000 with respect to the previous qe-7.1 release.

After this refactoring, the code is more clearly separated into conceptually different layers with different scopes. In Figure 2 an "upper" layer of code can be schematically identified, which contains most of the physics of the calculation, is strongly based on OpenACC directives, and is quite agnostic of the underlying computational architecture. Developers with no or little experience in GPU programming can easily continue to work here. A lower layer of code (represented by the *abc* interface and subroutine) is based on more specific procedures that specialize the computation for CPU and/or GPU architectures. These procedures can be easily called in a simple way from the parent code. Internally, these procedures further specialize execution, calling numerical specific libraries (represented by *cpu_backend* and *gpu_backend*) that are used as the final backends of the computation and whose development is the domain of computer scientists and deeply specialized programmers. This is in line with the aforementioned separation of concerns philosophy,[2,17] described in the previous section.

2.3. Eigensolver. The default GPU implementation[2] of the eigenvalue solver is based on the cuSOLVER library: real and complex Hamiltonian matrices are diagonalized with the

https://doi.org/10.1021/acs.jctc.3c00249
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

generalized eigenvalue problems using a single GPU. Although very fast, this approach is limited by the available GPU memory, becoming problematic for very large matrices (e.g., dimension beyond ∼7000 on a V100 NVIDIA GPU with 16GB VRAM). Moreover, the computational cost of a dense eigenproblem solution scales as $O(N^3)$ and may become sizable for large matrices.

A recently implemented alternative approach (not yet available in the production version) of a parallel dense-matrix eigensolver on GPU is based on the Eigenvalue soLvers for Petascale Applications (ELPA)[18] library. ELPA implements two types of distributed eigensolvers: one-stage (ELPA1) and two-stage (ELPA2) diagonalization methods. The latter algorithm is the most suitable for distributed-memory architectures because it scales more efficiently, as global communications are not required, and it has a faster computational part.

The ELPA library relies on the matrix layout defined by the ScaLAPACK library for parallel linear algebra; thus, it can be used as a drop-in improvement in ScaLAPACK-based applications like Quantum ESPRESSO.

2.4. Exchange−Correlation Library. Starting from the qe-6.8 version, the code for computing exchange−correlation (XC) kernels of Quantum ESPRESSO was encapsulated into an independent library that also supports the usage of other external XC libraries in a flexible way. The aim of this refactoring was twofold: to ease the maintenance and the development (notably, the addition of new functionals) of the XC code and to seamlessly integrate the functionalities of the popular Libxc library[19] into Quantum ESPRESSO. The XC library can be used by other electronic-structure softwares as well.

The library covers the local-density (LDA), generalized-gradient (GGA), and meta-GGA families of XC functionals. It is interfaced to Quantum ESPRESSO through a set of wrapper routines, which call either the internally provided functionals or the ones from Libxc, depending on the input choice. Therefore, the library consists of a few main routines that provide the energy, potential, and potential derivatives on the density grid and a number of initialization and setting routines that manage additional dependencies. The library allows any combination of internal and external (Libxc) functional forms.

The computational cost of the XC library is typically a small fraction of the total. Nonetheless a GPU porting allows one to avoid data movement and to significantly improve the performance in the exchange−correlation potential calculation. The simplicity of the driving algorithm: one main loop running over the density grid, where the routines computing the functional are called at each point, ensures optimal speed-up with little intervention (a few OpenACC directives) on the code. The input density (and possibly its derivatives) and the output energy and potential arrays of the main XC routines are assumed to be present on device memory, depending upon the value of a logical optional variable. If the latter is false (or omitted), then the offload is done internally to the library so that the developer is not forced to care about the offloading.

2.5. Hybrid Functionals. The computation of the Fock operator for hybrid functionals[20−22] is still a hard task for plane-wave-based codes. In this respect, a couple of major advances have been included in the Quantum ESPRESSO suite in the last years. In 2017, a new scheme for the parallel computation of exact exchange based on a band-pair parallelization approach was proposed by Barnes et al.[23] and integrated in the Quantum ESPRESSO suite. More or less in the same period, Lin developed a new method based on an Adaptively Compressed Exchange (ACE) operator[24,25] that allowed us to tear down the computational time of the self-consistent field (SCF) step with no loss of accuracy. Such a method was then implemented[26] in the Quantum ESPRESSO suite with some minor modifications, together with a variant that exploits orbital localization[27,28] to further reduce the computational burden. Benchmark tests showed a dramatic decrease in computational time to solution with respect to previous implementations. On top of these methodological developments, the entire exact-exchange code has been ported to GPU.[2]

Recently, the implementation of hybrid functionals in Quantum ESPRESSO has been improved with some minor changes. Every time the ACE projector is updated during an SCF calculation, it is also written on disk, in the same format used for wave functions I/O, so that it can be read to speed-up subsequent runs. The overall amount of memory and I/O bandwidth required for storing the ACE projector is quite relevant but generally affordable, as it is comparable to the memory needed for storing the wave functions. This feature is particularly useful in case one large calculation with an hybrid functional is stopped or crashes before reaching convergence, for example, due to wall time limit policies of the queue systems of HPC centers. In this case, the calculation can be recovered by reading the ACE projector and the wave functions from disk and restarting the calculation from the last (previously stopped or crashed) outer iteration, skipping the first exact exchange calculation of the restart.

The feature mentioned above is also very useful for band-structure calculations. In this case, one usually needs to perform a very heavy SCF calculation on a large and dense uniform (Monkhorst−Pack[29]) grid, including many virtual orbitals to compute unoccupied band energies. Then one resorts to some interpolation scheme, e.g., using Wannier functions,[30−32] to compute the band structure for the desired **k**-points. With the new feature, the heavy SCF can be split into two steps: a cheaper SCF run on occupied bands only and a subsequent non-SCF run including also virtual orbitals, thus avoiding the evaluation of the ACE operator on the virtual manifold at each SCF outer iteration. Noteworthy, since the virtual orbitals are not included in the first SCF step, in the non-SCF procedure the ACE potential is first read from file and used as is for a first diagonalization; then it is updated with the new virtual orbitals and a second diagonalization is performed to get correct virtual band energies. Currently, it is not yet possible to perform the non-SCF calculation on a set of **k**-points different from the one used in the previous SCF run.

As a last remark, a computationally inexpensive method[33,34] has been recently implemented in order to interpolate band structures directly from SCF or non-SCF calculations on uniform Monkhorst−Pack grids. This method is based on a fitting algorithm that minimizes a roughness function across the entire Brillouin zone and can be used as a quick alternative to using the Wannier functions.[35−37]

2.6. Parallelism and Data Distribution. Different levels of parallelism, based on both the message-passing interface (MPI) library and on multithreading, are currently implemented in the Quantum ESPRESSO suite. They can be flexibly combined together on the basis of the particular molecular system under study and the hardware architecture. In the following, the different parallelization schemes are

https://doi.org/10.1021/acs.jctc.3c00249
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

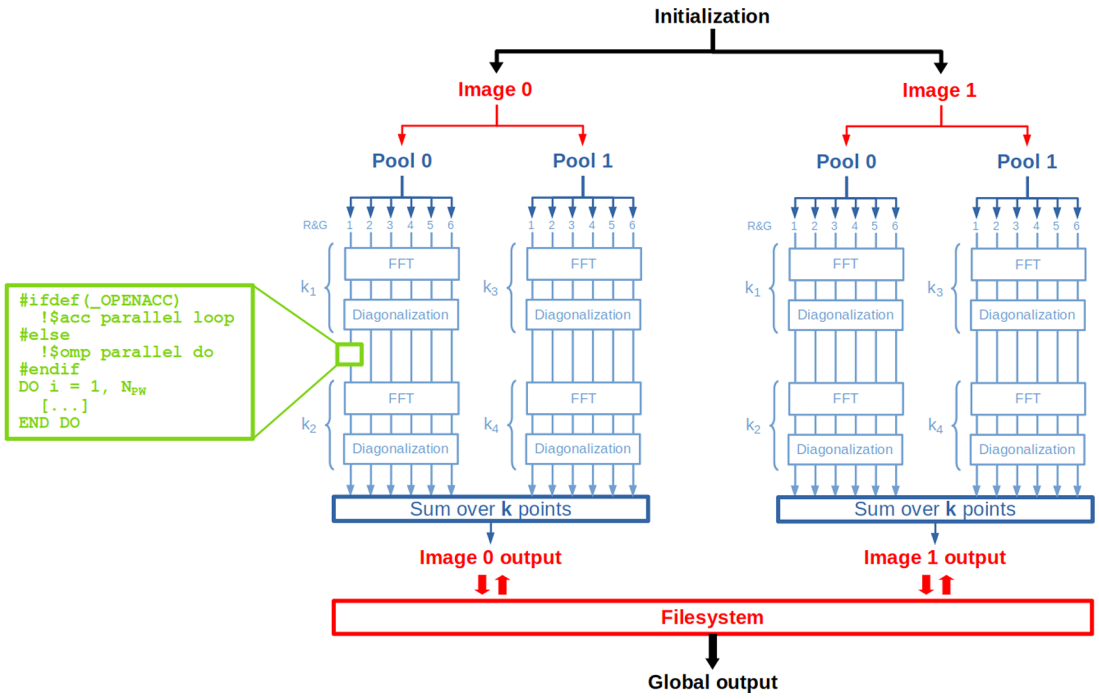


Figure 4. Sample execution flow with 2 *images* and four **k**-points, distributed in 2 *pools* and 24 total ranks. Global operations are represented in black, operations involving *images*, *pools*, and *R&G* groups are represented in red, dark blue, and light blue, respectively, and GPU acceleration and OpenMP multithreading are in the green box. Closed boxes represent operations that require communications among *images* (Filesystem access), *pools* (sum over **k**-points), and *R&G* groups (FFT, diagonalization).

briefly summarized with a focus on accelerated architectures, in order to facilitate the discussion of the performance on GPU in Section 3. More details about parallelism in QUANTUM ESPRESSO can be found in the original works[2−5] and other more tailored publications.[38]

The diagram in Figure 4 shows in a simplified way how the main types of parallelism implemented in the QUANTUM ESPRESSO codes are interconnected with each other in a prototypical execution. The outermost level of parallelism is the *image* parallelism, whereby different MPI ranks run different instances of a given calculation, and the communications are mostly performed through filesystem I/O. This scheme is very useful for algorithms that expose a natural parallelism based on repeated clearly distinguishable tasks that need to communicate with each other only to some very small extent. This is the case, for example, of Nudged Elastic Band (NEB)[39−41] calculations, where each *image* computes an independent geometry along the transition pathway. Geometries are connected by springs, and each *image* has to communicate only forces to the others. Another example is phonon calculations, where different *images* can concurrently solve different sets of Sternheimer equations[42] for different irreducible representations of nuclear displacements and/or for different wavevectors **q**.

Other levels of parallelism are *pools*, *band groups*, and *plane-wave* schemes (the latter also often referred to as *R&G*), that hierarchically distribute memory and computational load of those parts of the code that depend on the number of **k**-points (N_k), bands (N_b) and plane waves (N_{PW}), respectively.

Pool parallelism can be used for all systems described by more than one **k**-point and/or nondegenerate spin channels. In this case, the total number of MPI ranks available for the calculation, for example, N_{tot}, is divided into N_{pools} groups (*pools*) of N_{tot}/N_{pools} ranks each. N_{pools} distinct *R&G* data structures are allocated and distributed among their respective *pools*' groups and operate over separated sets of preassigned **k**-points. The efficiency of this type of parallelism—which is similarly exploited in many quantum-chemistry codes—relies on the fact that the communications among the *pools* are limited because the code performs most of the operations autonomously within each eigenspace of the translational symmetry group. For example, in Figure 4 communications among *pools* are graphically represented by summations over **k**-points only at the end of the execution and are not required by FFT and diagonalization steps. A typical example in the QUANTUM ESPRESSO suite is the Kohn-Sham solver (KS_Solver) library, where the Kohn−Sham equations are solved independently for each **k**-point and most of the communications among *pools* are needed only once the Kohn−Sham orbitals are found, to obtain the total electronic energy and density. In this respect, recalling the example in Figure 4, one particularly efficient setup is when all ranks belonging to the same *pool* are executed on the same node (e.g., one *pool* per node, two *pools* per node, etc.), because internode communications are avoided in FFT and diagonalization steps. Furthermore, concerning GPU calculations, since each GPU is internally massively parallel (e.g., the V100 architecture has 5120 CUDA cores, for an overall peak double-precision performance of 7.8 TFLops), in many cases it is feasible to move the whole *R&G* parallelism within one single GPU, removing most of the communication overhead and yielding a significant performance boost. In Figure 4, it is evident that using one rank per *pool* drastically reduces also intranode communications in FFT and diagonalization steps.

Band group parallelism works on conceptually similar grounds to the *pool* parallelism, also for systems where the latter cannot be used. The total number of ranks N_{tot}, or the total number of N_{tot}/N_{pools} ranks inside each *pool*, is further

https://doi.org/10.1021/acs.jctc.3c00249
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

Table 1. System Specification for the GPU (Marconi100 and Selene) and CPU (Galileo100) Partitions

Cluster	Galileo100	Marconi100	Selene
Centre	CINECA	CINECA	NVIDIA Corp
Model	Dual-Socket Dell PowerEdge	IBM Power AC922 (Whiterspoon)	NVIDIA DGX SuperPOD
Nodes	636	980	560
Processors (per node)	2 × 24 cores Intel Xeon Platinum 8260 @ 2.4 GHz	2 × 16 cores IBM POWER9 AC922 @ 2.6(3.1) GHz	2 × 64 cores AMD EPYC 7742 @ 2.25(3.4) GHz
GPUs (per node)		4 × NVIDIA Volta V100 SXM2 (2 pairs), 16 GB HBM2	8 × NVIDIA Ampere A100 SXM4 (with NVSwitch), 80 GB HBM2e
Cores	48 cores/node	32 cores/node, Hyperthreading x4	128 cores/node
RAM	384 GB/node	256 GB/node	1 TB/node
Node Performance (peak)	3.53 TFLOPS	32 TFLOPS	82.2 TFLOPS
Network Cards	Mellanox Infiniband 100GbE	1 × Mellanox ConnectX-4 EDR (100 Gb/s)	8 × Mellanox ConnectX-6 HDR (200 Gb/s)
Network Topology	Full Fat-tree	DragonFly++	Full Fat-tree

divided into N_{bg} *band groups*, each composed of $N_{tot}/N_{pools}/N_{bg}$ MPI ranks.

Finally, the ranks can also be used to distribute the data structures that depend on the number of plane waves (*R&G*). This latter parallelization is very effective in particular from the point of view of memory requirements, as it allows us to distribute memory along the largest dimension of the data structures. For example, in Figure 4, the plane-waves of each *pool* are distributed among 6 ranks.

As a general remark that will be often relevant for the following benchmarks, increasing the number of *pools* when possible is beneficial for performance but requires more memory, whereas decreasing the number of *pools* and increasing the dimension of each *pool* group with *R&G* parallelism is beneficial for memory consumption.

Another important level of parallelism, compatible with all the aforementioned ones, is related to the most important linear algebra operations (most noticeably diagonalization and matrix−matrix product) that are performed using distributed linear algebra libraries, e.g., SCALAPACK, ELPA.

Finally, OpenMP multithreading (OMP) can be used to accelerate loops at the finest level, inside each rank (see the green box in Figure 4). In case of CPU-only calculations, OMP threads can be effectively exploited especially when many nodes are needed for memory reasons, but deploying all the available CPU cores with MPI ranks leads to communication overheads. The optimal number of OMP threads can vary depending on the particular molecular system size and node configuration. From our experience based on the current HPC architectures, a good choice is usually to associate 2 to 8 OMP-threads per MPI rank.

In the case of GPU-accelerated calculations, a good practice is to bind the number of MPI ranks to the number of available GPUs, so as to work with one MPI rank per GPU. Other choices, involving GPU oversubscription, are usually less convenient. In this case, since usually HPC architectures are provided with more CPU cores than GPUs per node, OMP threads can be used to deploy all of the remaining CPU cores not deployed with MPI ranks. The effect on performance is however small if the GPU porting is effective, as usually only a small fraction of the computational workload is executed on the CPU.

3. CODE PERFORMANCE ON SELECTED BENCHMARK TESTS

In this section we report some results on computational performance of the main codes of the QUANTUM ESPRESSO suite on GPU-accelerated machines, in terms of times, speed-up, and parallel efficiency.

Reference "CPU" and "GPU" calculations have been performed respectively on Galileo100 and Marconi100 clusters at CINECA, whereas for some selected very large systems we also used a third cluster, Selene, made of NVIDIA DGX A100 nodes, each with 8 GPUs. Table 1 briefly summarizes the main technical features of the three clusters.

The speed-up S for an arbitrary number of ranks n, with respect to a reference minimum number of ranks n^{min}, is defined as

$$S(n) = \frac{t(n^{min})}{t(n)} \tag{1}$$

where $t(n)$ and $t(n^{min})$ are wall times, whereas the parallel efficiency ε is

$$\varepsilon(n) = \frac{t(n^{min})}{t(n)} \cdot \frac{n^{min}}{n} \tag{2}$$

The latter is a number in the interval $[0,1]$, where $\varepsilon = 1$ is the ideal efficiency corresponding to a perfect speed-up in which $t(n)$ is exactly $t(n^{min})/n$.

3.1. Ground State Energy Calculations for Large Systems. In order to show the performance of the core code of the QUANTUM ESPRESSO suite, PWSCF, we present here calculations on a large orthorhombic supercell of chromium iodide (CrI_3) bulk, with 1152 atoms, 7776 electrons, and cell parameters a = 22.48 au, b/a = 1.67, and c/a = 13.86. This calculation is part of a larger study of edge magnons,[43−45] where interfaces between different crystalline phases are tested within this large supercell. The LSDA method with collinear spin polarization has been used, in combination with norm-conserving pseudopotentials generated with the atomic code,[46] with plane-wave kinetic energy cutoff of 60 Ry. A Gaussian smearing with broadening of 0.01 eV was also used, resulting in a total number of 4666 Kohn−Sham states. A uniform Monkhorst−Pack[29] grid of 4 × 1 × 1 **k**-points has been used, resulting into 3 **k**-points per spin component.

In Figure 5 the time per iteration of CPU execution on Galileo100 and GPU execution on Marconi100 and Selene supercomputers is reported. In each case we picked the best

https://doi.org/10.1021/acs.jctc.3c00249
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

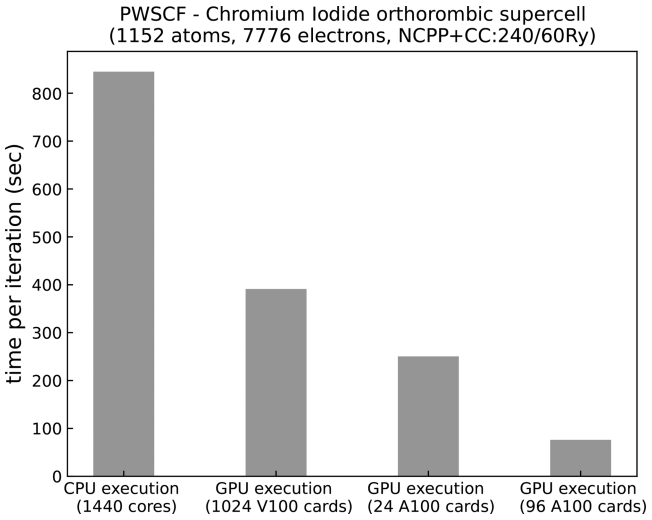


Figure 5. Time per SCF iteration of a chromium iodide (CrI_3) orthorhombic supercell (a = 11.89 Å, b = 19.81 Å, c = 164.81 Å), norm-conserving pseudopotential with core correction, cutoffs 60 and 240 Ry for wave functions and density, respectively. Each SCF iteration corresponds to a Davidson diagonalization.

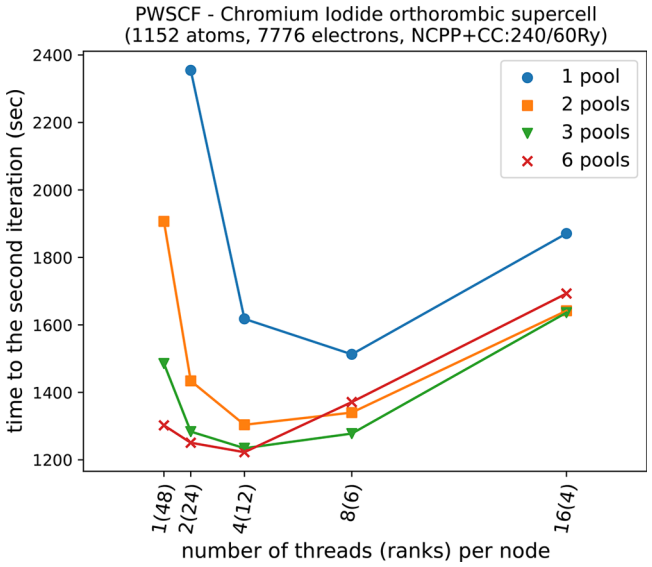


Figure 6. Time to the second SCF iteration (initialization + first iteration) of a chromium iodide (CrI_3) orthorhombic supercell (a = 11.89 Å, b = 19.81 Å, c = 164.81 Å), norm-conserving pseudopotential with core correction, cutoffs 60 and 240 Ry for wave functions and density, respectively. Different numbers of *pools* and different threads/ranks combinations are compared at a fixed number of 1440 total cores (30 nodes).

results in terms of balance between performance and memory consumption, obtained after running several calculations with different computational configurations. For CPU calculations on Galileo100, the setup with 30 nodes (1440 cores) allows an equally distributed workload of 3 **k**-points and two spin channels among 2, 3, or 6 *pools*. Using fewer nodes, either performance was inferior or there was not enough memory to fully test *pool* parallelism. On the other side, calculations with 32 nodes were affected by the maximum FFT dimension: there are 1536 FFT grid points along the longest cell dimension (c = 164.81 Å) and exactly 1536 cores, leading to too few plane waves per rank.

We also tested different combinations of *pools*, MPI ranks, and OMP threads, reported in Figure 6. Increasing the number of *pools* markedly improves computational times because calculations for different **k**-points are decoupled (cf. also Figure 4). When using extreme MPI parallelizations (1 thread per node, i.e., 1440 ranks), communications among ranks become predominant, whereas using too many threads leads to too small *pool* groups, and loops over **G**-vectors become more demanding. These two competitive trends lead to the nonmonotonic curves with minima shown in Figure 6, also often found in other benchmarks of the QUANTUM ESPRESSO codes (as for example here[47]). Optimal combinations found here are with 2 or 4 threads and 3 or 6 *pools*. Data in Figure 5 refer to 2 OMP threads per rank and 6 *pools* of 5 nodes (120 ranks) each.

On Marconi100, calculations with 16, 32, 64, 80, and 128 nodes were limited by memory requirements even with only one *pool* (16 GB is the maximum available for V100 GPU), and the time reported in Figure 5 refers to 256 nodes with *R&G* parallelism only. Eight OMP threads per rank have been used to fully deploy the CPU cores of the nodes. Despite the limitations, running on V100 GPU cards is significantly faster than running on CPU leveraging *pool* parallelism.

The benchmark executed on Selene overcomes the memory limitations of the V100 cards thanks to 80 GB of GPU memory. In fact, bigger GPU memory capacity allows us to run the full calculation with only 24 GPUs (with 16 OMP threads, no *pools*), showing an overall speedup with respect to the reference CPU execution of about 4×. It is also possible to fully exploit *pool* parallelism by increasing the number of nodes to avoid memory capacity constraints, further improving performance. For example running on 96 A100 GPUs with 3 *pools*, the time per iteration drops down from 250 to 76 s, resulting in a speed-up of about 10× with respect to the CPU reference calculation and a dramatically smaller amount of resources employed.

Generally speaking, for PWSCF and most other codes in the QUANTUM ESPRESSO suite, GPU memory is a quite crucial parameter. A large GPU memory allows both reduction of the total number of ranks employed and an increase of the number of *pools*, ultimately reducing communications and host−device synchronizations. On GPU architectures with limited memory, instead, calculations for large molecular systems may require an exceedingly large number of nodes.

3.2. Phonons and Vibrational Properties via DFPT Methods. The entire PHONON code[7,8] has been accelerated using the approach described in Section 2, based on OpenACC and relying on CUDA FORTRAN modules and libraries inherited from PWSCF.

In Figure 7 we show the phonon dispersions of the 100 surface of silicon with c(4 × 2) reconstruction (reported among the most stable ones by a number of previous studies[48,49]), simulated using a base centered orthorhombic primitive cell ($a = 2b$).[48,50] The first Brillouin Zone of the slab has been sampled with a 8 × 8 × 1 uniform Monkhorst−Pack mesh[29] (21 total symmetry inequivalent **k**-points), whereas two different depths have been considered in the third direction, one with 16 layers of silicon atoms (referred to as Si(100)-16L hereafter, with 64 atoms per cell) and one with 32 layers (Si(100)-32L, 128 atoms per cell).

The phonon dispersions have been obtained using a standard procedure based on Fourier interpolation technique, evaluating the dynamical matrices in reciprocal space on a

https://doi.org/10.1021/acs.jctc.3c00249
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

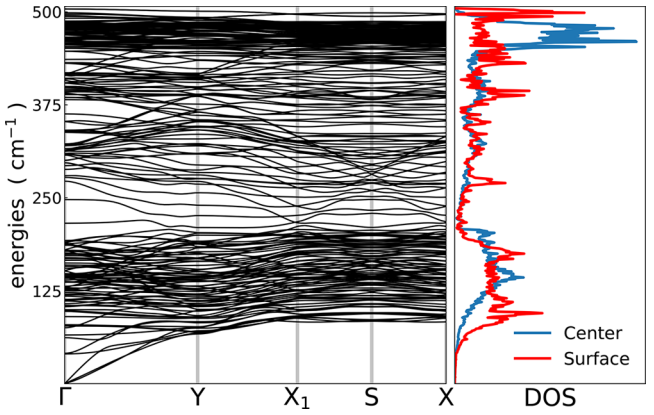
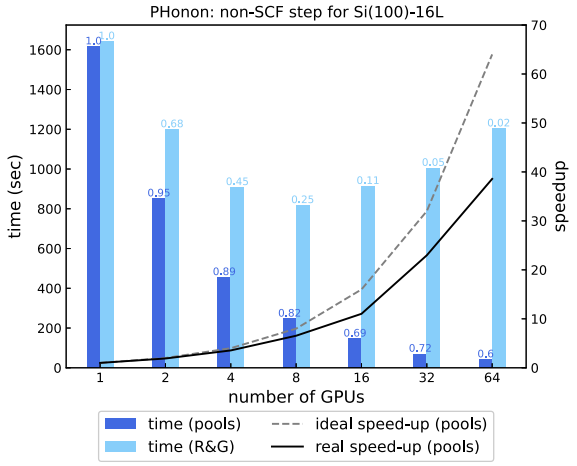


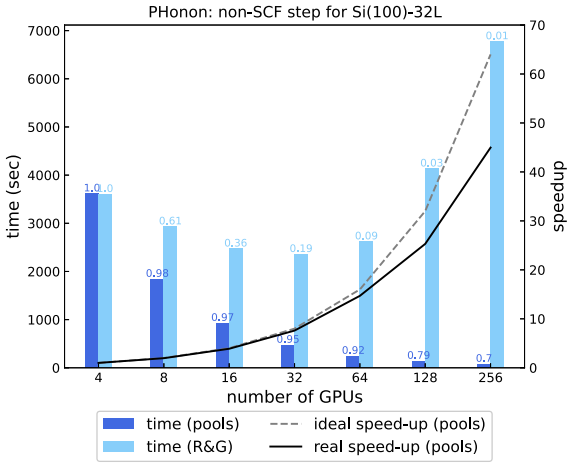
Figure 7. Phonon dispersion and projected density of states of the silicon (100) surface simulated using the Si(100)-16L model.

uniform 8 × 8 × 1 grid of **q**-points, and then interpolating along the path Γ-Y-X$_1$-S-X, used in another previous work.[51] PBE[52] functional and norm-conserving pseudopotentials[53] with a plane-wave kinetic-energy cutoff of 60 Ry have been employed. We observe that the depth of the slab is large enough to recover the periodic structure of the bulk phonons, resulting in a realistic description of the physical surface of the material, even with the smaller model with 16 layers.

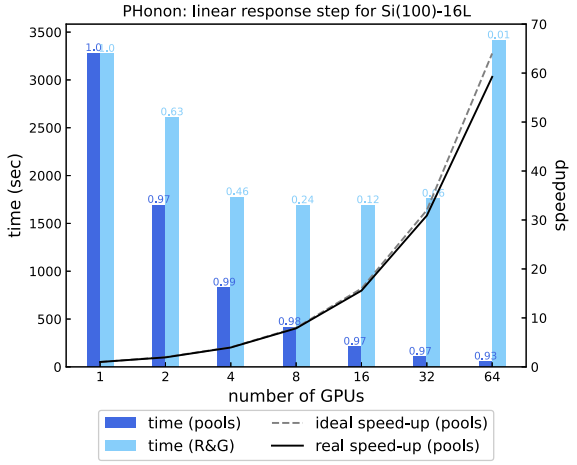
The computationally most intensive part of the dispersion calculation is the evaluation of the dynamical matrices in reciprocal space for all of the necessary **q**-points. For each **q**-point, orbitals and band energies at all (**k** + **q**)-points need to be computed with a non-SCF calculation, and a set of Sternheimer equations[42] has to be solved for each symmetry-inequivalent perturbation. In the present model systems, the perturbations are 192 and 381 irreducible representations of nuclear displacements (*irreps* hereafter), for Si(100)-16L and



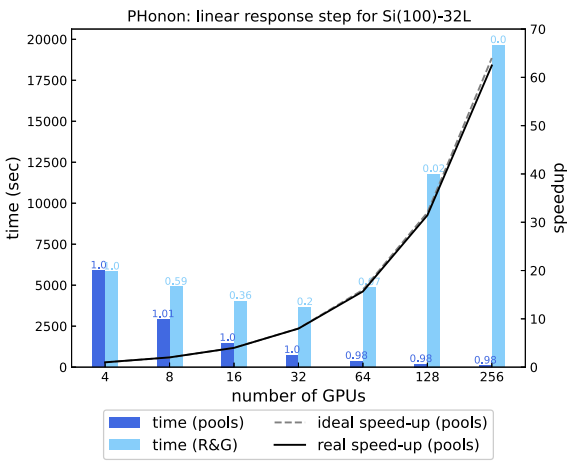
(a) Si(100)-16L, non-SCF step, scaling over *pools* with one *pool* per GPU, scaling over plane waves at fixed one *pool*



(b) Si(100)-16L, non-SCF step, scaling over *pools* with one *pool* per GPU, scaling over plane waves at fixed one *pool*



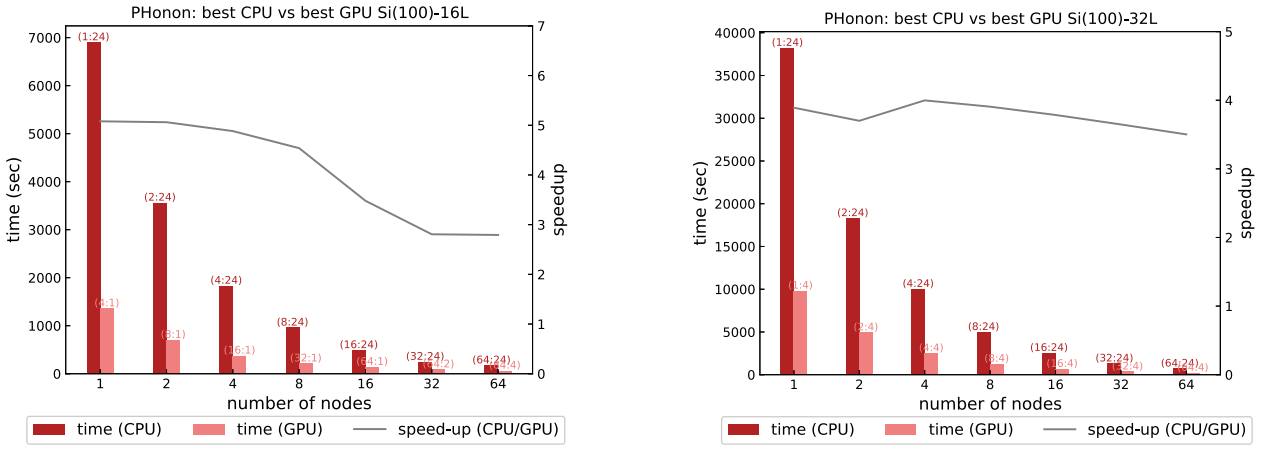
(c) Si(100)-16L, non-SCF step, scaling over *pools* with one *pool* per GPU, scaling over plane waves at fixed one *pool*



(d) Si(100)-16L, non-SCF step, scaling over *pools* with one *pool* per GPU, scaling over plane waves at fixed one *pool*

Figure 8. Scaling over *pools* and plane waves for one irreducible representation of a phonon calculation at fixed $\boldsymbol{q} = \frac{2\pi}{a}(-0.375, -1.250, 0.000)$, with 128 (**k** + **q**)-points. All calculations were done using one *image* and 8 OMP threads. Computational efficiency is reported on top of each bar.

https://doi.org/10.1021/acs.jctc.3c00249
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX



(a) Si(100)-16L model system (b) Si(100)-32L model system

Figure 9. Time to solution and CPU/GPU speed-up of a phonon calculation at fixed $\boldsymbol{q} = \frac{2\pi}{a}(-0.375, -1.250, 0.000)$, with 128 $(\mathbf{k} + \mathbf{q})$-points and one irreducible representation, using one *image* and different combinations of *pools* and *R&G* groups (*pools*:*R&G*). GPU and CPU calculations have been performed with 8 and 2 OMP threads, respectively, to optimize the respective performance.

Si(100)-32L, respectively. In the PHonon code, this heavy computational burden can be distributed by using the parallelization schemes briefly sketched in Section 2.6.

Images can be used to create nearly embarrassingly parallel tasks, each devoted to the computation of dynamical matrices for a subset of **q**-points and *irreps*. However, perturbations with $\mathbf{q} \neq 0$ lower the symmetry of the system and require a number of symmetry-independent $(\mathbf{k} + \mathbf{q})$-points that depend upon the **q**-point. As a consequence, the workload of different *images* can be quite unbalanced, and the efficiency of the overall distribution pattern depends on the particular system. For example, the present calculation has been distributed over 144 MPI ranks, subdivided in 9 *image* groups, with total number of $(\mathbf{k} + \mathbf{q})$-points varying from 21, 68, 72 and 128.

Let us first analyze the performances inside each *image* group and analyze how to optimally distribute the available ranks among *pools* and *R&G* groups. Figure 8 panels a−d show the scaling performance over *pools* and plane waves of the most computationally expensive non-SCF and Sternheimer steps (with 128 $(\mathbf{k} + \mathbf{q})$-points) for one single *irrep* at a fixed **q**-point. The maximum number of usable *pools* is 64, considering that each *pool* must have at least two points, namely, **k** and $\mathbf{k} + \mathbf{q}$. The *R&G* parallelism has an upper bound in the number of planes along the *z*-direction of the FFT grid: 256 and 512 for the Si(100)-16L and Si(100)-32L respectively. An arbitrary number of 8 and 2 OMP threads has been chosen for GPU and CPU executions, respectively, on the basis of single-node measurements (not reported here) and empirical observations reported in Section 2.6.

In all cases, we observe very good efficiencies and speed-ups when scaling over *pools*, while using *R&G* parallelism the efficiency tends to saturate earlier, with a lower amount of resources employed. Of course, the price paid for the high efficiency of the *pool* parallelism is a less favorable memory allocation, which is especially important when running on accelerated architectures. For example, the runs over 256 GPUs shown in Figure 8b,d allocated about 13 GB of GPU memory when distributed over *pools* and less than 5 GB when distributed over plane waves.

In Figure 9a,b a node-based performance comparison between our "best" CPU and GPU executions is shown, from 1 (48 cores versus 4 GPUs and 32 physical cores) to 64 nodes (3072 cores versus 256 GPUs and 128 physical cores); the latter is the maximum number of available nodes for production. The "best" CPU executions have been done using one *pool*, 24 MPI ranks per node and 2 threads per rank, and have been chosen among a set of single-node tests performed using different combinations of 1 and 2 *pools* per node and 1, 2, and 3 OpenMP threads per rank.

Regarding the GPU execution, for the smaller Si(100)-16L model, the optimal runs are those with one *pool* per GPU with no *R&G* parallelism—i.e., up to 16(64) nodes(GPUs)—that minimize communications among GPUs. Above 16(64) nodes(GPUs), the performance decreases to 3× due to the use of the additional plane-wave groups needed when *pool* parallelism is saturated.

For the larger Si(100)-32L model, the number of plane waves does not fit into one GPU memory. The best performances are obtained with one *pool* per node, with an *R&G* distribution of 4 ranks per *pool*. The maximum CPU/GPU speed-up achieved is around 5 for the smaller system and 4 for the larger system.

Once we now know how to configure parallelism inside a single *image*, we can move forward analyzing parallelism on multiple *images*. Figure 10 displays the times to solution to compute one full dynamical matrix on GPU, at one fixed **q**-point, including all the *irreps* of Si(100)-16L. Based on the previous discussion, *R&G* parallelism has not been used, and only *images* and *pools* have been varied, at a fixed *pool* size of one. Besides the obvious consideration that the calculations run faster on more nodes, an interesting aspect is that even at a fixed number of nodes, the choice of internal parallelism can play an important role for performance. This is especially evident for the most "extreme" computational setup, with 192 nodes, that shows variations in the computational times of more than 100% when changing the internal distribution of resources. If we transpose these variations in terms of GPU hours, the waste of resources due to a suboptimal choice of parallelism becomes even more dramatic. Noticeably, we observe that the best performance in Figure 10 is obtained when the available computational resources are balanced between *pools* and *images* (bars in the middle), whereas more

https://doi.org/10.1021/acs.jctc.3c00249
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

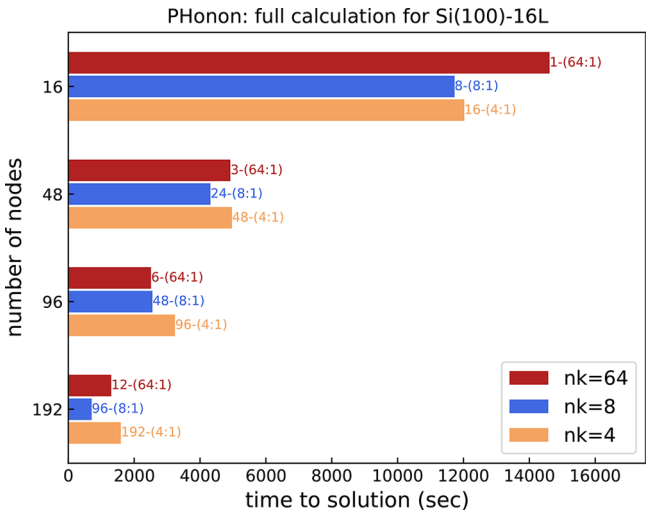


Figure 10. Time to solution (seconds) for one full phonon calculation at fixed $\mathbf{q} = \frac{2\pi}{a}(-0.375, -1.250, 0.000)$, for all the 192 irreducible representations of the Si(100)-16L model system, 8 OMP threads, and different MPI parallelization schemes. The parallelization scheme is highlighted by the labels on top of the columns as *images*-(*pools*:*R&G*).

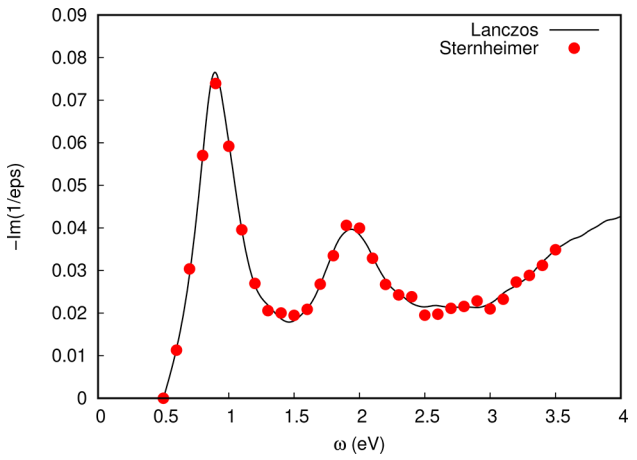


Figure 11. Electron energy loss spectrum of the silicon (100) surface, simulated using the Si(100)-16L model with transferred momentum $|\mathbf{q}| = 0.005$ Ry along $[0\bar{1}1]$, a broadening of 0.01 Ry, and a scissor shift of +0.5 eV. The solid line shows the line shape calculated using the Lanczos algorithm (20k coefficient extracted), and red points refer to Sternheimer calculations.

"extreme" allocations, where one or the other parallelism is overloaded, tend to lead to a less favorable performance.

In summary, although it is very difficult to state *a priori* what is the best parallelization scheme for a generic system, we can draw some general guidelines from our benchmark study that are helpful in the choice of the parallelization scheme of PHONON calculations for accelerated architectures. First of all, *R&G* parallelism is less efficient than the other schemes for performance, but it can be very helpful for coping with memory limitations. Then, whenever possible, it is a good strategy to balance resource allocation among *pools* and *images* using the smallest possible number of *R&G* processes.

3.3. EELS Line Shapes via TD-DFPT Methods. The turboEELS code is used to simulate the electron energy loss (EELS) and the inelastic X-ray scattering spectra in periodic solids, using two methods based on the Liouville−Lanczos scheme[11,12] and Sternheimer equations.[9,10] Both methods have been recently ported to GPU using the approach described in Section 2.

In Figure 11 the contribution to the EELS spectrum arising from the imaginary part of the dielectric function is shown for the Si(100)-16L model system described in Section 3.2, using a transferred momentum of $|\mathbf{q}| = 0.005$ Ry along the $[0\bar{1}1]$ direction. The spectral line shape has been computed using the Lanczos approach,[11,12] with a Lorentzian broadening of $\eta = 0.0035$ Ry for the charge-density susceptibility (loss function) and 20000 iterations. Along the line shape, single spectral transitions computed with the Sternheimer method[9,10] are shown, showing consistency between the two approaches. Noticeably, the spectral line shape is in fair agreement with the overall EELS spectrum obtained in more accurate and extensive studies.[48,49,54]

A turboEELS calculation with Lanczos scheme involves two main computationally intensive parts, namely, a non-SCF step, to compute wave functions and band energies at $(\mathbf{k} + \mathbf{q})$-points, and a following Lanczos chains step. *Images* are not available here, and the calculation can be distributed by using *pools* and *R&G* parallelism only.

In Figure 12a−d scaling over *pools* and plane waves of the two main steps of an EELS calculation are shown. Regarding the non-SCF step, we hereby note that, analogously to the phonon dispersion case discussed in the previous section, the chosen transferred momentum breaks all the crystal symmetries, resulting in a total number of 128 $(\mathbf{k} + \mathbf{q})$-points (i.e., 64 $\mathbf{k}$-points from the full $8 \times 8 \times 1$ Monkhorst−Pack grid[29] plus the equivalent $(\mathbf{k} + \mathbf{q})$-point ones). The scaling of the non-SCF step for the EELS spectrum is thus fully comparable to the one discussed in Figure 8a,b, with similar speed-ups and efficiencies.

Also the speed-up of the Lanczos chain step is very good, especially for the larger system (Figure 12d), where it reaches the ideal value of 16 when 256 GPUs are used.

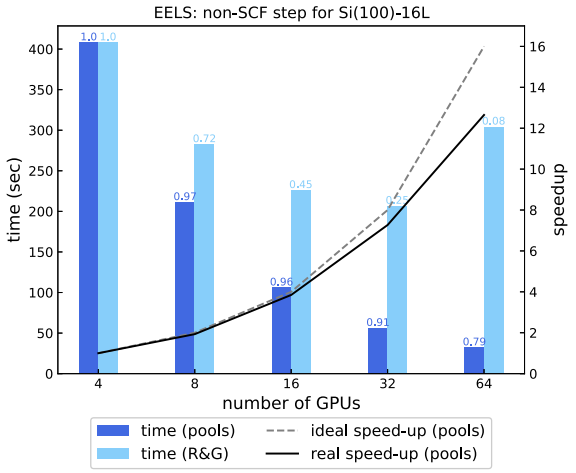
In Figure 13a,b the node-based CPU/GPU comparison is shown, and we observe that the turboEELS code shows acceleration values of 5 to 6, comparable to the PHONON code. The "best" CPU executions have been done using one *pool*, 12 MPI ranks per node and 4 threads per rank, and have been chosen among a set of single-node tests performed using different combinations of 1 and 2 *pools* per node and 1, 2, 4, and 8 OpenMP threads per rank.

3.4. Time Evolution of Large Systems with Car−Parrinello Molecular Dynamics. For a detailed description of the Car−Parrinello (CP) method we refer to the original work.[16]

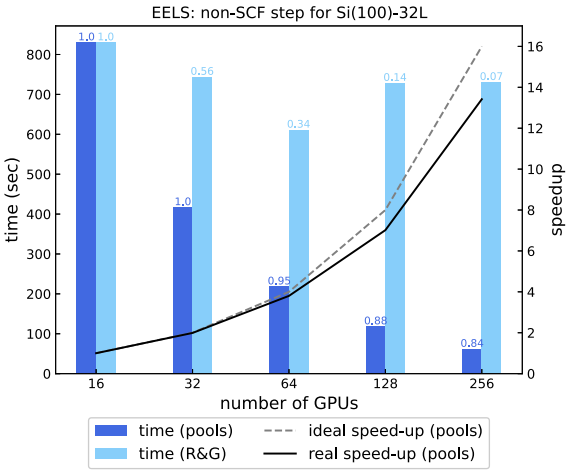
The Car−Parrinello code has been entirely ported to GPU following the approach described in Section 2, allowing the user to fully run a CP molecular dynamics simulation with nonzero initial wave function velocity on a machine with a GPU architecture.

The new code has been used to compute the superionic ammonia equation of state shown in Figure 14.[55] The color map represents the density of the system, and its pressure and temperature are shown, respectively, on the x and y axes. The superionic ammonia has been modeled with a hexagonal close-packed cell of volume 1830 Å, with 144 nitrogen and 432 hydrogen atoms (1152 electrons). Norm-conserving pseudopotentials[53,56] with nonlinear core correction have been used for both nitrogen and hydrogen atoms. The plane-wave cutoff

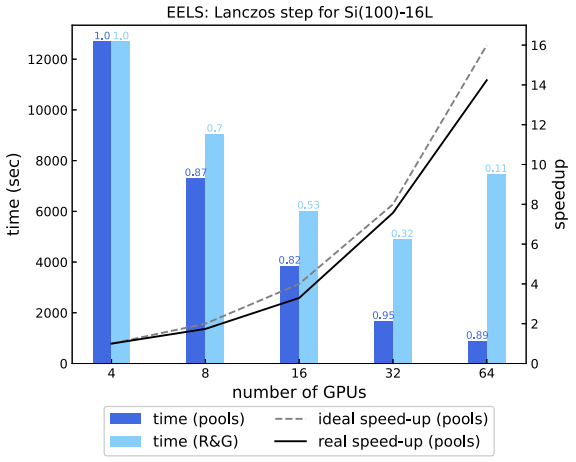
https://doi.org/10.1021/acs.jctc.3c00249
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX



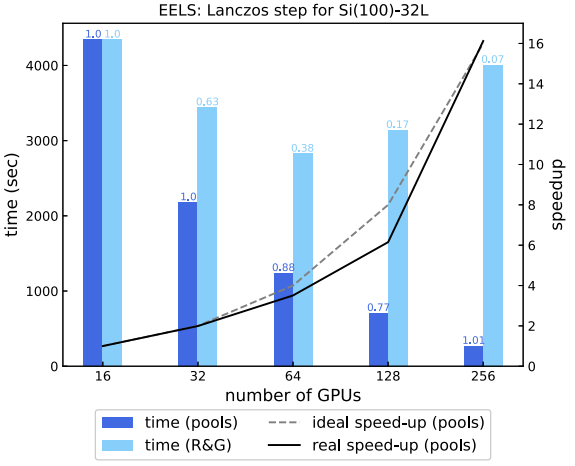
(a) Si(100)-16L, non-SCF step, scaling over *pools* with one *pool* per GPU, scaling over plane waves at fixed one *pool*



(b) Si(100)-32L, non-SCF step, scaling over *pools* with one *pool* per node, scaling over plane waves at fixed four *pools*



(c) Si(100)-16L, 1000 Lanczos iterations, scaling over *pools* with one *pool* per GPU, scaling over plane waves at fixed one *pool*



(d) Si(100)-32L, 200 Lanczos iterations, scaling over *pools* with one *pool* per node,

Figure 12. Scaling over *pools* and plane waves of different steps of an EELS calculation with transferred momentum $\mathbf{q} \| [0\bar{1}1]$, $|\mathbf{q}|$ = 0.005 Ry, with 128 ($\mathbf{k}$ + $\mathbf{q}$)-points. All calculations done using 8 OMP threads. Computational efficiency is reported on top of each bar.

was 90 Ry, sufficient for the full convergence of the energy, forces, and stress tensor. For the highest temperatures, near 3000 K, we used a time step of 0.041 fs, a fictitious electronic mass of 20 au, and a *emass_cutoff* of 2.5 Ry. At lower temperatures, we used larger time steps and electronic masses in order to speed up calculations. In all cases, we checked that a good enough conservation of the CP constant of motion was ensured.

In Figure 15, panels a and b show a node-based comparison between CPU and GPU scaling properties over plane waves of the conjugate gradient initialization and velocity Verlet step of a Car−Parrinello run on the ammonia system. Regarding CPU computations, we tried different computational setups with 1, 2, 4, and 8 threads per MPI process, and we choose those (2 threads) providing the best performance on Galileo100.

The optimal number of nodes for GPU calculations is 2, which is still significantly faster than calculations with 1 and a half nodes and negligibly slower than calculations with 3 nodes, outperforming roughly by a factor 2 the best CPU configuration with 9 nodes.

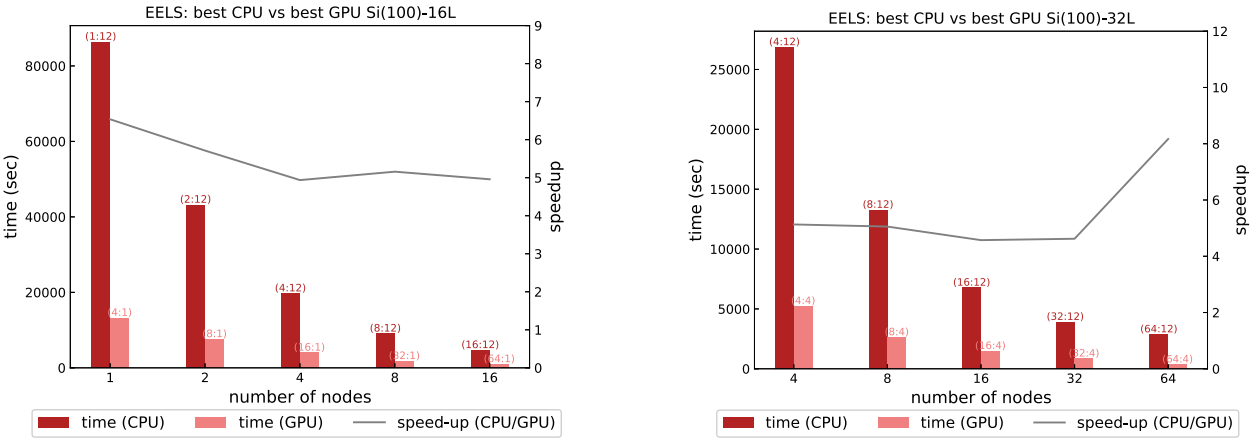
Also in this case, analogously to the PHonon and turboEELS codes, we found average speed-ups around 5−6 of GPU calculations with respect to the CPU ones.

4. CONCLUSIONS

In this article we have reviewed the current status of the Quantum ESPRESSO suite, with particular focus on the new developments done in the code since ref 2 was published.

We have first discussed the overall coding philosophy of the Quantum ESPRESSO project and of the GPU porting model, that has changed since the first versions of the code from a pure CUDA Fortran approach to a mixed interoperable OpenACC/CUDA Fortran scheme. The code refactoring done using the new scheme effectively allows people who are not necessarily experienced with GPU coding to keep contributing to the Quantum ESPRESSO project without the need to learn new languages. The porting experience

https://doi.org/10.1021/acs.jctc.3c00249
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX



(a) Si(100)-16L (b) Si(100)-32L

Figure 13. Time to solution and CPU/GPU speed-up of an EELS calculation with transferred momentum $\mathbf{q}\|[0\bar{1}1]$, $|\mathbf{q}| = 0.005$ Ry, with 128 ($\mathbf{k} + \mathbf{q}$)-points. All calculations done using 8 and 4 OMP threads for GPU and CPU executions, respectively.

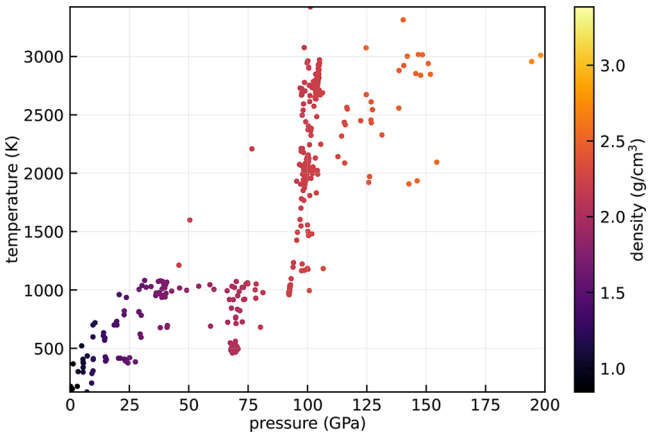
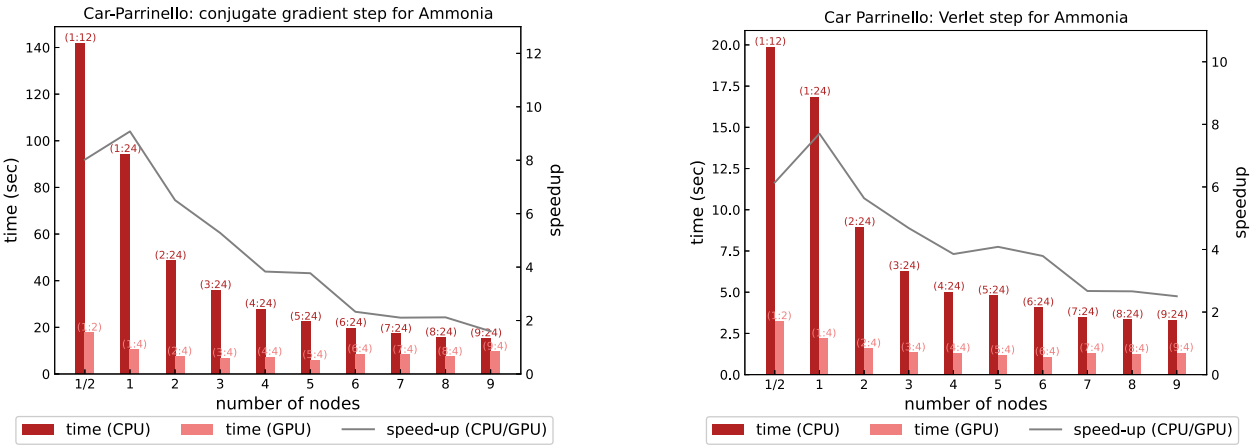


Figure 14. Equation of state diagram of ammonia, computed with the Car−Parrinello method. Each point is a CP run, and the color represents the density of the system for each particular combination of pressure and temperature.

described herein suggests that sometimes code developments done exclusively targeting performance might have downsides in terms of code readability and maintainability. It is thus very important to balance the effort to achieve outstanding performance gains with other relevant factors, such as simplicity of the code, ease of programming, and maintenance burden. In our case, the modularity of the codes also played a crucial role in the development process, allowing us to split the huge porting task into many smaller sub tasks and also allowing many different executables to take advantage of a relatively limited number of ported modules and libraries.

The GPU porting has significantly progressed in the last years, and now the most important codes of the suite, namely, PWscf, PHonon, turboEELS, turboLanczos, CP, and HP, are fully operative on heterogeneous architectures.

MPI data distribution and code parallelism schemes available in the Quantum ESPRESSO suite have been also reviewed in this article, highlighting the aspects related to GPU execution.



(a) Conjugate gradient step (times averaged over of 66 iterations) (b) Verlet steps (times averaged over of 20 iterations)

Figure 15. Times to solution and CPU/GPU speed-up of different steps of a CP simulation of Ammonia. CPU and GPU calculations done with 2 and 8 OMP threads, respectively.

https://doi.org/10.1021/acs.jctc.3c00249
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

Extensive benchmark tests have been provided for the main codes to assess performance and identify best practices for launching calculations. Results depend to some extent on factors that are related to the algorithms used in the QUANTUM ESPRESSO codes and can be considered quite general. For example, when **k**-points are present, *pool* parallelism effectively allows strong scaling, whereas when massive resources are employed, it can be beneficial to exploit a fraction of the available cores as OpenMP threads, in order to reduce communication bottlenecks. However, performance is also strongly influenced by many other system-dependent factors, related to both the molecular system (e.g., symmetry, number of electrons) and the computational architecture (e.g., inter- and intranode communication bandwidths, number of RAM memory channels, CPU clock frequencies, GPU architecture, host-device communication bandwidths) that can affect latency and efficiency. With respect to the architectures employed here, computations on Marconi100 using V100 GPU cards are about four to six times faster than CPU computations using the same number of nodes on the Galileo100 cluster. Tests done using A100 cards on the Selene cluster also suggest even better performance on architectures based on more modern GPU technologies.

One of the main limitations found in the present version of the QUANTUM ESPRESSO codes is the difficulty to scale computations with no **k**-points, as *R&G* parallelism suffers from communication bottlenecks already at a relatively small number of ranks. In this respect, work is in progress to improve the current *band* parallelism and communication protocols in the MPI libraries of the QUANTUM ESPRESSO suite.

AUTHOR INFORMATION

Corresponding Author

Ivan Carnimeo − *SISSA, Scuola Internazionale Superiore di Studi Avanzati, 34136 Trieste, Italy;* orcid.org/0009-0006-8640-9828; Email: icarnimeo@sissa.it

Authors

Fabio Affinito − *CINECA, 40033 Casalecchio di Reno, BO, Italy*

Stefano Baroni − *SISSA, Scuola Internazionale Superiore di Studi Avanzati, 34136 Trieste, Italy; CNR-IOM Istituto dell'Officina dei Materiali, area SISSA, 34136 Trieste, Italy*

Oscar Baseggio − *SISSA, Scuola Internazionale Superiore di Studi Avanzati, 34136 Trieste, Italy*

Laura Bellentani − *CINECA, 40033 Casalecchio di Reno, BO, Italy*

Riccardo Bertossa − *SISSA, Scuola Internazionale Superiore di Studi Avanzati, 34136 Trieste, Italy;* orcid.org/0000-0002-8551-1939

Pietro Davide Delugas − *SISSA, Scuola Internazionale Superiore di Studi Avanzati, 34136 Trieste, Italy*

Fabrizio Ferrari Ruffino − *CNR-IOM Istituto dell'Officina dei Materiali, area SISSA, 34136 Trieste, Italy*

Sergio Orlandini − *CINECA, 00185 Roma, RM, Italy*

Filippo Spiga − *NVIDIA Corporation, Santa Clara, California 95051, United States*

Paolo Giannozzi − *Dipartimento di Scienze Matematiche, Informatiche e Fisiche (DMIF), Università degli Studi di Udine, 33100 Udine, Italy; CNR-IOM Istituto dell'Officina dei Materiali, area SISSA, 34136 Trieste, Italy*

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jctc.3c00249

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

This work was funded by the European Union through the MAX "MAterials design at the eXascale" Centre of Excellence for Supercomputing applications (Grant agreement No. 101093374, co-funded by the European High Performance Computing joint Undertaking (JU) and participating countries, and EU H2020-INFRAEDI-2018-1 MAX Grant No. 824143), by the Italian MUR through the PRIN 2017 *FERMAT* (grant No. 2017KFY7XF) and the *National Centre for HPC, Big Data, and Quantum Computing* (grant No. CN00000013).

REFERENCES

(1) https://www.top500.org (last access Jun, 7th 2023).

(2) Giannozzi, P.; Baseggio, O.; Bonfà, P.; Brunato, D.; Car, R.; Carnimeo, I.; Cavazzoni, C.; de Gironcoli, S.; Delugas, P.; Ferrari Ruffino, F.; Ferretti, A.; Marzari, N.; Timrov, I.; Urru, A.; Baroni, S. Quantum ESPRESSO toward the exascale. *J. Chem. Phys.* **2020**, *152* (1−11), 154105.

(3) Giannozzi, P.; Andreussi, O.; Brumme, T.; Bunau, O.; Nardelli, M. B.; Calandra, M.; Car, R.; Cavazzoni, C.; Ceresoli, D.; Cococcioni, M.; Colonna, N.; Carnimeo, I.; Dal Corso, A.; de Gironcoli, S.; Delugas, P.; DiStasio, R. A., Jr; Ferretti, A.; Floris, A.; Fratesi, G.; Fugallo, G.; Gebauer, R.; Gerstmann, U.; Giustino, F.; Gorni, T.; Jia, J.; Kawamura, M.; Ko, H.-Y.; Kokalj, A.; Kucukbenli, E.; Lazzeri, M.; Marsili, M.; Marzari, N.; Mauri, F.; Nguyen, N. L.; Nguyen, H.-V.; Otero-de-la Roza, A.; Paulatto, L.; Ponce, S.; Rocca, D.; Sabatini, R.; Santra, B.; Schlipf, M.; Seitsonen, A. P.; Smogunov, A.; Timrov, I.; Thonhauser, T.; Umari, P.; Vast, N.; Wu, X.; Baroni, S. Advanced capabilities for materials modelling with QUANTUM ESPRESSO. *J. Phys.: Condens. Matter* **2017**, *29*, 465901.

(4) Giannozzi, P.; Baroni, S.; Bonini, N.; Calandra, M.; Car, R.; Cavazzoni, C.; Ceresoli, D.; do L Chiarotti, G.; Cococcioni, M.; Dabo, I.; Corso, A. D.; de Gironcoli, S.; Fabris, S.; Fratesi, G.; Gebauer, R.; Gerstmann, U.; Gougoussis, C.; Kokalj, A.; Lazzeri, M.; Martin-Samos, L.; Marzari, N.; Mauri, F.; Mazzarello, R.; Paolini, S.; Pasquarello, A.; Paulatto, L.; Sbraccia, C.; Scandolo, S.; Sclauzero, G.; Seitsonen, A. P.; Smogunov, A.; Umari, P.; Wentzcovitch, R. M. QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials. *J. Phys.: Condens. Matter* **2009**, *21*, 395502.

(5) Scandolo, S.; Giannozzi, P.; Cavazzoni, C.; de Gironcoli, S.; Pasquarello, A.; Baroni, S. First-principles codes for computational crystallography in the Quantum-ESPRESSO package. *Zeitschrift für Kristallographie* **2005**, *220*, 574−579.

(6) QUANTUM ESPRESSO FOUNDATION. https://foundation.quantum-espresso.org/ (last access Jun, 7th 2023).

(7) Giannozzi, P.; De Gironcoli, S.; Pavone, P.; Baroni, S. Abinitio Calculation of Phonon Dispersions in Semiconductors. *Phys. Rev. B* **1991**, *43*, 7231−7242.

(8) Baroni, S.; de Gironcoli, S.; Dal Corso, A.; Giannozzi, P. Phonons and related crystal properties from density-functional perturbation theory. *Rev. Mod. Phys.* **2001**, *73*, 515−562.

(9) Motornyi, O.; Raynaud, M.; Dal Corso, A.; Vast, N. Simulation of electron energy loss spectra with the turboEELS and thermo_pw codes. *XXIXTH IUPAP Conference on Computational Physics CCP2017*; 39th IUPAP Conference on Computational Physics (CCP), Univ Pierre Marie Curie Sorbonne, Paris, France, JUL 09−13, 2017; IUPAC, 2018.

(10) Motornyi, O.; Vast, N.; Timrov, I.; Baseggio, O.; Baroni, S.; Dal Corso, A. Electron energy loss spectroscopy of bulk gold with ultrasoft pseudopotentials and the Liouville-Lanczos method. *Phys. Rev. B* **2020**, *102*, 035156 035156.

(11) Walker, B.; Saitta, A. M.; Gebauer, R.; Baroni, S. Efficient Approach to Time-Dependent Density-Functional Perturbation Theory for Optical Spectroscopy. *Phys. Rev. Lett.* **2006**, *96*, 113001.
(12) Rocca, D.; Gebauer, R.; Saad, Y.; Baroni, S. Turbo charging time-dependent density-functional theory with Lanczos chains. *J. Chem. Phys.* **2008**, *128*, 154105.
(13) Timrov, I.; Marzari, N.; Cococcioni, M. HP - A code for the calculation of Hubbard parameters using density-functional perturbation theory. *Comput. Phys. Commun.* **2022**, *279*, 108455.
(14) Timrov, I.; Marzari, N.; Cococcioni, M. Hubbard parameters from density-functional perturbation theory. *Phys. Rev. B* **2018**, *98*, 085127.
(15) Timrov, I.; Marzari, N.; Cococcioni, M. Self-consistent Hubbard parameters from density-functional perturbation theory in the ultrasoft and projector-augmented wave formulations. *Phys. Rev. B* **2021**, *103*, 045141.
(16) Car, R.; Parrinello, M. Unified Approach for Molecular Dynamics and Density-Functional Theory. *Phys. Rev. Lett.* **1985**, *55*, 2471−2474.
(17) MAX: Materials at the eXascale. *An EU Centre of Excellence for Supercomputing Applications*. http://www.max-centre.eu/ (last access Jun, 7th 2023).
(18) Yu, V. W.-z.; Moussa, J.; Kus, P.; Marek, A.; Messmer, P.; Yoon, M.; Lederer, H.; Blum, V. GPU-acceleration of the ELPA2 distributed eigensolver for dense symmetric and hermitian eigenproblems. *Comput. Phys. Commun.* **2021**, *262*, 107808.
(19) Lehtola, S.; Steigemann, C.; Oliveira, M.; Marques, M. Recent developments in Libxc - A comprehensive library of functionals for density functional theory. *Software X* **2018**, *7*, 1−5.
(20) Heyd, J.; Scuseria, G.; Ernzerhof, M. Hybrid functionals based on a screened Coulomb potential. *J. Chem. Phys.* **2003**, *118*, 8207−8215.
(21) Becke, A. Density-Functional Thermochemistry.3. Ther Role of Exact Exchange. *J. Chem. Phys.* **1993**, *98*, 5648−5652.
(22) Adamo, C.; Barone, V. Toward reliable density functional methods without adjustable parameters: The PBE0 model. *J. Chem. Phys.* **1999**, *110*, 6158−6170.
(23) Barnes, T. A.; Kurth, T.; Carrier, P.; Wichmann, N.; Prendergast, D.; Kent, P. R.; Deslippe, J. Improved treatment of exact exchange in Quantum ESPRESSO. *Comput. Phys. Commun.* **2017**, *214*, 52−58.
(24) Lin, L. Adaptively Compressed Exchange Operator. *J. Chem. Theory Comput.* **2016**, *12*, 2242−2249.
(25) Lin, L.; Lindsey, M. Convergence of Adaptive Compression Methods for Hartree-Fock-Like Equations. *Commun. Pure Appl. Math* **2019**, *72*, 451−499.
(26) Carnimeo, I.; Baroni, S.; Giannozzi, P. Fast hybrid density-functional computations using plane-wave basis sets. *Electron. struct.* **2019**, *1*, 015009.
(27) Damle, A.; Lin, L.; Ying, L. Compressed Representation of Kohn-Sham Orbitals via Selected Columns of the Density Matrix. *J. Chem. Theory Comput.* **2015**, *11*, 1463−1469.
(28) Damle, A.; Lin, L.; Ying, L. SCDM-k: Localized orbitals for solids via selected columns of the density matrix. *J. Comput. Phys.* **2017**, *334*, 1−15.
(29) Monkhorst, H. J.; Pack, J. D. Special points for Brillouin-zone integrations. *Phys. Rev. B* **1976**, *13*, 5188−5192.
(30) Marzari, N.; Vanderbilt, D. Maximally localized generalized Wannier functions for composite energy bands. *Phys. Rev. B* **1997**, *56*, 12847−12865.
(31) Marzari, N.; Mostofi, A. A.; Yates, J. R.; Souza, I.; Vanderbilt, D. Maximally localized Wannier functions: Theory and applications. *Rev. Mod. Phys.* **2012**, *84*, 1419−1475.
(32) Souza, I.; Marzari, N.; Vanderbilt, D. Maximally localized Wannier functions for entangled energy bands. *Phys. Rev. B* **2001**, *65*, 035109.
(33) Koelling, D.; Wood, J. On the interpolation of eigenvalues and a resultant integration scheme. *J. Comput. Phys.* **1986**, *67*, 253−262.
(34) Pickett, W. E.; Krakauer, H.; Allen, P. B. Smooth Fourier interpolation of periodic functions. *Phys. Rev. B* **1988**, *38*, 2721−2726.
(35) Mostofi, A. A.; Yates, J. R.; Lee, Y.-S.; Souza, I.; Vanderbilt, D.; Marzari, N. wannier90: A tool for obtaining maximally-localised Wannier functions. *Comput. Phys. Commun.* **2008**, *178*, 685−699.
(36) Mostofi, A. A.; Yates, J. R.; Pizzi, G.; Lee, Y.-S.; Souza, I.; Vanderbilt, D.; Marzari, N. An updated version of wannier90: A tool for obtaining maximally-localised Wannier functions. *Comput. Phys. Commun.* **2014**, *185*, 2309−2310.
(37) Pizzi, G.; Vitale, V.; Arita, R.; Bluegel, S.; Freimuth, F.; Geranton, G.; Gibertini, M.; Gresch, D.; Johnson, C.; Koretsune, T.; Ibanez-Azpiroz, J.; Lee, H.; Lihm, J.-M.; Marchand, D.; Marrazzo, A.; Mokrousov, Y.; Mustafa, J. I; Nohara, Y.; Nomura, Y.; Paulatto, L.; Ponce, S.; Ponweiser, T.; Qiao, J.; Thoele, F.; Tsirkin, S. S.; Wierzbowska, M.; Marzari, N.; Vanderbilt, D.; Souza, I.; Mostofi, A. A.; Yates, J. R. Wannier90 as a community code: new features and applications. *J. Phys.: Condens. Matter* **2020**, *32*, 165902.
(38) Wagner, M.; Lopez, V.; Morillo, J.; Cavazzoni, C.; Affinito, F.; Gimenez, J.; Labarta, J. Performance Analysis and Optimization of the FFTXlib on the Intel Knights Landing Architecture. *2017 46TH International Conference on Parallel Processing Workshops (ICPPW), AUG 14−17, 2017*; ICPPW, 2017; pp 243−250.
(39) Henkelman, G.; Uberuaga, B.; Jonsson, H. A climbing image nudged elastic band method for finding saddle points and minimum energy paths. *J. Chem. Phys.* **2000**, *113*, 9901−9904.
(40) Henkelman, G.; Jonsson, H. Improved tangent estimate in the nudged elastic band method for finding minimum energy paths and saddle points. *J. Chem. Phys.* **2000**, *113*, 9978−9985.
(41) Asgeirsson, V.; Birgisson, B. O.; Bjornsson, R.; Becker, U.; Neese, F.; Riplinger, C.; Jonsson, H. Nudged Elastic Band Method for Molecular Reactions Using Energy-Weighted Springs Combined with Eigenvector Following. *J. Chem. Theory Comput.* **2021**, *17*, 4929−4945.
(42) Sternheimer, R. M. Electronic Polarizabilities of Ions from the Hartree-Fock Wave Functions. *Phys. Rev.* **1954**, *96*, 951−968.
(43) Gorni, T.; Baseggio, O.; Delugas, P.; Baroni, S.; Timrov, I. turboMagnon - A code for the simulation of spin-wave spectra using the Liouville-Lanczos approach to time-dependent density-functional perturbation theory. *Comput. Phys. Commun.* **2022**, *280*, 108500.
(44) Gorni, T.; Baseggio, O.; Delugas, P.; Timrov, I.; Baroni, S. First-principles study of the gap in the spin excitation spectrum of the CrI_3 honeycomb ferromagnet. 2022; https://arxiv.org/abs/2212.09516.
(45) Delugas, P.; Baseggio, O.; Timrov, I.; Baroni, S.; Gorni, T. Magnon-phonon interactions enhance the gap at the Dirac point in the spin-wave spectra of CrI_3 2D magnets. 2021; https://arxiv.org/abs/2105.04531.
(46) Dal Corso, A. Pseudopotentials periodic table: From H to Pu. *Comput. Mater. Sci.* **2014**, *95*, 337−350.
(47) MaX School on Advanced Materials and Molecular Modelling with Quantum ESPRESSO, Day 9, Hands-on on "QE on HPC and GPU systems. https://gitlab.com/QEF/materials-for-max-qe2021-online-school/-/blob/master/Day-9/Handson-IC.pdf (last access Jun, 7th 2023).
(48) Yoshinobu, J. Physical properties and chemical reactivity of the buckled dimer on Si(1 0 0). *Prog. Surf. Sci.* **2004**, *77*, 37−70.
(49) Hogan, C.; Caramella, L.; Onida, G. Simulation of the oxidation pathway on Si(100) using high-resolution EELS. *Phys. Status Solidi B* **2012**, *249*, 1132−1139.
(50) Chadi, D. J. Si(100) surfaces: Atomic and electronic structures. *J. Vac. Sci. Technol.* **1979**, *16*, 1290−1296.
(51) Setyawan, W.; Curtarolo, S. High-throughput electronic band structure calculations: Challenges and tools. *Comput. Mater. Sci.* **2010**, *49*, 299−312.
(52) Perdew, J. P.; Burke, K.; Ernzerhof, M. Generalized Gradient Approximation Made Simple. *Phys. Rev. Lett.* **1996**, *77*, 3865−3868.
(53) Hamann, D. R. Optimized norm-conserving Vanderbilt pseudopotentials. *Phys. Rev. B* **2013**, *88*, 085117.

https://doi.org/10.1021/acs.jctc.3c00249
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

(54) Farrell, H. H.; Stucki, F.; Anderson, J.; Frankel, D. J.; Lapeyre, G. J.; Levinson, M. Electronic excitations on Si(100)(2 × 1). *Phys. Rev. B* **1984**, *30*, 721−725.
(55) Bertossa, R. Theory, codes, and numerical simulation of heat transport in multicomponent systems. *Ph.D. thesis*, Scuola Internazionale Superiore di Studi Avanzati, Trieste, 2022.
(56) van Setten, M.; Giantomassi, M.; Bousquet, E.; Verstraete, M.; Hamann, D.; Gonze, X.; Rignanese, G.-M. The PseudoDojo: Training and grading a 85 element optimized norm-conserving pseudopotential table. *Comput. Phys. Commun.* **2018**, *226*, 39−54.

https://doi.org/10.1021/acs.jctc.3c00249
J. Chem. Theory Comput. XXXX, XXX, XXX−XXX