Lawrence Berkeley National Laboratory

LBL Publications

Title

Inq, a Modern GPU-Accelerated Computational Framework for (Time-Dependent) Density Functional Theory

Permalink

Journal

ISSN

Authors

Andrade, Xavier
Pemmaraju, Chaitanya Das
Kartsev, Alexey
et al.

Publication Date

DOI

Copyright Information

Peer reviewed

INQ, a Modern GPU-Accelerated Computational Framework for (Time-Dependent) Density Functional Theory

Xavier Andrade,* Chaitanya Das Pemmaraju, Alexey Kartsev, Jun Xiao, Aaron Lindenberg, Sangeeta Rajpurohit, Liang Z. Tan, Tadashi Ogitsu, and Alfredo A. Correa

Cite This: *J. Chem. Theory Comput.* 2021, 17, 7447–7467

Read Online

ACCESS | Metrics & More | Article Recommendations

ABSTRACT: We present INQ, a new implementation of density functional theory (DFT) and time-dependent DFT (TDDFT) written from scratch to work on graphic processing units (GPUs). Besides GPU support, INQ makes use of modern code design features and takes advantage of newly available hardware. By designing the code around algorithms, rather than against specific implementations and numerical libraries, we aim to provide a concise and modular code. The result is a fairly complete DFT/TDDFT implementation in roughly 12 000 lines of open-source C++ code representing a modular platform for community-driven application development on emerging high-performance computing architectures.
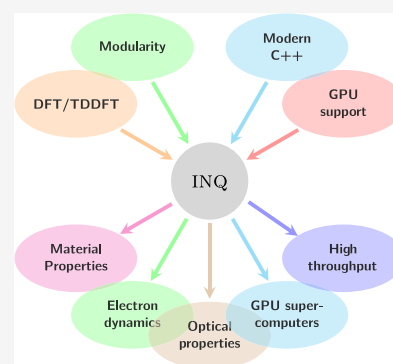
I. INTRODUCTION

The density functional theory (DFT) framework[1,2] provides a computationally tractable way to approximate and solve the quantum many-body problem for electrons,[3] both in the ground state and in the excited state.[4] In last decades, DFT has been extremely successful to the point of becoming the standard method for first-principles simulations in computational chemistry, solid-state physics, and material science.[5−8] This success has been possible in large part due to computer programs that can solve the DFT equations efficiently, accurately, and reliably.[9−26]

However, there are still many challenges that DFT faces in terms of theory and computer codes to be applicable to new kinds of problems.[27] One of these challenges is the application of data science to electronic structure, for example, through high-throughput material screening,[28−30] structure discovery,[31] or machine learning techniques,[32−36] where thousands or even millions[37] of DFT calculations are performed. For this, it is important to manage input parameters, output results, executions, and error exceptions of simulation codes as simply as possible.

Another issue is that the standard approximations for exchange and correlation (XC) functionals fail for many types of systems.[38−40] This is particularly important in time-dependent DFT (TDDFT)[4,41] where the usual adiabatic approximation for the XC functional misses part of the physics involved in the description of excited states.[41−45] For example, the adiabatic and semilocal functionals in TDDFT cannot describe excitons,[42] which are important to simulate from first

principles as they control the optical gaps, energy relaxation and transport pathways of semiconductors, molecular, and nano-structured systems.

Recently, it has been shown that it is possible to describe excitons in TDDFT using hybrid functionals and better XC approximations.[46−51] As expected, these improvements come with considerable additional computational costs.

Fortunately, computing power has increased at a rapid pace in the last few years, with exascale-level supercomputers coming in the next few years.[52] To use this new hardware to simulate new physics in more realistic systems, we require new software that can run efficiently on these modern computing platforms.

The super-scalar central processing units (CPUs) that have dominated high-performance computing for a long time have been largely replaced by graphic processing units (GPUs). GPUs have a much more parallel architecture that offers a considerably larger numerical throughput and memory bandwidth than a CPU with similar cost and power consumption. Unfortunately, GPUs cannot directly run codes written for the CPU as they need code that is written in parallel and the GPU memory needs to be managed explicitly.

ACS Publications

This means that existing DFT codes, many started more than two decades ago, need to be adapted to this new paradigm. This is not an easy task as the solution of the DFT/TDDFT equations is quite complex. It involves a combination of algorithms and computational kernels that need to work efficiently together. Many of the DFT software packages contain hundreds of thousands of lines of code. Due to these factors, the adoption of GPUs in DFT simulations has been slow when compared with other areas like classical molecular dynamics.[53,54] Right now, only a few DFT codes offer production-ready GPU support and with limited performance gains for specific cases.[55−59]

When faced with the necessity of an efficient DFT implementation that can make use of large GPU-based supercomputers, we decided that it was more practical to start an implementation from scratch, instead of modifying an existing code. We took this opportunity to create a modern code that profits from current design ideas and programming tools. At the same time, it is based on years of experience that our team has in the development of the codes OCTOPUS[14,56,60−63] and QBALL[20,64,65] (the TDDFT implementation branch from QBOX[15,66]). This allowed us to write INQ, a streamlined implementation of DFT and TDDFT that is very compact, only 12 000 lines of code, and that is easy to maintain and further develop.

This article gives a complete overview of the design and implementation of INQ. We show how modern approaches to coding can be applied to DFT problems with advantageous results. We then present results of some calculations performed with INQ and how they compare with other DFT codes in terms of accuracy. The numerical performance of the code will be the subject of a future publication.

The current version of INQ uses innovative coding practices to implement conventional DFT algorithms, including both plane-wave and real-space strategies that have been used in other DFT codes. This approach will allow newer algorithmic improvements to be incorporated into INQ in a systematic and fail-safe way, building upon a solid foundation of tried-and-tested methods.

II. CODE DEVELOPMENT

We start the article by discussing our general development strategy for INQ. This is one of the most fundamental aspects of code development that can make the difference between a successful piece of software or an abandoned project.

A scientific program is not a static entity that is written once and used for years without modification. Besides the usual problems in the code that are regularly discovered and need to be fixed, modification is essential in software. More precise or more computationally efficient algorithms and theories appear and need to be tested or implemented. And finally, computational platforms change and codes need to be adapted for them.

This is why our main objective when designing and developing INQ is that it can be easily modified and improved while providing consistent results and behavior. At the same time, it should offer all of the features expected of a modern electronic structure code and the highest performance possible. We combine several elements to achieve our objective, which we briefly discuss next and which are expanded in the following sections.

First, INQ does not follow the traditional code interface based on input files. Instead, INQ is a library where the user input is a computer program. We explain our approach in Section III.

Second, INQ has a highly modular structure where the code is divided into several components with well-defined tasks. In this way, the different components can be developed and tested independently. They can also be shared with other codes to avoid duplication of work and enable collaborative development. In fact, we use third-party components when they are available and, when possible, contribute to their development (as in the case of LIBXC). This is discussed in detail in Section IV.

A third aspect is the use of modern programming in C++, explained in Section V, that allows for a high level of abstraction while retaining the efficiency of a low-level language. The result is a code that is simple to write and read and that resembles as much as possible the underlying mathematics. Details like memory management and parallelization are hidden as much as possible, and most programmers do not need to worry about them. A particularly advantageous application of modern C++ is to facilitate GPU programming. We developed a simple and general model to write a GPU code that we describe in Section VI.

An important aspect of the code design of INQ is that we recognize that it is very hard to determine what is the best code design a priori, so we do not attempt to do it. When implementing something, our priority is to have the simplest implementation that works, write tests, and only then figure out what is the best way of coding it. And even then, we are always willing to change it since the "optimal" solution might change over time depending on other factors.

Finally, for a scientific code, the reliability of the results, and their consistency after code modification, is essential. With this idea in mind, we rely heavily on a systematic, exhaustive, and continuous testing of the code. Tests are written at the same time, or even before, a new component or feature is developed.[67] They ensure that the implementation always gives the expected results. Once the initial results are validated, the tests verify that the results do not change when the code is modified or when it is run on a different platform. In particular, they check that the CPU and GPU versions of the code give the same results.

We use two types of tests: unit tests and integration tests. Unit tests are small tests written for every individual component of the code, a class or function. They verify that the components
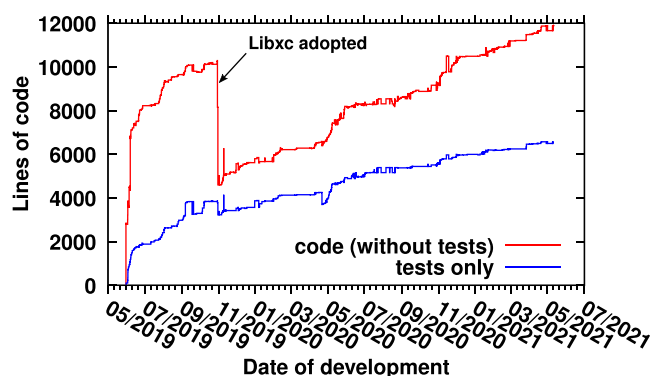

Figure 1. INQ source code lines *vs* time since the start of the project. Note that the test and code are developed side by side, and the amount of lines of code is comparable. We find testing fundamental for continuous development and maintenance of the code. The dip near 11/2019 reflects the removal of the internal exchange and correlation (XC) functional computation, which was replaced by the external library LIBXC. This illustrates the importance of delegating functionality to other high-quality libraries when possible.

are giving proper results under all conditions. The reference values for these tests are normally obtained from analytical results for known conditions, including corner cases. Integration tests in INQ are calculations for particular systems whose results are verified analytically or with other codes. These examples are designed to span all of the possible types of calculations and features.

All of the development of INQ is tracked through a source control system. For this, we use GIT[68] and keep our central repository on the GITLAB service. As a set of changes are made to the code, all of the tests are automatically executed by a "continuous integration" (CI) system provided by GITLAB. We configured this system to compile and run the code in a variety of platforms (CPU/GPU, serial/parallel, and different compilers and libraries). A failure in compilation, or any of the tests, blocks the changes from being included into the code until they are fixed.

The CI also uses the CODECOV tool to evaluate how well the tests are checking all of the lines of code. The code coverage in INQ is approximately 95%, which means that almost every line of code is executed during testing.

In Figure 1, we show the number of lines of code in INQ as a function of time since the start of the project. It illustrates several things. In the first place, it shows that INQ offers an implementation of DFT and TDDFT in roughly 12 000 lines of code (not considering tests). This is quite an achievement considering that a code that offers similar functionality like QBALL

has more than 60 000 lines. Other electronic structure codes can reach much higher counts, in the hundreds of thousands of lines of code. The figure also shows how the tests in INQ are developed concurrently with the code, and that constitute a significant part of the total amount of the code. While this might seem a waste of effort, in fact, it makes development easier and faster by reducing the time spent on code debugging. The tests make errors much less likely, and when errors do appear, they are much easier to detect and fix.

III. INQ USER INTERFACE

INQ follows a modular philosophy: the code is split into different components with well-defined tasks. Following that idea, INQ itself is designed as a component that can be used by other programs. This means that in practice, INQ is not a program that is executed by users but a C++ library that provides all of the functionality of a DFT/TDDFT code. Instead of using an ad hoc input-file format, INQ input files are directly written in standard C++ and compiled, possibly taking advantage of information specific to the desired calculation. A sample input file is shown in Listing 1. While this may require a bit of effort from users, this approach has several key advantages in comparison to traditional codes. A similar approach is used by the GPAW code that uses Python scripts as input[17] and by the Atomic Simulation Environment[69] in an attempt to control uniformly different existing codes with a single Python interface.

```cpp
#include <inq/inq.hpp>

int main(int argc, char * argv[]){

  using namespace inq;
  using namespace inq::input;
  using namespace inq::magnitude;

  inq::environment env(argc, argv);
  auto comm = mpi3::environment::get_world_instance();

  auto distance = 1.06_angstrom;

  std::vector<atom> geo(2);
  geo[0] = "N" | coord(0.0, 0.0, -distance/2);
  geo[1] = "N" | coord(0.0, 0.0,  distance/2);

  system::ions ions(cell::cubic(10.0_bohr) | cell::finite(), geo);
  system::electrons electrons(comm, ions, basis::cutoff_energy(80.0_Ry));

  ground_state::initialize(ions, electrons);

  auto result = ground_state::calculate(ions, electrons, interaction::pbe());

  std::cout <<"N2 energy = "<< result.energy.total() <<" Hartree\n";
}
```

Listing 1: Example of an INQ program (equivalent to an input file) for the DFT calculation of the nitrogen molecular ground state. The input is a regular C++ source code file that is compiled normally. Up to line 9, we initialize the code to use INQ. In line 12, we define the interatomic distance for our molecule; note that the user needs to explicitly give the units, in this case angstrom. Lines 14−16 create the geometry of the molecule inside a standard C++ dynamic array (`std::vector`). In line 18, the ionic configuration geometry and the cell are brought together in an ionic subsystem. In line 19, an electronic subsystem (Kohn−Sham (KS) electrons) is created and allocated across all message passing interface (MPI) tasks available. Line 20 initializes the electronic plane-wave coefficients with a guess for the ground-state calculation. In line 23, a convenience function called "calculate" optimizes the KS system to the ground state for fixed ion configuration. Finally, the calculated total energy is printed on the screen. In the

past, quantum simulations were quite expensive and researchers could only afford to make a few simulations in a research project. Today, fast computers, efficient and reliable codes, and data analysis techniques have made it feasible for a single user to perform large numbers of DFT/TDDFT simulations. In particular, this has led to the concept of high-throughput computational materials screening.

The library approach of INQ offers a considerable advantage in this scenario. In the first place, users need not learn an additional syntax for writing input files. With INQ, since the input file is already a program in a general programming language, all of the automation can be done directly and the code can be easily integrated with other libraries. For example, we have written a C++ program that can connect to the materials project[70] to download a structure and use it as an input for INQ.

A particularly complicated part of writing scripts that call a third-party code is the parsing and postprocessing of results from

the output files. In INQ instead, users can directly access the results as data structures instead of having to write parsing routines for output files. Any postprocessing of the data can be also done directly and make use of the functionality already provided by INQ.

An additional advantage is that users do not have to face a barrier when they need to modify or extend INQ. Since they are already using C++, it is natural to explore the code and its lower-level interfaces to tailor INQ for their specific needs. For example, if a user needs to implement a new observable, they can directly access the data structures and use INQ operations over them.

The difference between the usual paradigms of calculations and INQ is illustrated in Figure 2. INQ blurs the boundaries
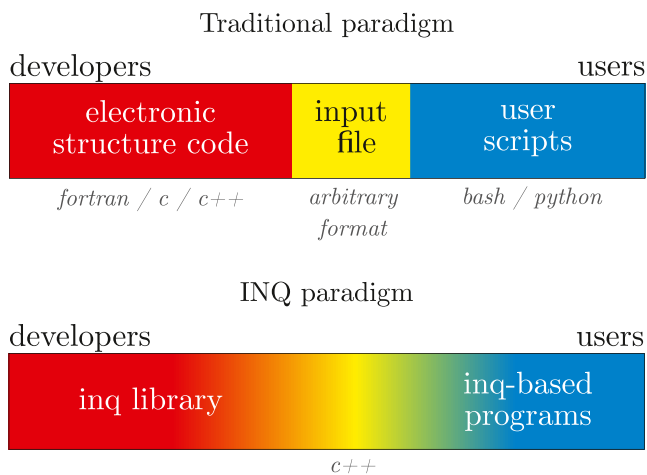

Figure 2. Schematically, we develop INQ with the idea of blurring the distinction between the input and main program. We offer all of the available opportunities to drive simulations in the same framework that the low-level functionality is implemented in. This paradigm obviates the need for input files, output files for postprocessing, and the potential need to deal with three different languages. A coherent organization of the information of a simulation is accessible in memory at any level of the framework. Simple or complex programs can be written in the same language than the rest of the library; there is no need to invent an ad hoc input format or language to specify the desired simulation. Simple simulation tasks (e.g., plain ground-state calculation, time-dependent propagation) correspond to simple programs that use the library. Complex uses of the system may need to access advanced levels of functionality, further development of the library, or interaction with other third-party libraries (e.g., other molecular codes, machine learning libraries).

between the electronic structure code, the input file, and the calculation processing scripts and merges them all into a continuum with different levels of abstraction and detailed access. This continuum gives more power to the users, and developers of other programs, to adapt the code to their specific needs.

Another purpose of INQ is to provide highly efficient GPU-accelerated routines that can be used by other DFT codes that do not support GPUs natively. As shown in Figure 3, in this modality, users can run INQ through another code that acts as a front end, instead of running it directly. The advantage would be that higher-level functionality can be directly implemented on top of INQ and that users can use INQ through a familiar interface.

All of the source code for INQ is freely accessible online from our GITLAB website https://gitlab.com/NPNEQ/inq/. The code is released under the Lesser General Public License version 3 (LGPLv3). This is an open-source license that allows anyone
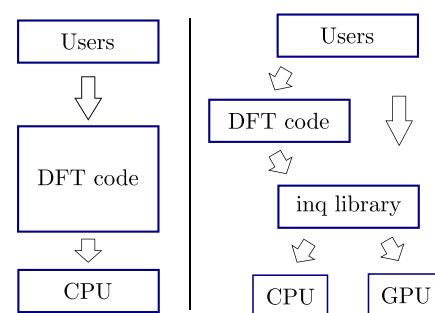

Figure 3. Different paradigms of numerical simulation in relation with the usage and development. On the left, a traditional paradigm: the users interact with a monolithic code. The code defines its own interface and uses the underlying hardware. Even if some users become developers themselves, the task of developing and the task of usage are clearly separated. Each new development has to be reflected in the interface (e.g., input format) for the users. This has the advantage that the monolithic code has stability and gives well-defined access to functionality at the cost of development turnaround and efficiency, e.g., postprocessing and interaction with other codes are always a separate task. On the right, the proposed paradigm: the system is prepared in such a way that users have access to low-, intermediate-, or high-level functionality. The resulting library offers a trade-off between the level of access and the level of difficulty (knowledge on the part of the user). The library hides for the most part specialized interaction with the hardware (e.g., CPU or GPU) but no more. Simple or complex programs can be written in the same language than the rest of the library. Optionally, a more traditional "DFT code" can be adapted on top of the INQ library, allowing the DFT code to use the high-performance components and GPU support of INQ while retaining the traditional interface and capabilities.

access to the source code of INQ and works that derive from it. We think that this is important to ensure the reproducibility and transparency of the results and to ensure open access to science.[71,72] Note, however, that this specific license allows other codes to link with INQ as a library independently of their license.

As developers, we are aware that the installation of a scientific code can become cumbersome. To make it easier for users to compile INQ, we aim to provide an easy-to-use build system that works out of the box in most cases. The build system for INQ is based on CMAKE, but we use a wrapper script to offer the familiar `configured` script. We limit the number of library dependencies to the standard ones for scientific codes and include in the software package libraries that would be difficult for the users to obtain and compile.

IV. MODULAR STRUCTURE OF INQ

The implementation of INQ is based on a modular design. We have identified from the code some components that perform different tasks and turned them into independent components. It is remarkable how modern coding techniques can make several separate libraries work together with absolutely no code interdependence. This is achieved by writing code against "concepts" or interfaces rather than leaning on specific implementations.[73]

This movement toward libraries rather than monolithic codes is slowly picking up in the community. For example, the CECAM Electronic Structure Library (ESL)[74] is a collection of modules for electronic structure calculations written in several languages to help reduce duplicate effort.

The main components of INQ are shown in Figure 4. In the following subsections, we briefly describe these components. In
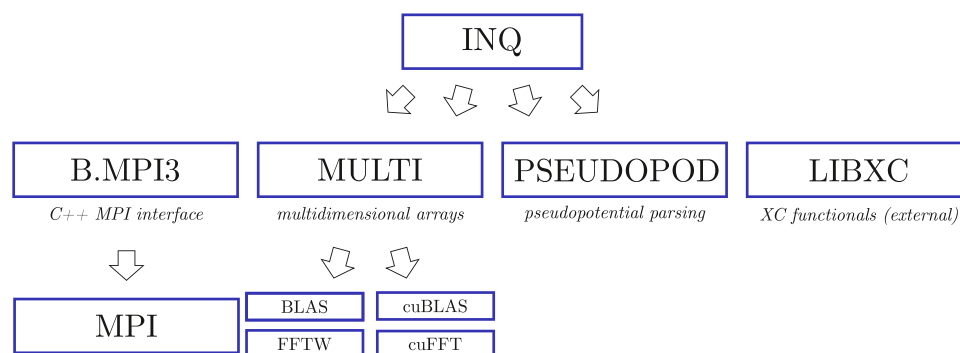
Figure 4. Simplified architecture of the INQ code and the main library dependencies. The first layer of libraries corresponds to facilities directly related to the implementation of DFT and TDDFT. The facilities are understandable for a professional researcher in the theory of electronic structure: access to different exchange and correlation functionals and pseudopotential information. Various array representations of states, and spatial discretizations, high-level linear algebra operation, like orthogonalization, application of Hamiltonian, and representation of potential fields and the communication patterns. At the lowest level, we have very specialized libraries, usually vendor-specific, that can be switched depending on the platform.

the future, we plan to integrate other components, like SPGLIB,[75] to handle symmetries and k-point generation. We also plan to split other parts of the code into independent libraries once they have a mature enough interface.

IV.I. MULTI: Multidimensional Arrays. An essential component needed to implement DFT in a simple and elegant way is multidimensional arrays. The type of basis that defines regular grids in real and reciprocal space and the utilization of massive GPU parallelism make regular contiguous arrays the ideal data structure to perform numeric operations. Historically, C++ does not provide any particular kind of multidimensional arrays, leaving to the user the option to implement or use different libraries that emulate multidimensional indexed access. We found that no pre-existing library was particularly suitable for use in the GPU and with a sufficiently modern design that matched our needs and therefore choose to implement it ourselves.

MULTI provides multidimensional array access and description of the data layout in memory, in both CPUs and GPUs. Its key goal is to abstract as much as possible operations over arrays and make it easier to write algorithms without compromising maximum performance. The library also provides interoperability with existing libraries, in particular, numerical ones such as linear algebra (basic linear algebra subprograms (BLAS)-like) and fast Fourier transforms (FFTs),[76] and the C++ Standard Library (STL) algorithms.

MULTI is currently heavily used in the INQ implementation and it is its main internal data structure. Specifically, electronic state data (complex coefficients) is viewed simultaneously, depending on the part of the code, as two-dimensional (2D) or four-dimensional. Three-dimensional (3D) subarrays can represent the spatial (volumetric) structure of the problem in either real or reciprocal space. This referential manipulation of arrays (also known as "views") allows us to minimize copies and especially avoid copies between GPU and CPU.

In addition, arrays can support different memory spaces (e.g., GPUs) through custom pointers. The use of custom pointers allows us to syntactically separate (at compilation-time) GPU and CPU memories and their corresponding operation (e.g., choose between a GPU and CPU code to dispatch an abstract operation). Finally, the library is responsible for memory allocations, which are notoriously slow in the GPU. In INQ, the library allows reducing the number of allocations by optionally managing internal blocks of preallocated memory.

The fact that this library is used in a completely different HPC code, QMCPACK,[77] is witness to the general purpose aspect of this library.

IV.II. B-MPI3: Message Passing for C++. INQ achieves distributed-memory parallelism (parallelism between several nodes) using the message passing interface (MPI) infrastructure, including extensions that allow direct GPU to GPU data transfer. The problem with standard MPI is that its interface can be quite cumbersome to use since it requires a large amount of user-provided information (explicit types, data pointers, layout) even for constructing simple messages. To avoid this issue, the low-level MPI calls in INQ are wrapped by a higher-level library called B-MPI3.

B-MPI3 is a C++ library wrapper for version 3.1 of the MPI standard that simplifies its utilization, maintaining a similar level of performance. We find this C++ interface more convenient, powerful, and less error-prone than the standard MPI C-based interface. For example, pointers are not utilized directly and it is replaced by an iterator-based interface and most data, in particular, arrays and complex objects are serialized automatically into messages by the library. B-MPI3 interacts well with the C++ Standard Library and containers and can take advantage of the aforementioned MULTI array library. The library is general purpose and, as a standalone open-source project, it can be reutilized in other scientific projects, also including QMCPACK.[77]

IV.III. PSEUDOPOD Pseudopotential Parser. Pseudopotentials are an essential part of electronic structure codes that rely on uniform representations like plane waves or real-space grids. They provide two benefits: make the wave functions smoother around the nuclei and avoid the explicit simulations of the core electrons. There are many types of pseudopotentials, and even within each type, there are multiple ways of generating them. This means that users normally have to carefully select the pseudopotentials they want to use from dozens of options.

To make things worse, pseudopotentials come in many different file formats that are not compatible with each other. The idea of INQ is to support as many formats as possible instead of introducing a new one, as this gives the largest possible flexibility for users. This means we need to know how to parse most pseudopotential formats. This is a task that is fairly disconnected from the rest of a DFT code, which only needs to access the pseudopotential information in memory. Based on this, we decided to make an independent library, named PSEUDOPOD, that could take care of this task and other pseudopotential-related functionalities.

The goal of PSEUDOPOD is to provide a fully standalone library that can be used for other electronic structure codes. PSEUDOPOD is based on code written for OCTOPUS and QBALL, and as such, it has been tested quite extensively. It provides INQ and any code that uses it, a unified interface, independent of the file format, to access the information in pseudopotential files. This interface is, in addition, GPU-aware, so that the construction of the potential and projectors by the calling code can be done directly from a GPU kernel.

PSEUDOPOD can parse several popular formats like UPF1 and 2 (QUANTUM ESPRESSO (QE)), PSML[78] (SIESTA), psp8 (ABINIT), and QSO (QBOX). This includes Optimized Norm-Conserving Vanderbilt (ONCV) pseudopotentials.[79] The formats based on XML files are parsed using the RAPIDXML library. (RAPIDXML is a simple library that is distributed with PSEUDOPOD, so the users need not install it separately.)

PSEUDOPOD also has additional functionality, as it includes routines to filter pseudopotentials to remove high-frequency components and avoid aliasing effects,[11,80] something that INQ uses and that is essential for real-space DFT codes. It also provides auxiliary routines like Bessel transforms, range separation, and spherical harmonics that codes might need to process and apply pseudopotentials.

Finally, a very important functionality of PSEUDOPOD is to handle pseudopotential sets. Pseudopotential sets are a relatively recent development that makes it much simpler and reliable to run DFT calculations. Several research groups have provided a collection of pseudopotentials for most of the elements in the periodic table,[81−88] which have been curated and validated. This means that users do not need to select individual pseudopotentials but they can just pick a consistent set of them.

PSEUDOPOD contains as part of its files two of these sets: Pseudodojo[88] and SG15;[86] so they can be used directly by the code without needing to download separately by the user. The calling code only needs to select the set and requests the pseudopotential for the specific element.

IV.IV. LIBXC and GPU Support. LIBXC is a standalone library of exchange and correlation (XC) functionals initially developed by Marques.[89,90] We rely on LIBXC to calculate the XC energy and potential for DFT calculations. This refers to a number of programmed analytic functions that map values of the density and density gradients to energies (and higher derivatives thereof). This is a part of an electronic structure code that requires a lot of code and that is quite cumbersome and error-prone to implement. The effect of adopting LIBXC can be seen in Figure 1; it shows a big drop in INQ's line of code count early on when we started to rely on this external library instead of a few internal functionals.

One previous limitation of LIBXC is the lack of native GPU support. In general, the cost of the XC evaluation is comparatively small in a DFT code, so the computational cost difference is not very large. However, a CPU-only implementation forces the code to copy data back and forth between the CPU and the GPU, which is expensive. So, for an efficient GPU code, we need to evaluate the XC functional on the GPU.

As part of the development on INQ, we implemented a GPU version of LIBXC based on CUDA and contributed it back to the official version of the library. Our implementation makes minimal changes to the library and relies on CUDA "unified memory" to store the internal data structures on LIBXC.

V. MODERN C++ PROGRAMMING

Ideally, a programming language should allow us to express the physics clearly, without exposing too much the implementation details or the underlying data structures. At the same time, it should be efficient so that the programmer has control of what the low-level code is doing and it does not introduce spurious operations. Additionally, the availability of compilers, portability, and compatibility with GPU computing are key to this project. Considering all of these factors, we decided to use C++ for INQ.

C++ is an evolving programming language that is especially attractive to create high-performance applications. Due to its legacy, it is a system language that allows us to control low-level hardware operations while at the same time offering various abstraction mechanisms to represent complex problems. Separate compilation and linking allow interoperability with hardware, vendor-specific libraries, as well as abstraction libraries. The combination is ideal for simulation problems that require high performance but also require expressing simulation steps with reasonable simplicity.

In C++, there is no single way to use the language; a rather complex syntax allows paradigms to coexist in a single unit of code, usually correlated with the level of abstraction. The absence of a single paradigm requires certain discipline, which we describe below as modern techniques. Techniques change as the language evolves and the community continues research. Yet, some characteristics tend to stick as they are deemed to produce good code. Here, we mention some of these techniques that are relevant to this simulation code, in particular.

A simulation code usually deals with the manipulation of a simulated system whose state is represented by a set of program variables stored in memory. Historically, academic codes tend to make this state global to the program since usually only one system is simulated at a time. This state is usually represented by global variables or variables that are designed to live for the whole duration of the program execution. A global state makes it easy to access the data of the simulation from any place in the program; it makes the code very easy to change and reduces the number of arguments taken by functions that otherwise run in the tens of arguments. In addition, the memory associated with the system does not need to be managed by any special mechanism; in the worst case, memory is released by the operating system itself once the program shuts down.

Global state has a long-term maintenance cost though; the program becomes hard to reason about because changes to the state of the simulation can be made by any part of the code in an inconsistent way, by parts that can be far removed from each other. Global state also implies the existence of a single simulation object at a time; generalizing the code to simulate several systems simultaneously is hard to achieve or requires the use of isolated instances of the program.

As explained, we designed the code to be run as a standalone code or as a library. As a library, the constraint of a single simulated object is not natural and overly restrictive. Besides, a library needs to interact with other unknown-in-advance parts of the program and therefore cannot take over all of the resources or memory to itself. Therefore, we rejected the idea of a single global simulated system from the start of our design.

It turns out that this fundamental decision led us to use other interesting techniques. First, since simulated systems had to be manipulated by functions, and there is no single instance of it, functions have to take the simulation variables explicitly.

Therefore, sets of variables associated with a system are naturally bunched together as objects or user-defined C++ classes. Also, objects can be more easily protected against inconsistent modifications (or against modifications at all) by certain functions if they do not naturally need to modify the state.

In all realistic settings, computers have limited resources. Each object or subobject requires controlled access to resources. Since an object's lifetime is not indefinite, in general, in our programming paradigm, the request for resources should be accompanied by release when the resource is no longer needed. It is key to recognize that resources include more than just memory; it includes file handles, MPI communicators, threads, mutexes, GPU memory, or anything that is limited in a computer. Any resource can leak if not properly managed. Ideally, this management should be automatic and not explicitly coded.

C++ can emulate automatic and deterministic management, by which any resource that needs explicit control can be requested on the construction of a manager object and released upon its destruction; after that, most resources are managed automatically by scopes or logical units of the code (such as functions or other higher-level classes).

Finally, in scientific programming, the code is an expression of equations and mathematical operations of a certain model or theory. Ideally, the code should represent as closely as possible the mathematics to make the software simple to understand, write, and verify. Unfortunately, this is not always possible to achieve as programming languages offer limited expressiveness and when they do, they usually come with a considerable performance overhead. The key point is that performance (or scalability or computational complexity) itself cannot be abstracted away or hidden from the user. In modern C++, it is possible to design the code such that simple expressions can be written without penalizing performance. For example, in INQ, the application of the Hamiltonian, and other operators, in the code looks as simple as

```
auto hphi = hamiltonian(phi);
```

while in most DFT codes, the Hamiltonian operator (if it exists as such) is a complicated function with many arguments. Moreover, the above syntax achieves optimal performance without unnecessary copies, an issue that plagued old ways of utilizing the C++ language.

Such simplicity allows the developers to focus on writing algorithms and equations and not on the intricacies of the specific data structures in the code. Additionally, it is a polymorphic code that can work with any type of Hamiltonian operator. The previous line is an example of code that is written with a "concept" in mind, rather than a specific implementation of the `phi` or `Hamiltonian` objects. For example, in INQ, many algorithms are tested with a simple "Hamiltonian" given by a simple dense Hermitian matrix, using exactly the same routine that works for any type of operator. The idea is that as long as the operation makes sense syntactically and that reasonable developer(s) of the code agree on the semantic meaning of an operation, a function can be written simultaneously for different classes that share the same conceptual meaning. This type of code becomes a template, a fundamental C++ feature, that can be compiled into very different machine code, even using different underlying numerical libraries, but still representing the same conceptual operation. C++ templates not only make code more generally applicable but also have the side effect of "late binding". This is a

feature by which the actual machine code is compiled only when all information about the code is available. More importantly, template functions can have injected code, allowing the compiler to optimize code across function boundaries, including inlining. These ideas are part of a very productive trend in C++ called generic programming, which is based on a well-defined mathematical reasoning.[91]

VI. GRAPHIC PROCESSING UNITS

Programming on GPU presents some unique challenges for code developers. The main ones are that the GPU has its own memory space where data must be stored prior to use and that the code has to be explicitly parallelized. Because of this, it is challenging to adapt an existing code to run on the GPU, as extensive modifications to the whole code are needed. This is one of the reasons we decided to start a new code, designed from the ground up to run on GPUs (but that also runs on CPUs). This effort is guided by our previous experience in the GPU port of the code OCTOPUS.[56,62,92]

Since GPUs from different vendors are available or will be available in the near future, it is essential to design a code that is portable and that ideally is performance portable. That is, the code can run on different platforms and moreover it can run efficiently on all of them without extensive retuning.

With this objective in mind, we studied the different available platforms for high-level GPU programming like RAJA,[93] KOKKOS,[94] OPENMP,[95] and SYCL.[96] Unfortunately, we could not find the one that was directly suitable for the operations needed for DFT/TDDFT and that was mature enough at the moment we started the project (mid-2019). So we decided to design our code using the CUDA C++ extensions with a thin layer of abstraction on top, which allows us to make most of the code independent of the GPU backend. This layer has two components that take care of different tasks; a scheme of this approach is shown in Figure 5. The first component is the MULTI library that takes care of allocations, array copies and transpositions, and GPU-accelerated libraries for linear algebra
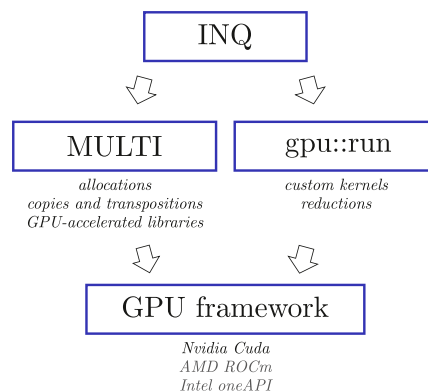

Figure 5. Two components that INQ uses to run GPU code. MULTI is an array library, which, besides the multidimensional indexing abstraction, takes care of memory allocations and memory copy and provides an interface for lineal algebra and fast Fourier transforms. `gpu::run` is a simple abstraction layer to run kernels and perform reductions on the GPU. This layer of abstraction allows us to write code that is agnostic to the specific processors that are used. Right now, we are based on CUDA, but in the near future, we plan to support the code on AMD and Intel GPUs. Both MULTI and `gpu::run` also support execution on the CPU, so we do not need to write duplicated code for each processor type.

and FFTs. The second one is a routine we wrote called `gpu::run`. It executes kernels on the GPU and adds some extra functionality like the calculation of reductions. To adapt INQ to other GPUs, we will just need to extend these two components and not the whole code.

To simplify the code and its development, we use what Nvidia calls managed memory, a type of memory that can transparently be accessed from the CPU and GPU. However, a large performance price is paid when the memory is accessed (even inadvertently) from the wrong processor. Our memory management philosophy is to assume that all of the data will be kept in the GPU memory. We think this is reasonable since the amount of memory on the GPU is increasing substantially and that the individual GPU memory requirements can be reduced using more MPI nodes with GPUs. This contrasts with the upgrade strategy followed by many CPU legacy codes, in which some identified tasks are delegated to the GPU together with a two-way copy of data from CPU to GPU and back.

In the code, we use MULTI arrays that are allocated in CUDA-managed memory, so that they can be accessed from both CPU and GPU codes. This completely hides both the allocation and memory location in most of the high-level code. However, our experience in the optimization of the code shows that managed memory can slow down significantly GPU kernels accessing large memory blocks, memory that has been just allocated. To avoid this problem, it is necessary to selectively use a "prefetch" call after allocation to speed up kernel execution considerably.[97]

The MULTI library offers generic operations like matrix multiplications and FFTs that, behind the scenes, are dispatched by the compiler to the corresponding library. For example, BLAS for the CPU data and CUBLAS for (Nvidia) GPU data. This allows us to abstract a large part of the operations and make them run as efficiently as possible using vendor-optimized libraries under a common syntax.

There are, however, several operations in a DFT code that are quite specific and are not available from external libraries. For them, we have to write our own GPU kernels in the CUDA language, which is an extension of the C++ language. We use modern C++ to do this in a simple, readable, and portable way.

C++ offers a nice way of defining functions locally (inside another function) called lambdas. These lambdas can "capture" local information and they can be passed to other functions, such as GPU kernels. Based on this functionality, we wrote a simple function to execute lambdas on the GPU called `gpu::run`. The idea behind this function is that it can replace loops in a way that is similarly readable. Unlike the C++ STL, `gpu::run` is oriented to the index execution pattern in multiple dimensions and not to one-dimensional (1D) data structures.

This is a simple example of `gpu::run`. Consider the following double loop that is how a standard CPU code would be written

```
for(int i = 0; i < n; i++){
  for(int j = 0; j < m; j++){
    c[i][j] = a[i][j] + b[i][j];
  }
}
```

With `gpu::run`, we would write this code like this

```
gpu::run(
  m, n,
  [...](auto j, auto i){
    c[i][j] = a[i][j] + b[i][j];
  }
);
```

(Where the lambda "capture" arguments [...] are omitted for brevity.)

The number of size arguments (2 in this case) determines how many loops there are and what are the corresponding index ranges (0 to m and 0 to n in this case). The last argument is a lambda function that indicates the work that each iteration does. When compiling for the CPU, the code above is interpreted as normal loops, just like the original code. When compiling for the GPU, the lambda is executed in parallel inside a GPU kernel instead of loops.

Many operations can be written in this fashion, such as element-by-element transformations or copies. The main exception happens when there are sums (or in general reductions) inside some of the loops. So we have implemented a special case of `gpu::run` for these cases.

Take, for example, the case of a matrix multiplication. This can be written in a loop like this

```
for(int i = 0; i < n; i++){
  for(int j = 0; j < m; j++){
    c[i][j] = 0.0;
    for(int l = 0; l < k; l++){
      c[i][j] += a[i][l]*b[l][j];
    }
  }
}
```

Notice that there is a sum in the loop over k. In that case, to use the version of `gpu::run` we just showed, we would need to include the loop with the reduction inside the lambda, like this

```
gpu::run(m, n,
  [...](auto j, auto i, auto l){
    c[i][j] = 0.0;
    for(int l = 0; l < k; l++){
      c[i][j] += a[i][l]*b[l][j];
    }
});
```

The problem with this strategy is that it is not very efficient on a GPU in the case when m and n are relatively small compared to k. So we need a more general solution that also parallelizes the loop over k on the GPU.

In general, performing reductions in parallel over GPU code is not simple since horizontal communication between threads is not straightforward. The main strategy for reductions is to perform a tree-based approach,[98] even though more advanced hardware-dependent strategies exist. Our approach in INQ was to implement this reduction algorithm and expose it through a special `gpu::run` interface. This would make the matrix multiplication look like this

```
c = gpu::run(m, n, gpu::reduce(k),
  [...](auto j, auto i, auto l){
    return a[i][l]*b[l][j];
  }
);
```

As before, the m and n are normal iterations, while the `gpu::reduce` tag around k indicates that a reduction should

be performed over this dimension (third index). The result of the reduction is returned as an array c of dimensions n by m. Note that in this case the lambda must return a value for each iteration; this is the value that will be accumulated.

This strategy allows us to easily write operations that otherwise are complicated to implement on the GPU. Plus, they hide the reduction algorithm that could be optimized depending on the specific hardware.

Please note that we used the matrix multiplication just as an example in this section. In INQ, we always use gemm for matrix multiplications, as it is much more efficient as it is optimized by the vendor for each platform (e.g., as part of CUBLAS). The main objective of gpu::run is to allow us to quickly write GPU code for many operations. Some kernels that are critical for performance might need further optimization.

It is well known that allocation (explicit request of memory) and deallocation (release of memory) of GPU memory are relatively time consuming. Therefore, it is paramount to avoid these types of requests in performance-critical parts of the code. A naive approach, such as making the variables live in larger scopes, or even making them global, can spoil the architecture of the program. Instead, INQ adds another layer of abstraction by utilizing custom memory allocators that can optimize memory management for particular use patterns. Specifically, large chunks of memory can be recycled across different objects avoiding the costly system-level release of memory.

VII. DISTRIBUTED-MEMORY PARALLELIZATION (MPI)

If we want to model large systems, it is essential to use distributed-memory parallelization, where the code runs simultaneously over multiple nodes at the same time (up to hundreds of thousands in some cases). For this parallelization, we use the standard message passing paradigm, where each process has its own data and can exchange information with other processes.

The simplest and more efficient method of parallelization for message passing is to distribute the data among processors, with processors executing mostly the same code (single-program multiple data).[99] It allows us to increase the number of processors as the problem size increases. The challenge is to use an optimal data distribution to avoid communication as much as possible. In the case of INQ, this means a distribution of the arrays that is optimal for parallel FFTs (this is explained in detail in Section VIII.III).

DFT and TDDFT have traditionally performed very well in CPU-based parallel supercomputers.[20,61,66,100,101] However, GPUs present additional challenges for parallel programming. While GPUs can compute much faster than a CPU, network communication hardware has not seen a similar jump in performance. This means that the relative cost of communication with respect to computation has increased in GPU-based supercomputers. On top of that, communicating data between GPUs through MPI can be challenging to implement for vendors.

In INQ, we directly call MPI over data in GPU memory. Ideally, most modern MPI implementations are GPU-aware and can recognize the GPU memory space and they can directly access data. In the best scenario, they can also communicate the data to and from GPU memory without passing through the main memory, but this is not always the case. For non-GPU-aware implementations, managed memory residing on the GPU would be automatically copied to main memory before MPI calls.

Unfortunately, this last case has a non-negligible overhead in the communication cost.

VIII. DFT AND TDDFT IMPLEMENTATION

In this section, we describe the physical model that INQ simulates, the solution algorithms, the main data representation, and the computational kernels we need to use.

The density functional framework provides a way to calculate the density, and other observables, of an interacting many-body system using a non-interacting system as reference.[4] The main quantities are then the density n and the set of Kohn–Sham (KS) orbitals (or states) φ_k of the reference non-interacting system. Mathematically, these objects are fields or functions: they have a value associated with each point in space. The density is generated from the occupied orbitals by the formula

$$n(\mathbf{r}, t) = \sum_k |\varphi_k(\mathbf{r}, t)|^2 \tag{1}$$

The operator that generates the dynamics of φ_k is the KS Hamiltonian (in atomic units)

$$\hat{H} = -\frac{1}{2}\hat{\nabla}^2 + \sum_I \hat{v}_I(\mathbf{r} - \mathbf{R}_I) + v_{\text{pert}}(\mathbf{r}, t) + v_{\text{h}}[n](\mathbf{r})$$
$$+ v_{\text{xc}}[n](\mathbf{r}) \tag{2}$$

The first term is the Laplacian operator that represents the kinetic energy. The second term is the sum of the potentials of the ions, at positions $\mathbf{R}_I$, represented by a nonlocal pseudopotential. v_{pert} represents a possible time-dependent perturbation, for example, an external electric field. The two last terms $v_{\text{h}}[n](\mathbf{r})$ and $v_{\text{xc}}(\mathbf{r})$ mimic the electronic interaction in the density functional approach. They are, in principle, functionally dependent on the density at all points and, for the time-dependent case, at all past times.[4]

The actual dynamics of the electrons is obtained from the time-dependent KS equation

$$i\frac{\text{d}}{\text{d}t}|\varphi_k(t)\rangle = \hat{H}|\varphi(t)\rangle \tag{3}$$

Note that because of the density dependence in $\hat{H}$, this equation is coupled with eq 1, so both must be solved self-consistently.

In the stationary case, eq 3 yields the ground-state KS equation[2]

$$\hat{H}|\varphi_k\rangle = \varepsilon_k|\varphi_k\rangle \tag{4}$$

that minimizes the energy

$$E = \sum_k \varepsilon_k - \frac{1}{2}\int \text{d}\mathbf{r}\, n(\mathbf{r})v_{\text{h}}(\mathbf{r}) + E_{\text{xc}}[n] - \int \text{d}\mathbf{r}$$
$$n(\mathbf{r})v_{\text{xc}}(\mathbf{r}) \tag{5}$$

The real-time equation for the electrons (eq 3) can be coupled with a set of classical equations of motions for the ions under the force generated by the electrons. This produces Ehrenfest dynamics[102,103] that approximates the nonadiabatic dynamics of the system.

Our idea is to provide a computational toolkit that allows not only to solve eqs 3 and 4 but also to perform all of the operations that appear in the density functional approach as distinct code routines. All of these routines are well tested and implemented so that they can execute efficiently and transparently on CPUs, GPUs, and MPI parallelization. This will allow INQ developers

and users to implement new functionalities and test new theories in a simple-to-use and fast framework.

The main choice for a density functional code is to select a basis where the orbitals, densities, and other fields are going to be represented by a finite amount of data. In INQ, we use the popular plane-wave approach, which, despite the name, actually uses two representations. One is the basis of plane waves or Fourier space (FS), while the second one is a real-space grid. This is advantageous because the Laplacian operator is diagonal in Fourier space, while the potential terms are local or semilocal in real space and because we can go efficiently between the two representations using FFTs. Of course, other alternatives exist for the discretization of the equations. In chemistry, the dominant approach is to expand the fields into a set of atomic orbitals, usually represented by Gaussian wave functions.[104,105] In real-time TDDFT, it is quite common to use real-space grids where the Laplacian operator is approximated by high-order finite differences.[10]

VIII.I. Ground-State Solution. To obtain the ground state of an electronic system in DFT, we need to solve the KS equation (eq 4) numerically. We follow the standard approach of directly solving the nonlinear eigenvalue problem through a self-consistent iteration that deals with the nonlinearity, which appears through the density dependency.[12] Each iteration, we need to solve a linear eigenvalue problem for the Hamiltonian operator for a given guess density. The solution gives a new density that is mixed with the previous guess density as a guess for the next iteration. When the input and output densities are sufficiently similar, the solution is converged.

The mixing of the density in each step is relatively simple to implement and does not require much computation while, at the same time, stabilizing the convergence of the iterative solution. In INQ, we use the Broyden method[106] by default. The code also implements (static) linear mixing and Pulay mixing[12,107] as alternatives.

The computationally expensive part comes from the solution of the eigenvalue problem. Since the dimension of our space can be very large, it is not practical to use a method that directly works over a dense matrix (e.g., the basis representation of $\hat{H}$). Instead, we use eigensolver algorithms that only need the application of the operator over trial vectors.[108] These methods are usually called iterative since they progressively refine approximate eigenvectors until convergence is achieved. (However, it must be noted that as a corollary of the Abel–Ruffini theorem all eigensolvers, including the dense-matrix ones, must be iterative in some sense.[109])

In practice, it is not necessary or convenient to fully solve the eigenvalue problem in each self-consistency iteration. Instead, a few iterations of the eigensolver are done at each step, so eigenvector and self-consistency convergence are achieved together.

The eigensolver we use in INQ by default is the preconditioned steepest (SD) descent algorithm. This is a simple method that has the advantage that it can work simultaneously and independently on all eigenvectors, which is particularly useful for GPUs, as it makes a lot of data parallelism available. (We are working on the implementation of the RMM-DIIS method[12] that converges faster than SD and is also highly parallelizable.) We also implement the conjugate gradient[110] and Davidson[111] eigensolvers; however, they do not perform as efficiently as SD at the moment. For preconditioning, we use the method of Teter et al.[112] that is applied in Fourier space.

As part of the eigensolver process, we need two operations that involve rotations in the space of eigenvectors: orthogonalization and subspace diagonalization.[12] The cost of these procedures is dominated by linear algebra operations that scale cubically with the number of atoms, while other operations are quadratic. As this linear algebra becomes dominant for large systems, it is important to perform them efficiently. In the serial case, this is not complicated to do, since highly optimized versions of BLAS[113] and LAPACK[114] are available for the CPU and GPU. For the parallel case, the situation is more complicated at the moment. While the SCALAPACK[115] library provides a set of parallel linear algebra routines, it only runs on CPUs. The SLATE library,[116] currently under development, in the near future will provide a modern replacement for SCALAPACK. Until SLATE is complete or another library becomes available, INQ cannot fully run in parallel for ground-state calculations. Only domain parallelization is available.

An important aspect of the solution of the ground state is selecting an appropriate initial guess for the density and the orbitals. The density is initialized as a sum of the atomic density of the atoms obtained from the pseudopotential files. The orbitals are initialized as random values for each coefficient in real space, uniformly distributed in a symmetric range around 0. We have found that this produces orbitals that are linearly independent and that contain components of the ground-state orbitals.

However, special care must be taken when running in parallel, both in the GPU and in MPI, with the generation of random numbers. If each parallel domain uses a different random number generator, the initial guess would depend on the number of processors used, making it difficult to get consistent and repeatable results. (And this is without even considering the issue of correlation between generators, which can happen in parallel.) Our solution was to use a permuted congruential generator (PCG)[117] that can be fast-forwarded by skipping steps in logarithmic time. So, when running in parallel, for each point, we can forward the generator to obtain the same number it would have in the serial case. This approach produces random orbitals that are exactly the same independently of the number or processors used and independently of whether we are running on the CPU or GPU. The overall cost of the randomization is quasi-linear, and in practice, we see that it is negligible compared to other operations. For the implementation, we use the small library provided by the author of PCG, which we modified to run on GPUs.

Having a consistent starting point that does not depend on the parallelization is essential for testing. Otherwise, it would be very hard to determine whether differences in the results come from errors in the MPI or GPU parallelization or from the differences in the starting guess.

VIII.II. Real-Time Propagation. Time propagation is used to calculate excited-state properties by following the real-time dynamics of the electrons under external perturbations. It can be used to calculate a large number of linear and nonlinear response properties. It requires the integration in time of the time-dependent KS equation (eq 3).

How to perform the real-time propagation efficiently is an area of active research, and many methods have been proposed.[64,118−123] We use the enforced time-reversal symmetry (ETRS) propagator with the exponential approximated by a first-order Taylor expansion.[118] This is a quite popular method due to its efficiency, numerical stability, and relatively simple implementation. However, we have introduced an additional

"trick" with respect to previous implementations that allows us to reduce the computational time by 33%.

In ETRS, the propagator is

$$|\varphi_n(t + dt)\rangle = \exp\left(-i\frac{\delta t}{2}H(t + dt)\right)\exp\left(-i\frac{\delta t}{2}H(t)\right)|\varphi_n(t)\rangle \tag{6}$$

Since $H(t + dt)$ is not known due to the nonlinearity of the equations, we need to approximate it. This approximation is usually obtained from performing a full-step propagation of the states

$$|\varphi_n^{\text{approx}}(t + dt)\rangle = \exp(-i\delta tH(t))|\varphi_n(t)\rangle \tag{7}$$

From $|\varphi_n^{\text{approx}}(t + dt)\rangle$, we then get an approximation for $n(\mathbf{r}, t + dt)$, and from there, $H(t + dt)$.

Hence, for each step of ETRS, we need to calculate three exponentials, two in eq 6 and one in eq 7. The important detail is that two of these exponentials, $\exp(-i\delta/2tH(t))|\varphi_n(t)\rangle$ and $\exp(-i\delta tH(t))|\varphi_n(t)\rangle$, have a very similar form. They only differ by a factor of 2 in the exponent.

Let us consider the truncated Taylor approximation of the exponential of an operator multiplied by a scalar

$$\exp(\lambda A)|v\rangle = \sum_{k=0}^{4} \frac{1}{k!}\lambda^k A^k|v\rangle \tag{8}$$

When we evaluate this expression numerically, the expensive part is $A^k|v\rangle$. It is easy to see then that we can evaluate this expression for several values of λ for almost the same cost of a single value, since λ only appears as a multiplicative coefficient. In practice, this means we can calculate together two of the three exponentials needed for ETRS, reducing the cost to only two exponentials per step.

For Ehrenfest dynamics, we propagate the ions with the velocity Verlet[124] algorithm using the forces given by eq 11. Both electrons and ions must be propagated consistently to ensure that the ionic potential is evaluated at the correct time in eq 6. Otherwise, the time-reversal symmetry is broken and a drift appears in the total energy.

The propagation of the ion that coordinates itself is not numerically expensive. However, it is the calculation of the forces from the electronic state and the recalculation of the ionic potential that can be the most time consuming when including ion dynamics. For these reasons, these operations must be implemented efficiently and run on the GPU.

One significant advantage of TDDFT is its potential for parallelization. The real-time propagation in TDDFT conserves orthogonality of the KS states mathematically and also numerically.[103] The first consequence of this property is that, unlike the ground state, we do not need the orthogonalization operation, which makes the overall cost of the time propagation quadratic with the size of the system instead of cubic. The second consequence is that the propagation of each state is independent of the rest, with the only "interaction" between them coming from the self-consistency of the equations. This means that it is very efficient to parallelize the propagation by distributing states among processors.[61]

VIII.III. Fields. A fundamental data structure in INQ is the `field` type, which represents a mathematical function or field in a finite domain. The information it contains is a basis and the coefficients of the field in that basis. Quantities represented by field are, for example, the density and the local potential.

A property of the basis used inside a field is the type of representation, for example, real space (RS) or Fourier space (FS). This type of basis is represented through C++ types, which allows us to define polymorphic functions that work over a `field`, independently of the basis on which it is represented. For example, the `gradient` function can both accept a `field` in RS or FS as input and perform the correct operation for each case. Of course, INQ provides functions to transform a field from one representation to the other by an FFT.

In both RS and FS, the coefficients in a field have a geometrical structure as a three-dimensional (3D) array. However, for many operations that do not rely on a geometric context, it is more convenient to access the coefficients as a one-dimensional (1D) array. Thanks to MULTI, a `field` object offers both a linear and 3D view of the coefficient data. Both representations share the underlying data so no copies are done, and the access is done without any overhead in both cases.

When running in parallel, a field is distributed by giving each processor a part of the coefficients. A large part of the operations we need to perform are local; they do not mix information from different coefficients. Integrals can be calculated efficiently by computing the local sums on each processor and then performing a reduction operation over all of the processors. For this, the field also includes a communicator that allows communication between all processors that contain parts of the field. The operations that really mix information from all points, and use the larger amount of communication, are the FFTs required to move from RS to FS and vice versa. It is the FFT operation then that determines the way we divide our points among processors so that communication is minimized.

A 3D FFT is calculated as a sequence of 1D FFTs in each direction. To minimize communication, we distribute the 3D grid into slabs by dividing one dimension evenly among processors, as shown in Figure 6. For an RS-field, the x-dimension is the one that is split. So we Fourier transform the y-
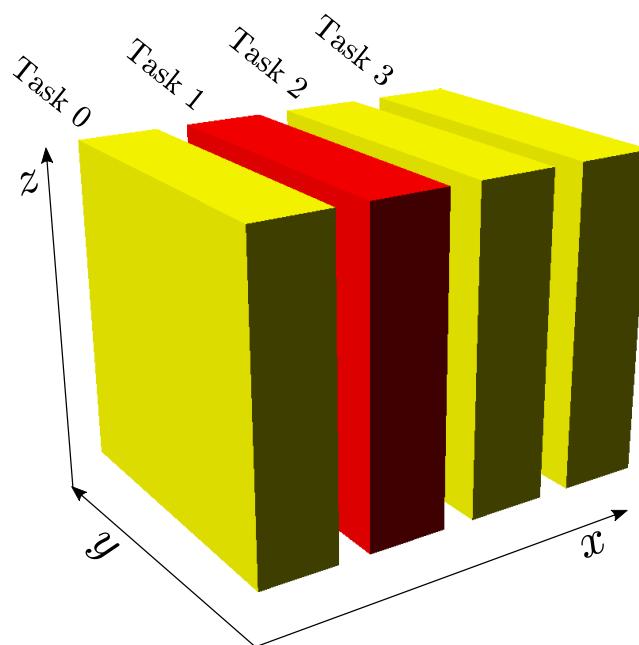

Figure 6. Distribution of the three-dimensional grid into slabs for optimal communication when performing FFTs in parallel. The order of spatial axis corresponds to a real-space field; in reciprocal space, x and z are interchanged.

and z-dimensions first as those transforms can be done locally. Now, we redistribute the array among nodes, so that the z dimension is split. Of course, this is the step that involves communication as all nodes need to exchange information. Once this is done, we can perform the FFT in the remaining x-dimension that now is local. Note that in the case of FS-field, we end up with a 3D grid that is split along the z-direction.

The approach we describe above is the standard approach for the parallelization of 3D FFTs and it is implemented for the CPU in FFTW and other libraries. For the GPU unfortunately, there is not a general MPI library yet that provides all of the required functionality for INQ, so we need to use our own implementation. The HeFFTe[125] library promises to fill this gap, and we expect to use it in the near future.

VIII.IV. Field Sets and Kohn–Sham States. One of the fundamental objects we need to represent in DFT is the KS states. As they are just a group of several fields, it would be natural to represent them as an array of field objects. However, this is not practical for numerical performance as we will end up with separated blocks of memory. In code, it is much more efficient to operate over data all at once instead of one by one,[126] especially on the GPU.[56,92]

Instead, we define a specialized field-set data structure that represents a group of several fields. As well as the basis information, it contains the coefficients for the whole set stored in a single array. This array can be seen as two-dimensional (2D), or equivalently a matrix, where one index corresponds to the coefficient index and the other to the index in the set. This representation is particularly useful for linear algebra operations that can be done directly using MULTI. The array of coefficient can also be accessed as a four-dimensional array with indices corresponding to the three spatial dimensions of the coefficients plus the set index. We use this representation for operations with geometrical context including FFTs.

The parallelization strategy for a field set is to use a 2D decomposition of the coefficient array. The coefficient dimension is divided in the same way done for `field`. The second division is done along the set indices, distributing the states among processes. In practice, we are giving each processor a 2D block of the orbitals, as shown in Figure 7.

Using such a decomposition has several advantages. In the first place, as the system size grows, both the number of states and the number of basis coefficients grow linearly. So, a two-level parallelization can keep the amount of data local to each processor constant, if the number of processors is grown accordingly (this is the weak-scaling scenario). Second, each parallelization level is naturally limited by either cost of communication or Ahmdal's law. By combining multiple levels of parallelization, we can use a larger number of processors than with a single one (this would be the strong scaling case).

Most parallel operations are trivially parallelizable in one of these two levels while requiring communication in the other level. For example, a parallel FFT requires the mixing of coefficients, which implies communication, as discussed in Section VIII.III; however, it is trivially parallelizable for the different states. On the other hand, the calculation of the density, eq 1, is trivially parallelizable in RS points but requires a parallel reduction over states. The case where we need communication across both levels of parallelization is linear algebra operations that only appear for the ground-state case and that need to be handled by linear algebra libraries (as mentioned in Section VIII.I).
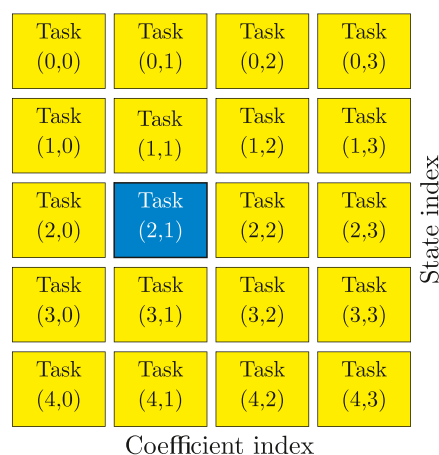

Figure 7. Representation of parallel distribution of the Kohn–Sham states in a two-dimensional (2D) data decomposition. We can consider the states as a 2D matrix where one dimension is the state index and the other is the basis coefficient index (where the three-dimensional indices have been flattened). In parallel, this matrix is distributed in blocks to the different processors. For convenience, we can think of the processors as arranged in a 2D grid, where each one is labeled by two coordinates that indicate which range of states they have and which range of coefficients. For example, if there are 400 coefficients and 50 states, task (2,1) (shown in blue) would have coefficients 200–299 from states 10 to 19.

VIII.V. Implementation of the KS Hamiltonian. The KS Hamiltonian, eq 2, is the main operation in the density functional formalism. Even though it is a linear operator (for a given density, in each self-consistency iteration), it would not be convenient to store it explicitly as a dense, or even as a sparse, matrix. Instead, we use it in operator form by having a routine that applies the Hamiltonian over a field set. This way, each one of the terms that appear in eq 2 can be calculated in the optimal way.

The kinetic energy operator, given by the Laplacian in eq 2, is calculated in Fourier space, where it is diagonal. The local potential is diagonal in real space instead. And the nonlocal part of the pseudopotential is also applied in real space, where the projectors are localized (this is discussed in detail in the next section). This means that the main operations inside the Hamiltonian are two Fourier transforms to switch back and forth between real space and Fourier representations using FFTs.

VIII.VI. Pseudopotentials. The current version of INQ uses Optimized Norm-Conserving Vanderlbit (ONCV) pseudopotentials.[79] In comparison to ultrasoft pseudopotentials[127] and projector-augmented waves (PAWs),[128] ONCVs are simpler to apply, do not introduce additional terms in the equations, and still produce accurate results.[63,81,88,129]

In INQ, the pseudopotentials are fully applied in real space in the Kleinman–Bylander (KB) form. In this form, there are two terms for the pseudopotential, the local part and a nonlocal part.

The local part is, in RS, a simple multiplicative potential that is applied together with the Hartree and XC potentials. Since the local part is long range, it needs to take into account the effect of all of the periodic replicas of the atoms when the system is periodic. To calculate it, we separate the local term into two terms, short range and long range, using the standard error function separation. The short-range potential is calculated directly in the grid. The long-range potential is given by the error function, which is generated by a Gaussian charge distribution. So, the solution of the Poisson equation for the Gaussian charges

of all atoms gives us the long-range part with the correct boundary conditions.

The nonlocal part requires a bit more attention to apply it efficiently. The KB separation yields a nonlocal potential of the form

$$v_{nl}\varphi_n(\mathbf{r}) = \sum_I \sum_{lm} \beta_{lm}^I(\mathbf{r} - \mathbf{R}_I) \int_{|\mathbf{r}' - \mathbf{R}_I| < R_c} d\mathbf{r}'$$
$$\beta_{lm}^I(\mathbf{r}' - \mathbf{R}_I)\varphi_n(\mathbf{r}') \tag{9}$$

where β_{lm}^I are the projector functions for each angular momentum component lm for each atom I. The important property is that the projectors are localized in space, so the integral in eq 9 can be done over a sphere around each atom instead of over the whole RS grid. This means that the cost of applying the pseudopotential is proportional to N_{atoms}, a scaling similar to the other components of the KS Hamiltonian.

The problem of applying the pseudopotentials in real space is that there is a spurious dependency of the energy with respect to the relative position of the atoms (point particles) and grid points. This is known as the egg-box effect. The cause of the problem is the aliasing of the high-frequency components of the pseudopotential that cannot be represented in the grid. When the pseudopotential is applied in Fourier space, these components are naturally filtered out.

To control the egg-box effect, most real-space codes use some sort of filtering that removes those high-frequency components.[11] A hard cutoff in Fourier space would not work, as it introduces ripples in real space that destroy the localization of the projectors. So, a more sophisticated approach is needed that yields a localized and soft pseudopotential in real space. In INQ, we use the approach by Tafipolsky and Schmid,[80] which in our experience with OCTOPUS has produced good results.[60,130] The filtering process is done once per run and calculated directly in the radial representation of the pseudopotential by the PSEUDOPOD library, so it does not add any additional computational cost.

VIII.VII. Poisson Solver. The Poisson solver is needed in INQ to obtain the long-range ionic potential and the Hartree potential (produced by the electron density itself). We solve the equation in FS, where the Poisson equation becomes a simple algebraic equation. To solve the equation in RS, additionally two FFTs are needed to go to FS and back.

In principle, the solution obtained in Fourier space has natural periodic boundary conditions. For finite systems, where neither the density nor the potential is periodic, this method introduces spurious interactions between cells, slowing the convergence with the size of the supercell. In this case, we use a modified kernel that exactly reproduces the free boundary conditions.[131] Unfortunately, this approach needs the simulation cell to be duplicated in each direction, making the cost of the Poisson solution 8 times more expensive for finite systems. Even with this factor of 8, this method is quite competitive in comparison with other approaches for free boundary conditions.[132] Both kinds of boundary conditions are implemented in INQ and can be chosen by the user depending on the physical system.

VIII.VIII. Forces. An accurate and efficient calculation of the forces is essential for adiabatic and nonadiabatic molecular dynamics. In the density functional framework, the forces are given by the formula

$$\mathbf{F}_I = -\sum_k \left\langle \varphi_k \left| \frac{\partial \hat{v}_I(\mathbf{r} - \mathbf{R}_I)}{\partial \mathbf{r}} \right| \varphi_k \right\rangle \tag{10}$$

Note that for the adiabatic forces in ground-state DFT, this expression comes from applying the Hellmann–Feynman theorem. While in the nonadiabatic case, this is directly the expression for the forces since φ_k and R_I are independent variables.[103]

The problem of eq 10 is that the gradient of the ionic potential appears. This gradient has higher-energy components than the potential itself. This means that the force calculated using eq 10 would be the bottleneck in the convergence with the basis. The gradient can also be complicated to calculate since it can have many terms, in particular, in the case of spin−orbit coupling. We can avoid this problem by transforming eq 10 into an expression that contains the gradients of the orbitals instead[133]

$$\mathbf{F}_I = \sum_k \langle \nabla \varphi_k | \hat{v}_I(\mathbf{r} - \mathbf{R}_I) | \varphi_k \rangle + c.c \tag{11}$$

Since the orbitals are smoother than the ionic potential, this last expression has better numerical properties as it converges much faster with the basis set. It is also much simpler to implement since the derivatives of the ionic potential are not needed. It only needs the calculation of the gradient of a field that, just like the Laplacian, is a local operation in Fourier space. The approach has also been extended for second-order derivatives,[62] which will be needed in the calculation of phonon frequencies.[134]

IX. RESULTS

In this section, we show some of the results obtained with the INQ code. The main objective is to validate the results and show that INQ can be reliably used for production runs. For these reasons, all of the calculations shown were done on GPUs; still, the CPU version of the code gives the same results to a high precision. In future work, we will show in detail numerical performance benchmarks; our focus here is on the physical predictions. We start by comparing the results of INQ with other codes using the same simulation parameters to show that the results are practically the same. Additionally, RT-TDDFT-based electronic stopping power, which investigates the system response to time-dependent spatially localized perturbations, is simulated and compared to previously published results.

IX.I. Validation with Other Codes. For a scientific code, it is fundamental to produce results that are consistent with other approaches and codes. For that, in this section, we compare the results of INQ with other established codes for a few cases. They illustrate a wider validation work that has been consistently done since we started the development on INQ. Our ultimate goal is to use the Delta factor[129] to assess the reliability of the codes for a large number of systems. However, some missing functionality like symmetry reduction of the Brillouin zone and non-orthogonal cells prevents us from performing that at the moment. (These features are not a priority for large-scale real-time simulations.)

INQ currently implements DFT total energies, forces, and real-time propagation of the time-dependent KS equations for both molecular and solid-state systems within a periodic supercell framework. In the following, results from INQ are compared with corresponding quantities obtained from established plane-wave and real-space grid codes: QUANTUM ESPRESSO (QE)[21] and OCTOPUS.[14] The quantities we compare are molecular bond lengths and crystal lattice parameters based on total energy
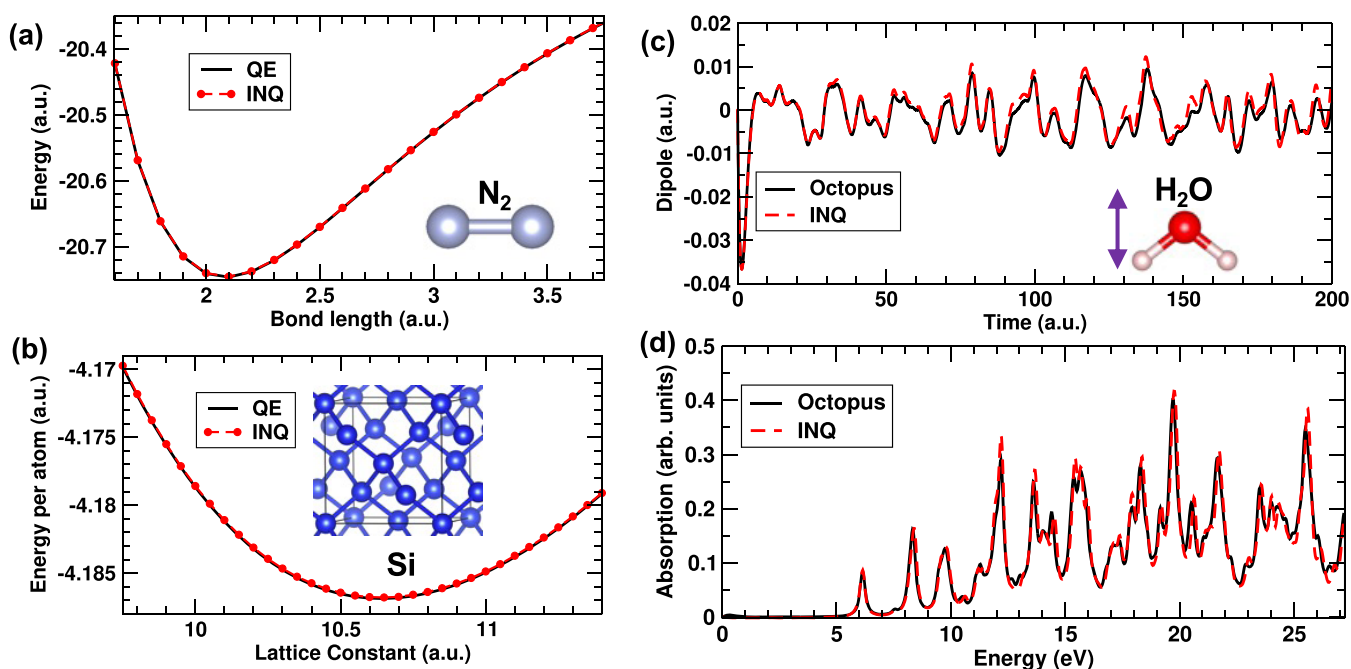
Figure 8. Comparison between INQ and the established electronic structure codes QUANTUM ESPRESSO (QE) and OCTOPUS. Results for ground-state total energies and real-time TDDFT optical response are shown. (a) Total energy vs bond length in N_2. (b) Total energy vs lattice constant in bulk silicon. (c) Time evolution of the dipole moment in gas-phase H_2O following a "kick" perturbation. (d) Linear optical absorption of molecular H_2O. In all cases, there is a high level of agreement between the codes.

minimization, as well as the linear optical response from real-time propagation. The total energy and optical response simulations employed standard norm-conserving scalar relativistic PBE pseudopotentials from Pseudodojo,[88] which are compatible with all of the codes used in the validation.

Figure 8a shows the total energy of the N_2 molecule as a function of the bond length as obtained from INQ and QE at a plane-wave cutoff of 80 au. Figure 8b shows the total energy dependence on the lattice constant for an eight-atom conventional unit cell of bulk Si. A plane-wave cutoff of 40 au was used in this instance, and the Brillouin zone was sampled only at the zone center. In each case, we find that the absolute total energies from INQ are within 0.3 mhartree/atom of the QE reference. The residual difference in total energies is attributed to the real-space treatment of the nonlocal pseudopotential term within INQ in contrast to the traditional reciprocal space treatment in plane-wave codes such as QE. As mentioned in Section VIII.VI, the real-space approach is expected to scale more favorably for larger system sizes, which INQ aims to target for applications.

In Figure 8c,d, the electronic response of a gas-phase water molecule to a time-localized "kick" electric field perturbation is plotted, as obtained from INQ and OCTOPUS. The simulation cell consists of a water molecule enclosed in a 10 Å × 10 Å × 10 Å box at a plane-wave cutoff of 40 au. Within INQ following the perturbation applied at the initial time $t = 0$, the time-dependent dipole moment of the system is recorded as given by the first moment of the time-evolving charge density distribution. In OCTOPUS, the time-dependent current is first calculated within the velocity gauge RT-TDDFT approach[135] and the dipole is then estimated as the time integral of the current. It is clear from Figure 8c, which shows the real-time evolution of the dipole moment for a perturbation along the C_2 symmetry axis of H_2O, that the INQ and OCTOPUS results are consistent as expected for a finite system. The above procedure is repeated for perturbations applied along three orthogonal coordinate directions, and

subsequently from the Fourier transform of the time-dependent dipole moments, the optical absorption spectrum is calculated and plotted in Figure 8d. It is apparent that excitation frequencies predicted by INQ and OCTOPUS are in good agreement over a wide frequency range, validating the real-time evolution algorithm implemented in INQ.

IX.II. Electronic Stopping. Stopping power in materials is a fundamental physical process by which energetic particles penetrating matter are slowed down by generating excitations of the material media, such as atomic displacements, phonons, electron–holes, secondary electrons, or plasmons.[136] If these fast particles move at velocities that are at least a fraction of that of the electrons in the materials (e.g., the Fermi velocity), the dissipative process is dominated by a continuum of electronic excitations. For example, this can happen as a result of an energetic nuclear event or by artificial ion accelerators, Electronic stopping power results are typically condensed into curves that relate dissipation rate, energy loss per unit distance, versus projectile velocity. Extensive databases of experimental curves exist in the literature.[137] Besides being a quantity of great importance in nuclear technology,[138] ion implantation,[139] and medicine,[140] the modeling of electronic stopping power is deeply intertwined with the development of the atomistic theory of matter[141−143] and the electron gas.[144−146]

Electronic stopping power is, from the point of view of the ion dynamics, a fundamentally nonadiabatic and dissipative process. As such, the process can, in principle, be tackled by direct simulations based on real-time TDDFT. If we imagine the process as a specific realization of a particle (projectile) traversing tens or hundreds of lattice parameters in a well-defined trajectory, we realize that the simulation naturally calls for a large supercell. The interaction between the projectile and the electron gas is simultaneously an aperiodic localized perturbation in space and also it is extended spatially as the simulation progresses. Therefore, simulations of stopping power

had become one of the most important applications of large-scale TDDFT simulations, both for their predictive power and as a benchmark of different approximations of TDDFT. By varying a single parameter (projectile velocity), different technical limitations of the methods can be reached; for example, (i) the position of the maximum of the stopping curve is sensitive to the completeness of the basis set,[147] (ii) the asymptotic behavior is affected by supercell size effects,[65] (iii) accuracy of the low-velocity limit is affected by the ability of simulating long times,[148] and (iv) highly charged projectiles (high Z) probe deep core electrons and the pseudopotential approximation.[149] For a full review and details on this type of calculation, see ref 150.

In this section, we present a benchmark calculation of stopping power in the prototypical case of a proton projectile in face-centered cubic (fcc) aluminum. Briefly, stopping power is extracted from simulation by forcing the projectile ion to move across a well-defined direction in the supercell and monitoring the additional energy gained by the system in the process. The average rate of increase in the energy corresponds to the kinetic energy loss by the projectile to the electronic system. The rate is generally obtained by linear fitting an energy−distance curve resulting from the real-time simulation.[151]

We use a plane-wave cutoff of 25 au to simulate three valence electrons per aluminum ion. Figure 9 shows how the energy
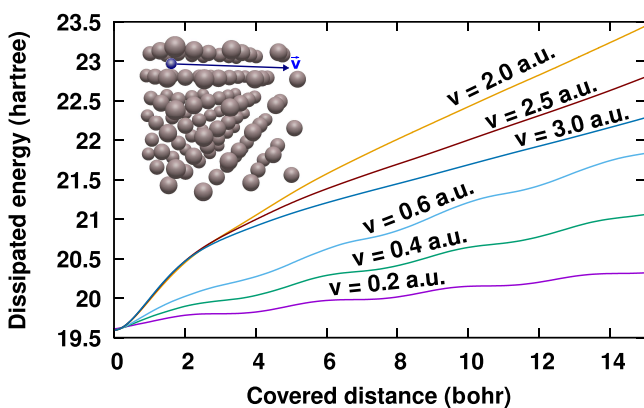

Figure 9. Energy dissipation as a function of distance covered by a proton in a rectilinear trajectory in the ⟨100⟩ channel for different velocities in an aluminum fcc supercell as calculated by a real-time calculation with the INQ code. Note that at low velocity the dissipation rate is proportional to the velocity; instead, above a certain velocity, the dissipation rate decays. Since the ions in the host material are fixed, the dissipation rate in the steady state is defined as the electronic stopping power for proton in aluminum shown in Figure 10.

transfer achieves a steady state whose slope corresponds to the stopping power at that velocity. This information is collected as an electronic stopping curve (Figure 10) of energy dissipation per unit length as a function of velocity.

These electronic stopping results illustrate that INQ is ready for state-of-the-art large-scale time-dependent computations, retaining accuracy for high-energy excitation processes such as in stopping power and other particle radiation-related applications thanks to a robust plane-wave basis.

X. FUTURE OF INQ

INQ is designed to be extensible. Not by any means a finished code, INQ allows new features to be added easily so as to tackle scientifically challenging problems that cannot be addressed
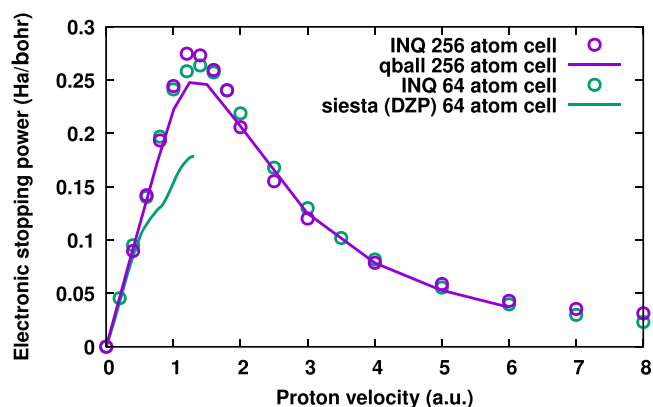

Figure 10. Electronic stopping power obtained by direct simulation of a proton projectile moving in aluminum atom fcc supercell with different codes: INQ, QBALL (from ref 65), and SIESTA with atom-centered double-ζ plus polarization basis (DZP) (from ref 147). Each point is obtained by evaluating the steady energy rate dissipation, e.g. from Figure 9. All simulations are with three valences (explicitly simulated) electrons per aluminum, which is enough for channeling stopping power.

with current codes. These problems might require a unique combination of functionalities not found in a single existing code base or might require modification of current algorithms or implementation of new ones.

The support of exact exchange[152−154] is planned for a future release of INQ. Exact exchange is needed for the accurate description of many transition-metal oxides. Manganite perovskites[155−157] and vanadium oxides VO_2[158,159] undergo metal−insulator transitions following optical excitation, likely associated with the emergence of ferromagnetic correlations on sub-picosecond time scales. Similarly, the melting of antiferromagnetic order in photoexcited nickelates within a few pico-seconds[160] is suggested to be directly linked to the melting of charge order. The combination of exact exchange and GPU scalability is needed for real-time TDDFT simulations at sizes and time scales required to investigate coherent dynamics in these systems. These simulations will clarify the roles of defects, dynamic disorder, and phonons beyond the accuracy of simplified models.[161−165] This will enable comparisons with a broad range of experimental techniques that now exist for probing these coupled electronic and structural dynamics including time-resolved optical and terahertz spectroscopy,[166,167] as well as ultrafast X-ray and electron scattering approaches.[168,169]

Support of relativistic effects will also be included in a future release of INQ. The spin−orbit interaction is important in topological materials and in systems with heavy atoms with effects such as Rashba/Dresselhaus splittings.[170,171] Other relativistic effects, such as Fermi contact and spin−dipole interactions,[172,173] are often neglected in solid-state DFT simulations but can be of importance in quantum information science applications.[174] INQ provides a framework where these effects can be implemented in an incremental and unintrusive way. As an example application, two-dimensional layered van der Waals systems with 5d heavy transition metals are a materials class where electron−electron interactions and the spin−orbit coupling are important energy scales. These materials are known to exhibit newly broken symmetry or topological phases, including charge density waves and metastable metallic states.[175−179] Real-time TDDFT will clarify details of the mechanisms behind the evolution of charge instabilities,

periodic lattice distortions, and coherent phonon generation in these 2D-layered systems. In systems with antiferromagnetic order, magnon modes may emerge as collective excitations. Twisted boundary conditions are often used to simulate such systems and those with incommensurate spiral physics in general.[180] These capabilities can be included in INQ by modifying the currently available boundary conditions (Section VIII.VII).

INQ supports Ehrenfest dynamics and can be extended to other forms of nonadiabatic molecular dynamics.[181,182] A variety of proposed dynamics departing from Born−Oppenheimer (BO) and beyond Ehrenfest would be faster to experiment with. For example, it has been noted recently that friction terms can be added ab initio to model electron−phonon coupling with classical ions. These dissipative terms can be calculated on-the-fly by launching tangential short TDDFT simulations (in parallel to the main trajectory evolution). The dissipative contributions can then be added to Newton's equation of motion by terms in the form of $M\ddot{R} = F_{BO} - \beta\dot{R} + \xi$, where β and ξ are a friction tensor and correlated noise, respectively.[183]

These functionalities are required for applications that involve any form of coupled electron−ion dynamics and become especially useful in challenging materials systems that also require advanced electronic structure methods such as those above. For instance, the bulk photovoltaic effect involves a number of nonlinear photocurrent generation effects driven by different mechanisms.[184−189] Recent tight binding model simulations have predicted an enhanced bulk photovoltaic effect due to coupled spin and phonon dynamics in a strongly correlated manganite perovskite.[190] First-principles exploration and comparison of this enhancement effect across a variety of materials systems will result in materials design insights for maximizing its magnitude.

Another interest area of application for INQ is high-throughput screening and machine learning methods combined with DFT.[191] Since INQ is a library, it allows for a horizontal integration with other libraries and software packages (for example, materials informatics, machine learning, or data analysis tools) rather than having one software depending on the other. Programs that need to collect data for large numbers of DFT calculations can directly run INQ through function calls. This way, they can directly control the execution of the simulation and naturally pass the input and receive the output through data structures in memory, instead of files that need to be parsed.

Thanks to its flexibility, INQ could be used as well for more advanced integration with machine learning. One of these applications is fully differentiable DFT that can be used to improve exchange−correlation approximations.[35,192] In C++, this differentiation can be done automatically, with minimal modifications of the code, through templates and operator overloading.[193,194]

INQ is written with the intention that the user can take an active role in developing electronic structure calculations according to their needs. While the above examples provide some possible future directions, the user is in fact limited only by their imagination when it comes to new feature developments.

XI. CONCLUSIONS

We have presented a new framework for the computational simulation of electronic systems. It has several new design characteristics that offer unique advantages over existing legacy DFT codes. Using a modern approach to coding, based on the C

++ language, has allowed us to write a very compact code that directly expresses the formulation of the problem and not the implementation details. When combined with an extensive use of testing of the code, we can develop and implement new features very fast. The code was designed from scratch to work with GPUs and MPI parallelization, which means it can make use of modern supercomputing platforms and be quickly adapted for future ones. Within the scope of basic DFT, DFT-MD, and TDDFT, the code is production ready.

This collection of features makes INQ an ideal platform to apply TDDFT to a range of physical problems that are hard to approach with current software. Of course, this will require further development of the code and theoretical tools by INQ developers and the broader electronic structure community. In this spirit, we look to make INQ an open platform that other researchers can use and adapt for their research and that can interact with other software components.

■ AUTHOR INFORMATION

Corresponding Author

Xavier Andrade − *Quantum Simulations Group, Lawrence Livermore National Laboratory, Livermore, California 94551, United States;* ⓞ orcid.org/0000-0002-0416-637X; Email: xavier@llnl.gov

Authors

Chaitanya Das Pemmaraju − *Stanford Institute for Materials and Energy Sciences, SLAC National Accelerator Laboratory, Menlo Park, California 94025, United States;* ⓞ orcid.org/0000-0002-9016-7044

Alexey Kartsev − *Stanford Institute for Materials and Energy Sciences, SLAC National Accelerator Laboratory, Menlo Park, California 94025, United States;* ⓞ orcid.org/0000-0002-3916-3443

Jun Xiao − *Stanford Institute for Materials and Energy Sciences, SLAC National Accelerator Laboratory, Menlo Park, California 94025, United States;* ⓞ orcid.org/0000-0003-4248-8190

Aaron Lindenberg − *Stanford Institute for Materials and Energy Sciences, SLAC National Accelerator Laboratory, Menlo Park, California 94025, United States;* ⓞ orcid.org/0000-0003-3233-7161

Sangeeta Rajpurohit − *The Molecular Foundry, Lawrence Berkeley National Laboratory, Berkeley, California 94720, United States*

Liang Z. Tan − *The Molecular Foundry, Lawrence Berkeley National Laboratory, Berkeley, California 94720, United States;* ⓞ orcid.org/0000-0003-4724-6369

Tadashi Ogitsu − *Quantum Simulations Group, Lawrence Livermore National Laboratory, Livermore, California 94551, United States*

Alfredo A. Correa − *Quantum Simulations Group, Lawrence Livermore National Laboratory, Livermore, California 94551, United States*

Complete contact information is available at:
https://pubs.acs.org/10.1021/acs.jctc.1c00562

Notes

The authors declare no competing financial interest.

■ REFERENCES

(1) Hohenberg, P.; Kohn, W. Inhomogeneous Electron Gas. *Phys. Rev.* **1964**, *136*, B864−B871.

(2) Kohn, W.; Sham, L. J. Self-Consistent Equations Including Exchange and Correlation Effects. *Phys. Rev.* **1965**, *140*, A1133−A1138.

(3) Kohn, W. Nobel Lecture: Electronic structure of matter—wave functions and density functionals. *Rev. Mod. Phys.* **1999**, *71*, 1253−1266.

(4) Runge, E.; Gross, E. K. U. Density-Functional Theory for Time-Dependent Systems. *Phys. Rev. Lett.* **1984**, *52*, 997−1000.

(5) Hafner, J.; Wolverton, C.; Ceder, G. Toward computational materials design: the impact of density functional theory on materials research. *MRS Bull.* **2006**, *31*, 659−668.

(6) Burke, K. Perspective on density functional theory. *J. Chem. Phys.* **2012**, *136*, No. 150901.

(7) Mardirossian, N.; Head-Gordon, M. Thirty years of density functional theory in computational chemistry: an overview and extensive assessment of 200 density functionals. *Mol. Phys.* **2017**, *115*, 2315−2372.

(8) Hasnip, P. J.; Refson, K.; Probert, M. I.; Yates, J. R.; Clark, S. J.; Pickard, C. J. Density functional theory in the solid state. *Philos. Trans. R. Soc., A* **2014**, *372*, No. 20130270.

(9) Hehre, W. J.; Stewart, R. F.; Pople, J. A. Self-consistent molecular-orbital methods. I. Use of Gaussian expansions of Slater-type atomic orbitals. *J. Chem. Phys.* **1969**, *51*, 2657−2664.

(10) Chelikowsky, J. R.; Troullier, N.; Saad, Y. Finite-difference-pseudopotential method: Electronic structure calculations without a basis. *Phys. Rev. Lett.* **1994**, *72*, 1240.

(11) Briggs, E.; Sullivan, D.; Bernholc, J. Real-space multigrid-based approach to large-scale electronic structure calculations. *Phys. Rev. B* **1996**, *54*, 14362.

(12) Kresse, G.; Furthmüller, J. Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set. *Phys. Rev. B* **1996**, *54*, 11169.

(13) Soler, J. M.; Artacho, E.; Gale, J. D.; García, A.; Junquera, J.; Ordejón, P.; Sánchez-Portal, D. The SIESTA method for ab initio order-N materials simulation. *J. Phys.: Condens. Matter* **2002**, *14*, 2745.

(14) Castro, A.; Appel, H.; Oliveira, M.; Rozzi, C. A.; Andrade, X.; Lorenzen, F.; Marques, M. A.; Gross, E.; Rubio, A. Octopus: a tool for the application of time-dependent density functional theory. *Phys. Status Solidi B* **2006**, *243*, 2465−2488.

(15) Gygi, F. Architecture of Qbox: A scalable first-principles molecular dynamics code. *IBM J. Res. Dev.* **2008**, *52*, 137−144.

(16) Blum, V.; Gehrke, R.; Hanke, F.; Havu, P.; Havu, V.; Ren, X.; Reuter, K.; Scheffler, M. Ab initio molecular simulations with numeric atom-centered orbitals. *Comput. Phys. Commun.* **2009**, *180*, 2175−2196.

(17) Enkovaara, J.; Rostgaard, C.; Mortensen, J. J.; Chen, J.; Du lak, M.; Ferrighi, L.; Gavnholt, J.; Glinsvad, C.; Haikola, V.; Hansen, H.; et al. Electronic structure calculations with GPAW: a real-space implementation of the projector augmented-wave method. *J. Phys.: Condens. Matter* **2010**, *22*, No. 253202.

(18) Shao, Y.; Gan, Z.; Epifanovsky, E.; Gilbert, A. T.; Wormit, M.; Kussmann, J.; Lange, A. W.; Behn, A.; Deng, J.; Feng, X.; et al. Advances in molecular quantum chemistry contained in the Q-Chem 4 program package. *Mol. Phys.* **2015**, *113*, 184−215.

(19) Genova, A.; Ceresoli, D.; Krishtal, A.; Andreussi, O.; DiStasio, R. A., Jr.; Pavanello, M. eQE: An open-source density functional embedding theory code for the condensed phase. *Int. J. Quantum Chem.* **2017**, *117*, No. e25401.

(20) Draeger, E. W.; Andrade, X.; Gunnels, J. A.; Bhatele, A.; Schleife, A.; Correa, A. A. Massively parallel first-principles simulation of electron dynamics in materials. *J. Parallel Distrib. Comput.* **2017**, *106*, 205−214.

(21) Giannozzi, P.; et al. Advanced capabilities for materials modelling with Quantum ESPRESSO. *J. Phys.: Condens. Matter* **2017**, *29*, No. 465901.

(22) Noda, M.; Sato, S. A.; Hirokawa, Y.; Uemoto, M.; Takeuchi, T.; Yamada, S.; Yamada, A.; Shinohara, Y.; Yamaguchi, M.; Iida, K.; et al. Salmon: Scalable ab-initio light−matter simulator for optics and nanoscience. *Comput. Phys. Commun.* **2019**, *235*, 356−365.

(23) Aprá, E.; et al. NWChem: Past, present, and future. *J. Chem. Phys.* **2020**, *152*, No. 184102.

(24) Kühne, T. D.; Iannuzzi, M.; Del Ben, M.; Rybkin, V. V.; Seewald, P.; Stein, F.; Laino, T.; Khaliullin, R. Z.; Schütt, O.; Schiffmann, F.; et al. CP2K: An electronic structure and molecular dynamics software package-Quickstep: Efficient and accurate electronic structure calculations. *J. Chem. Phys.* **2020**, *152*, No. 194103.

(25) Gonze, X.; et al. The Abinit project: Impact, environment and recent developments. *Comput. Phys. Commun.* **2020**, *248*, No. 107042.

(26) Seritan, S.; Bannwarth, C.; Fales, B. S.; Hohenstein, E. G.; Isborn, C. M.; Kokkila-Schumacher, S. I.; Li, X.; Liu, F.; Luehr, N.; Snyder, J. W., Jr; et al. TeraChem: A graphical processing unit-accelerated electronic structure package for large-scale ab initio molecular dynamics. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **2021**, *11*, No. e1494.

(27) Verma, P.; Truhlar, D. G. Status and challenges of density functional theory. *Trends Chem.* **2020**, *2*, 302−318.

(28) Curtarolo, S.; Hart, G. L.; Nardelli, M. B.; Mingo, N.; Sanvito, S.; Levy, O. The high-throughput highway to computational materials design. *Nat. Mater.* **2013**, *12*, 191−201.

(29) Oba, F.; Kumagai, Y. Design and exploration of semiconductors from first principles: A review of recent advances. *Appl. Phys. Express* **2018**, *11*, No. 060101.

(30) Marzari, N.; Ferretti, A.; Wolverton, C. Electronic-structure methods for materials design. *Nat. Mater.* **2021**, *20*, 736−749.

(31) Oganov, A. R.; Glass, C. W. Crystal structure prediction usingab initioevolutionary techniques: Principles and applications. *J. Chem. Phys.* **2006**, *124*, No. 244704.

(32) Lemonick, S. Is machine learning overhyped? *Chem. Eng. News* **2018**, *96*, 16−20.

(33) Schmidt, J.; Marques, M. R.; Botti, S.; Marques, M. A. Recent advances and applications of machine learning in solid-state materials science. *npj Comput. Mater.* **2019**, *5*, No. 83.

(34) Tkatchenko, A. Machine learning for chemical discovery. *Nat. Commun.* **2020**, *11*, No. 4125.

(35) Li, L.; Hoyer, S.; Pederson, R.; Sun, R.; Cubuk, E. D.; Riley, P.; Burke, K.; et al. Kohn-Sham equations as regularizer: Building prior knowledge into machine-learned physics. *Phys. Rev. Lett.* **2021**, *126*, No. 036401.

(36) Shaidu, Y.; Küçükbenli, E.; Lot, R.; Pellegrini, F.; Kaxiras, E.; de Gironcoli, S. A systematic approach to generating accurate neural network potentials: the case of carbon. *npj Comput. Mater.* **2021**, *7*, No. 52.

(37) Hachmann, J.; Olivares-Amaya, R.; Jinich, A.; Appleton, A. L.; Blood-Forsythe, M. A.; Seress, L. R.; Román-Salgado, C.; Trepte, K.; Atahan-Evrenk, S.; Er, S.; et al. Lead candidates for high-performance organic photovoltaics from high-throughput quantum chemistry—the Harvard Clean Energy Project. *Energy Environ. Sci.* **2014**, *7*, 698−704.

(38) Cohen, A. J.; Mori-Sánchez, P.; Yang, W. Insights into current limitations of density functional theory. *Science* **2008**, *321*, 792−794.

(39) Gori-Giorgi, P.; Seidl, M. Density functional theory for strongly-interacting electrons: perspectives for physics and chemistry. *Phys. Chem. Chem. Phys.* **2010**, *12*, 14405−14419.

(40) Jain, A.; Shin, Y.; Persson, K. A. Computational predictions of energy materials using density functional theory. *Nat. Rev. Mater.* **2016**, *1*, No. 15004.

(41) Ullrich, C. A.; Yang, Z.-h. A brief compendium of time-dependent density functional theory. *Braz. J. Phys.* **2014**, *44*, 154−188.

(42) Onida, G.; Reining, L.; Rubio, A. Electronic excitations: density-functional versus many-body Green's-function approaches. *Rev. Mod. Phys.* **2002**, *74*, 601.

(43) Castro, A.; Marques, M. A.; Varsano, D.; Sottile, F.; Rubio, A. The challenge of predicting optical properties of biomolecules: what can we learn from time-dependent density-functional theory? *C. R. Phys.* **2009**, *10*, 469−490.

(44) Elliott, P.; Goldson, S.; Canahui, C.; Maitra, N. T. Perspectives on double-excitations in TDDFT. *Chem. Phys.* **2011**, *391*, 110−119.

(45) Lian, C.; Guan, M.; Hu, S.; Zhang, J.; Meng, S. Photoexcitation in Solids: First-Principles Quantum Simulations by Real-Time TDDFT. *Adv. Theory Simul.* **2018**, *1*, No. 1800055.

(46) Izmaylov, A. F.; Scuseria, G. E. Why are time-dependent density functional theory excitations in solids equal to band structure energy gaps for semilocal functionals, and how does nonlocal Hartree−Fock-type exchange introduce excitonic effects? *J. Chem. Phys.* **2008**, *129*, No. 034101.

(47) Ullrich, C. A.; Yang, Z.-h. In *Density-Functional Methods for Excited States*; Ferré, N., Filatov, M., Huix-Rotllant, M., Eds.; Springer International Publishing: Cham, 2016; pp 185−217.

(48) Refaely-Abramson, S.; Jain, M.; Sharifzadeh, S.; Neaton, J. B.; Kronik, L. Solid-state optical absorption from optimally tuned time-dependent range-separated hybrid density functional theory. *Phys. Rev. B* **2015**, *92*, No. 081204.

(49) Kümmel, S. Charge-Transfer Excitations: a challenge for time-dependent density functional theory that has been met. *Adv. Energy Mater.* **2017**, *7*, No. 1700440.

(50) Pemmaraju, C. D. Valence and core excitons in solids from velocity-gauge real-time TDDFT with range-separated hybrid functionals: An LCAO approach. *Comput. Condes. Matter.* **2019**, *18*, No. e00348.

(51) Sun, J.; Lee, C.-W.; Kononov, A.; Schleife, A.; Ullrich, C. A. Real-Time Exciton Dynamics with Time-Dependent Density-Functional Theory, 2021, arXiv:2102.01796. arXiv.org e-Print archive. https://arxiv.org/abs/2102.01796.

(52) Kothe, D.; Lee, S.; Qualters, I. Exascale Computing in the United States. *Comput. Sci. Eng.* **2019**, *21*, 17−29.

(53) Stone, J. E.; Phillips, J. C.; Freddolino, P. L.; Hardy, D. J.; Trabuco, L. G.; Schulten, K. Accelerating molecular modeling applications with graphics processors. *J. Comput. Chem.* **2007**, *28*, 2618−2640.

(54) Biagini, T.; Petrizzelli, F.; Truglio, M.; Cespa, R.; Barbieri, A.; Capocefalo, D.; Castellana, S.; Tevy, M. F.; Carella, M.; Mazza, T. Are Gaming-Enabled Graphic Processing Unit Cards Convenient for Molecular Dynamics Simulation? *Evol. Bioinf.* **2019**, *15*, No. 1176934319850144.

(55) Genovese, L.; Ospici, M.; Deutsch, T.; Méhaut, J.-F.; Neelov, A.; Goedecker, S. Density functional theory calculation on many-cores hybrid central processing unit-graphic processing unit architectures. *J. Chem. Phys.* **2009**, *131*, No. 034103.

(56) Andrade, X.; Aspuru-Guzik, A. Real-space density functional theory on graphical processing units: computational approach and comparison to gaussian basis set methods. *J. Chem. Theory Comput.* **2013**, *9*, 4360−4373.

(57) Hacene, M.; Anciaux-Sedrakian, A.; Rozanska, X.; Klahr, D.; Guignon, T.; Fleurat-Lessard, P. Accelerating VASP electronic structure calculations using graphic processing units. *J. Comput. Chem.* **2012**, *33*, 2581−2589.

(58) Romero, J.; Phillips, E.; Ruetsch, G.; Fatica, M.; Spiga, F.; Giannozzi, P. In *A Performance Study of Quantum ESPRESSO's PWscf Code on Multi-Core and GPU Systems*, International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, 2017; pp 67−87.

(59) Huhn, W. P.; Lange, B.; Yu, V. W.-z.; Yoon, M.; Blum, V. GPU acceleration of all-electron electronic structure theory using localized numeric atom-centered basis functions. *Comput. Phys. Commun.* **2020**, *254*, No. 107314.

(60) Andrade, X. Linear and Non-Linear Response Phenomena of Molecular Systems within Time-Dependent Density Functional Theory. University of the Basque Country, 2010.

(61) Andrade, X.; Alberdi-Rodriguez, J.; Strubbe, D. A.; Oliveira, M. J.; Nogueira, F.; Castro, A.; Muguerza, J.; Arruabarrena, A.; Louie, S. G.; Aspuru-Guzik, A.; et al. Time-dependent density-functional theory in massively parallel computer architectures: the octopus project. *J. Phys.: Condens. Matter* **2012**, *24*, No. 233202.

(62) Andrade, X.; Strubbe, D.; De Giovannini, U.; Larsen, A. H.; Oliveira, M. J.; Alberdi-Rodriguez, J.; Varas, A.; Theophilou, I.; Helbig, N.; Verstraete, M. J.; et al. Real-space grids and the Octopus code as tools for the development of new simulation approaches for electronic systems. *Phys. Chem. Chem. Phys.* **2015**, *17*, 31371−31396.

(63) Tancogne-Dejean, N.; Oliveira, M. J.; Andrade, X.; Appel, H.; Borca, C. H.; Le Breton, G.; Buchholz, F.; Castro, A.; Corni, S.; Correa, A. A.; et al. Octopus, a computational framework for exploring light-driven phenomena and quantum dynamics in extended and finite systems. *J. Chem. Phys.* **2020**, *152*, No. 124119.

(64) Schleife, A.; Draeger, E. W.; Kanai, Y.; Correa, A. A. Plane-wave pseudopotential implementation of explicit integrators for time-dependent Kohn-Sham equations in large-scale simulations. *J. Chem. Phys.* **2012**, *137*, No. 22A546.

(65) Schleife, A.; Kanai, Y.; Correa, A. A. Accurate atomistic first-principles calculations of electronic stopping. *Phys. Rev. B* **2015**, *91*, No. 014306.

(66) Gygi, F.; Yates, R. K.; Lorenz, J.; Draeger, E. W.; Franchetti, F.; Ueberhuber, C. W.; de Supinski, B. R.; Kral, S.; Gunnels, J. A.; Sexton, J. C. In *Large-Scale First-Principles Molecular Dynamics Simulations on the Bluegene/l Platform Using the qbox Code*, SC'05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing, 2005; pp 24.

(67) Martin, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*; Pearson Education, 2009.

(68) Loeliger, J.; McCullough, M. *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*; O'Reilly Media, 2012.

(69) Larsen, A. H.; et al. The atomic simulation environment—a Python library for working with atoms. *J. Phys.: Condens. Matter* **2017**, *29*, No. 273002.

(70) Jain, A.; Ong, S. P.; Hautier, G.; Chen, W.; Richards, W. D.; Dacek, S.; Cholia, S.; Gunter, D.; Skinner, D.; Ceder, G.; et al. Commentary: The Materials Project: A materials genome approach to accelerating materials innovation. *APL Mater.* **2013**, *1*, No. 011002.

(71) Ince, D. C.; Hatton, L.; Graham-Cumming, J. The case for open computer programs. *Nature* **2012**, *482*, 485−488.

(72) Smart, A. G. The war over supercooled water. *Phys. Today* **2018**, DOI: 10.1063/PT.6.1.20180822a.

(73) Stepanov, A.; McJones, P. *Elements of Programming*; Addison-Wesley, 2009.

(74) Oliveira, M. J. T.; et al. The CECAM electronic structure library and the modular software development paradigm. *J. Chem. Phys.* **2020**, *153*, No. 024117.

(75) Togo, A.; Tanaka, I. Spglib: A Software Library for Crystal Symmetry Search. 2018, arXiv:1808.01590. arXiv.org e-Print archive. https://arxiv.org/abs/1808.01590.

(76) Cooley, J. W.; Tukey, J. W. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.* **1965**, *19*, 297−301.

(77) Kim, J.; et al. QMCPACK: an open source ab initio quantum Monte Carlo package for the electronic structure of atoms, molecules and solids. *J. Phys.: Condens. Matter* **2018**, *30*, No. 195901.

(78) García, A.; Verstraete, M. J.; Pouillon, Y.; Junquera, J. The psml format and library for norm-conserving pseudopotential data curation and interoperability. *Comput. Phys. Commun.* **2018**, *227*, 51−71.

(79) Hamann, D. Optimized norm-conserving Vanderbilt pseudopotentials. *Phys. Rev. B* **2013**, *88*, No. 085117.

(80) Tafipolsky, M.; Schmid, R. A general and efficient pseudopotential Fourier filtering scheme for real space methods using mask functions. *J. Chem. Phys.* **2006**, *124*, No. 174102.

(81) Willand, A.; Kvashnin, Y. O.; Genovese, L.; Vázquez-Mayagoitia, Á.; Deb, A. K.; Sadeghi, A.; Deutsch, T.; Goedecker, S. Norm-conserving pseudopotentials with chemical accuracy compared to all-electron calculations. *J. Chem. Phys.* **2013**, *138*, No. 104109.

(82) Dal Corso, A. Pseudopotentials periodic table: From H to Pu. *Comput. Mater. Sci.* **2014**, *95*, 337−350.

(83) Garrity, K. F.; Bennett, J. W.; Rabe, K. M.; Vanderbilt, D. Pseudopotentials for high-throughput DFT calculations. *Comput. Mater. Sci.* **2014**, *81*, 446−452.

(84) Kucukbenli, E.; Monni, M.; Adetunji, B.; Ge, X.; Adebayo, G.; Marzari, N.; De Gironcoli, S.; Corso, A. D. Projector Augmented-Wave and All-Electron Calculations across the Periodic Table: A Comparison of Structural and Energetic Properties. 2014, arXiv:1404.3015. arXiv.org e-Print archive. https://arxiv.org/abs/1404.3015.

(85) Topsakal, M.; Wentzcovitch, R. Accurate projected augmented wave (PAW) datasets for rare-earth elements (RE= La−Lu). *Comput. Mater. Sci.* **2014**, *95*, 263−270.

(86) Schlipf, M.; Gygi, F. Optimization algorithm for the generation of ONCV pseudopotentials. *Comput. Phys. Commun.* **2015**, *196*, 36−44.

(87) Prandini, G.; Marrazzo, A.; Castelli, I. E.; Mounet, N.; Marzari, N. Precision and efficiency in solid-state pseudopotential calculations. *npj Comput. Mater.* **2018**, *4*, No. 72.

(88) Van Setten, M.; Giantomassi, M.; Bousquet, E.; Verstraete, M. J.; Hamann, D. R.; 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.

(89) Marques, M. A.; Oliveira, M. J.; Burnus, T. Libxc: A library of exchange and correlation functionals for density functional theory. *Comput. Phys. Commun.* **2012**, *183*, 2272−2281.

(90) Lehtola, S.; Steigemann, C.; Oliveira, M. J.; Marques, M. A. Recent developments in libxc − A comprehensive library of functionals for density functional theory. *SoftwareX* **2018**, *7*, 1−5.

(91) Stepanov, A.; Rose, D. *From Mathematics to Generic Programming*; Addison-Wesley, 2014.

(92) Andrade, X.; Genovese, L. *Fundamentals of Time-Dependent Density Functional Theory*; Springer, 2012; pp 401−413.

(93) Beckingsale, D. A.; Burmark, J.; Hornung, R.; Jones, H.; Killian, W.; Kunen, A. J.; Pearce, O.; Robinson, P.; Ryujin, B. S.; Scogland, T. R. In *RAJA: Portable Performance for Large-Scale Scientific Applications*, 2019 IEEE ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC), 2019; pp 71−81.

(94) Edwards, H. C.; Trott, C. R.; Sunderland, D. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *J. Parallel Distrib. Comput.* **2014**, *74*, 3202−3216.

(95) Lee, S.; Eigenmann, R. In *OpenMPC: Extended OpenMP Programming and Tuning for GPUs*, SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, 2010; pp 1−11.

(96) Alpay, A.; Heuveline, V. In *SYCL beyond OpenCL: The Architecture, Current State and Future Direction of hipSYCL*, Proceedings of the International Workshop on OpenCL, 2020; pp 1.

(97) Chien, S.; Peng, I.; Markidis, S. In *Performance Evaluation of Advanced Features in CUDA Unified Memory*, 2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC), 2019; pp 50−57.

(98) Harris, M. et al. *Optimizing Parallel Reduction in CUDA*; NVIDIA Developer Technology, 2007; Vol. 2, pp 1−39.

(99) Pacheco, P. *Parallel Programming with MPI*; Elsevier Science, 1997.

(100) Jornet-Somoza, J.; Alberdi-Rodriguez, J.; Milne, B. F.; Andrade, X.; Marques, M. A.; Nogueira, F.; Oliveira, M. J.; Stewart, J. J.; Rubio, A. Insights into colour-tuning of chlorophyll optical response in green plants. *Phys. Chem. Chem. Phys.* **2015**, *17*, 26599−26606.

(101) Hasegawa, Y.; Iwata, J.-I.; Tsuji, M.; Takahashi, D.; Oshiyama, A.; Minami, K.; Boku, T.; Shoji, F.; Uno, A.; Kurokawa, M. et al. In *First-Principles Calculations of Electron States of a Silicon Nanowire with 100,000 Atoms on the K Computer*, Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, 2011; pp 1−11.

(102) Ehrenfest, P. Bemerkung über die angenäherte Gültigkeit der klassischen Mechanik innerhalb der Quantenmechanik. *Z. Phys.* **1927**, *45*, 455−457.

(103) Andrade, X.; Castro, A.; Zueco, D.; Alonso, J.; Echenique, P.; Falceto, F.; Rubio, A. Modified Ehrenfest formalism for efficient large-scale ab initio molecular dynamics. *J. Chem. Theory Comput.* **2009**, *5*, 728−742.

(104) Szabo, A.; Ostlund, N. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory (Dover Books on Chemistry)*; Dover Publications, 2012.

(105) Olsen, J. *Basis Sets in Computational Chemistry*; Springer International Publishing: Cham, 2021; pp 1−16.

(106) Broyden, C. G. A class of methods for solving nonlinear simultaneous equations. *Math. Comput.* **1965**, *19*, 577−593.

(107) Pulay, P. Convergence acceleration of iterative sequences. The case of SCF iteration. *Chem. Phys. Lett.* **1980**, *73*, 393−398.

(108) Saad, Y. *Numerical Methods for Large Eigenvalue Problems*; Society for Industrial and Applied Mathematics, 2011.

(109) Axler, S. *Linear Algebra Done Right (Undergraduate Texts in Mathematics)*; Springer International Publishing, 2014; p 151.

(110) Payne, M. C.; Teter, M. P.; Allan, D. C.; Arias, T. A.; Joannopoulos, J. D. Iterative minimization techniques for ab initio total-energy calculations: molecular dynamics and conjugate gradients. *Rev. Mod. Phys.* **1992**, *64*, 1045.

(111) Davidson, E. The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *J. Comput. Phys.* **1975**, *17*, 87−94.

(112) Teter, M. P.; Payne, M. C.; Allan, D. C. Solution of Schrödinger's equation for large systems. *Phys. Rev. B* **1989**, *40*, 12255.

(113) Blackford, L. S.; Petitet, A.; Pozo, R.; Remington, K.; Whaley, R. C.; Demmel, J.; Dongarra, J.; Duff, I.; Hammarling, S.; Henry, G.; et al. An updated set of basic linear algebra subprograms (BLAS). *ACM Trans. Math. Software* **2002**, *28*, 135−151.

(114) Anderson, E.; Bai, Z.; Bischof, C.; Blackford, S.; Demmel, J.; Dongarra, J.; Du Croz, J.; Greenbaum, A.; Hammarling, S.; McKenney, A.; Sorensen, D. *LAPACK Users' Guide*, 3rd ed.; Society for Industrial and Applied Mathematics: Philadelphia, PA, 1999.

(115) Blackford, L. S.; Choi, J.; Cleary, A.; D'Azevedo, E.; Demmel, J.; Dhillon, I.; Dongarra, J.; Hammarling, S.; Henry, G.; Petitet, A.; Stanley, K.; Walker, D.; Whaley, R. C. *ScaLAPACK Users' Guide*; Society for Industrial and Applied Mathematics: Philadelphia, PA, 1997.

(116) Gates, M.; Kurzak, J.; Charara, A.; YarKhan, A.; Dongarra, J. In *Slate: Design of a Modern Distributed and Accelerated Linear Algebra Library*, Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2019; pp 1−18.

(117) O'Neill, M. E. PCG: A family of simple fast space-efficient statistically good algorithms for random number generation. *ACM Trans. Math. Software* **2014**, 1–46.

(118) Castro, A.; Marques, M. A.; Rubio, A. Propagators for the time-dependent Kohn−Sham equations. *J. Chem. Phys.* **2004**, *121*, 3425–3433.

(119) Kidd, D.; Covington, C.; Varga, K. Exponential integrators in time-dependent density-functional calculations. *Phys. Rev. E* **2017**, *96*, No. 063307.

(120) Gómez Pueyo, A.; Marques, M. A.; Rubio, A.; Castro, A. Propagators for the time-dependent Kohn−Sham equations: Multistep, Runge−Kutta, exponential Runge−Kutta, and commutator free magnus methods. *J. Chem. Theory Comput.* **2018**, *14*, 3040–3052.

(121) Jia, W.; An, D.; Wang, L.-W.; Lin, L. Fast real-time time-dependent density functional theory calculations with the parallel transport gauge. *J. Chem. Theory Comput.* **2018**, *14*, 5645–5652.

(122) Bader, P.; Blanes, S.; Kopylov, N. Exponential propagators for the Schrödinger equation with a time-dependent potential. *J. Chem. Phys.* **2018**, *148*, No. 244109.

(123) Zhu, Y.; Herbert, J. M. Self-consistent predictor/corrector algorithms for stable and efficient integration of the time-dependent Kohn-Sham equation. *J. Chem. Phys.* **2018**, *148*, No. 044117.

(124) Verlet, L. Computer" experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Phys. Rev.* **1967**, *159*, 98.

(125) Ayala, A.; Tomov, S.; Haidar, A.; Dongarra, J. In *heffte: Highly Efficient fft for Exascale*, International Conference on Computational Science, 2020; pp 262–275.

(126) Wadleigh, K.; Crawford, I. *Software Optimization for High-Performance Computing (HP Professional)*; Prentice Hall PTR, 2000.

(127) Vanderbilt, D. Soft self-consistent pseudopotentials in a generalized eigenvalue formalism. *Phys. Rev. B* **1990**, *41*, 7892.

(128) Blöchl, P. E. Projector augmented-wave method. *Phys. Rev. B* **1994**, *50*, 17953.

(129) Lejaeghere, K.; Bihlmayer, G.; Björkman, T.; Blaha, P.; Blügel, S.; Blum, V.; Caliste, D.; Castelli, I. E.; Clark, S. J.; Dal Corso, A.; et al. Reproducibility in density functional theory calculations of solids. *Science* **2016**, *351*, No. aad3000.

(130) Varsano, D.; Espinosa-Leal, L. A.; Andrade, X.; Marques, M. A.; Di Felice, R.; Rubio, A. Towards a gauge invariant method for molecular chiroptical properties in TDDFT. *Phys. Chem. Chem. Phys.* **2009**, *11*, 4481–4489.

(131) Rozzi, C. A.; Varsano, D.; Marini, A.; Gross, E. K.; Rubio, A. Exact Coulomb cutoff technique for supercell calculations. *Phys. Rev. B* **2006**, *73*, No. 205119.

(132) García-Risueño, P.; Alberdi-Rodriguez, J.; Oliveira, M. J.; Andrade, X.; Pippig, M.; Muguerza, J.; Arruabarrena, A.; Rubio, A. A survey of the parallel performance and accuracy of Poisson solvers for electronic structure calculations. *J. Comput. Chem.* **2014**, *35*, 427–444.

(133) Hirose, K. *First-Principles Calculations in Real-Space Formalism: Electronic Configurations and Transport Properties of Nanostructures*; Imperial College Press, 2005; p 12.

(134) 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.

(135) Yabana, K.; Sugiyama, T.; Shinohara, Y.; Otobe, T.; Bertsch, G. F. Time-dependent density functional theory for strong electromagnetic fields in crystalline solids. *Phys. Rev. B* **2012**, *85*, No. 045134.

(136) Sigmund, P. *Particle Penetration and Radiation Effects*; Springer, 2006.

(137) Ziegler, J. F.; Littmark, U.; Biersack, J. P. *The Stopping and Range of Ions in Solids*; Pergamon: New York, 1985; 321 p.

(138) Zhang, Y.; et al. Influence of chemical disorder on energy dissipation and defect evolution in advanced alloys. *J. Mater. Res.* **2016**, *31*, 2363–2375.

(139) Haussalo, P.; Nordlund, K.; Keinonen, J. Stopping of 5−100 keV helium in tantalum, niobium, tungsten, and AISI 316L steel. *Nucl. Instrum. Methods Phys. Res., Sect. B* **1996**, *111*, 1−6.

(140) Caporaso, G. J.; Chen, Y.-J.; Sampayan, S. E. The Dielectric Wall Accelerator. *Rev. Accel. Sci. Technol.* **2009**, *02*, 253−263.

(141) Thomson, J. J. XLII. Ionization by moving electrified particles. *London, Edinburgh Dublin Philos. Mag. J. Sci.* **1912**, *23*, 449−457.

(142) Darwin, C. X. C. A theory of the absorption and scattering of the alpha rays. *London, Edinburgh Dublin Philos. Mag. J. Sci.* **1912**, *23*, 901−920.

(143) Bohr, N. I. On the constitution of atoms and molecules. *London, Edinburgh Dublin Philos. Mag. J. Sci.* **1913**, *26*, 1−25.

(144) Bethe, H. Zur Theorie des Durchgangs schneller Korpuskular-strahlen durch Materie. *Ann. Phys.* **1930**, *397*, 325−400.

(145) Fermi, E.; Teller, E. The Capture of Negative Mesotrons in Matter. *Phys. Rev.* **1947**, *72*, 399−408.

(146) Lindhard, J.; Winther, A. Stopping Power of Electron Gas and Equipartition Rule. *Mat.-Fys. Medd.—K. Dan. Vidensk. Selsk.* **1964**, *34*, 1−24.

(147) Correa, A. A.; Kohanoff, J.; Artacho, E.; Sánchez-Portal, D.; Caro, A. Nonadiabatic Forces in Ion-Solid Interactions: The Initial Stages of Radiation Damage. *Phys. Rev. Lett.* **2012**, *108*, No. 213201.

(148) Quashie, E. E.; Saha, B. C.; Correa, A. A. Electronic band structure effects in the stopping of protons in copper. *Phys. Rev. B* **2016**, *94*, No. 155403.

(149) Ullah, R.; Artacho, E.; Correa, A. A. Core Electrons in the Electronic Stopping of Heavy Ions. *Phys. Rev. Lett.* **2018**, *121*, No. 116401.

(150) Correa, A. A. Calculating electronic stopping power in materials from first principles. *Comput. Mater. Sci.* **2018**, *150*, 291−303.

(151) Pruneda, J. M.; Sánchez-Portal, D.; Arnau, A.; Juaristi, J. I.; Artacho, E. Electronic Stopping Power in LiF from First Principles. *Phys. Rev. Lett.* **2007**, *99*, No. 235501.

(152) Lin, L. Adaptively compressed exchange operator. *J. Chem. Theory Comput.* **2016**, *12*, 2242−2249.

(153) Jia, W.; Lin, L. Fast real-time time-dependent hybrid functional calculations with the parallel transport gauge and the adaptively compressed exchange formulation. *Comput. Phys. Commun.* **2019**, *240*, 21−29.

(154) Carnimeo, I.; Baroni, S.; Giannozzi, P. Fast hybrid density-functional computations using plane-wave basis sets. *Electron. Struct.* **2019**, *1*, No. 015009.

(155) Ehrke, H.; et al. Photoinduced Melting of Antiferromagnetic Order in $La_{0.5}Sr_{1.5}MnO_4$ Measured Using Ultrafast Resonant Soft X-Ray Diffraction. *Phys. Rev. Lett.* **2011**, *106*, No. 217401.

(156) Li, T.; Patz, A.; Mouchliadis, L.; Yan, J.; Lograsso, T. A.; Perakis, I. E.; Wang, J. Femtosecond switching of magnetism via strongly correlated spin−charge quantum excitations. *Nature* **2013**, *496*, 69−73.

(157) Rajpurohit, S.; Jooss, C.; Blöchl, P. E. Evolution of the magnetic and polaronic order of $Pr_{1/2}Ca_{1/2}MnO_3$ following an ultrashort light pulse. *Phys. Rev. B* **2020**, *102*, No. 014302.

(158) Morrison, V. R.; Chatelain, R. P.; Tiwari, K. L.; Hendaoui, A.; Bruhács, A.; Chaker, M.; Siwick, B. J. A photoinduced metal-like phase of monoclinic VO2 revealed by ultrafast electron diffraction. *Science* **2014**, *346*, 445−448.

(159) Otto, M. R.; René de Cotret, L. P.; Valverde-Chavez, D. A.; Tiwari, K. L.; Émond, N.; Chaker, M.; Cooke, D. G.; Siwick, B. J. How optical excitation controls the structure and properties of vanadium dioxide. *Proc. Natl. Acad. Sci. U.S.A.* **2019**, *116*, 450−455.

(160) Caviglia, A. D.; et al. Photoinduced melting of magnetic order in the correlated electron insulator $NdNiO_3$. *Phys. Rev. B* **2013**, *88*, No. 220401.

(161) De Filippis, G.; Cataudella, V.; Nowadnick, E. A.; Devereaux, T. P.; Mishchenko, A. S.; Nagaosa, N. Quantum Dynamics of the Hubbard-Holstein Model in Equilibrium and Nonequilibrium: Application to Pump-Probe Phenomena. *Phys. Rev. Lett.* **2012**, *109*, No. 176402.

(162) Werner, P.; Eckstein, M. Field-induced polaron formation in the Holstein-Hubbard model. *EPL* **2015**, *109*, No. 37002.

(163) Köhler, T.; Rajpurohit, S.; Schumann, O.; Paeckel, S.; Biebl, F. R. A.; Sotoudeh, M.; Kramer, S. C.; Blöchl, P. E.; Kehrein, S.; Manmana,

S. R. Relaxation of photoexcitations in polaron-induced magnetic microstructures. *Phys. Rev. B* **2018**, *97*, No. 235120.

(164) Stolpp, J.; Herbrych, J.; Dorfner, F.; Dagotto, E.; Heidrich-Meisner, F. Charge-density-wave melting in the one-dimensional Holstein model. *Phys. Rev. B* **2020**, *101*, No. 035134.

(165) Rajpurohit, S.; Tan, L. Z.; Jooss, C.; Blöchl, P. E. Ultrafast spin-nematic and ferroelectric phase transitions induced by femtosecond light pulses. *Phys. Rev. B* **2020**, *102*, No. 174430.

(166) Stoica, V. A.; Laanait, N.; Dai, C.; Hong, Z.; Yuan, Y.; Zhang, Z.; Lei, S.; Mccarter, M. R.; Yadav, A.; Damodaran, A. R.; et al. Optical creation of a supercrystal with three-dimensional nanoscale periodicity. *Nat. Mater.* **2019**, *18*, 377−383.

(167) Guzelturk, B.; Mei, A. B.; Zhang, L.; Tan, L. Z.; Donahue, P.; Singh, A. G.; Schlom, D. G.; Martin, L. W.; Lindenberg, A. M. Light-Induced Currents at Domain Walls in Multiferroic $BiFeO_3$. *Nano Lett.* **2020**, *20*, 145−151.

(168) Trigo, M.; et al. Fourier-transform inelastic X-ray scattering from time- and momentum-dependent phonon−phonon correlations. *Nat. Phys.* **2013**, *9*, 790−794.

(169) Sie, E. J.; Nyby, C. M.; Pemmaraju, C. D.; Park, S. J.; Shen, X.; Yang, J.; Hoffmann, M. C.; Ofori-Okai, B. K.; Li, R.; Reid, A. H.; et al. An ultrafast symmetry switch in a Weyl semimetal. *Nature* **2019**, *565*, 61−66.

(170) Dresselhaus, G. Spin-Orbit Coupling Effects in Zinc Blende Structures. *Phys. Rev.* **1955**, *100*, 580−586.

(171) Bychkov, Y. A.; Rashba, E. I. Properties of a 2D electron gas with lifted spectral degeneracy. *JETP Lett.* **1984**, *39*, 78.

(172) Helgaker, T.; Watson, M.; Handy, N. C. Analytical calculation of nuclear magnetic resonance indirect spin−spin coupling constants at the generalized gradient approximation and hybrid levels of density-functional theory. *J. Chem. Phys.* **2000**, *113*, 9402−9409.

(173) Melo, J. I.; Ruiz de Azúa, M. C.; Peralta, J. E.; Scuseria, G. E. Relativistic calculation of indirect NMR spin-spin couplings using the Douglas-Kroll-Hess approximation. *J. Chem. Phys.* **2005**, *123*, No. 204112.

(174) Gugler, J.; Astner, T.; Angerer, A.; Schmiedmayer, J.; Majer, J.; Mohn, P. Ab initio calculation of the spin lattice relaxation time T_1 for nitrogen-vacancy centers in diamond. *Phys. Rev. B* **2018**, *98*, No. 214442.

(175) Stojchevska, L.; Vaskivskyi, I.; Mertelj, T.; Kusar, P.; Svetin, D.; Brazovskii, S.; Mihailovic, D. Ultrafast Switching to a Stable Hidden Quantum State in an Electronic Crystal. *Science* **2014**, *344*, 177−180.

(176) Lee, S.-H.; Goh, J. S.; Cho, D. Origin of the Insulating Phase and First-Order Metal-Insulator Transition in $1T$−TaS_2. *Phys. Rev. Lett.* **2019**, *122*, No. 106404.

(177) Siddiqui, K. M.; Durham, D. B.; Cropp, F.; Ophus, C.; Rajpurohit, S.; Zhu, Y.; Carlström, J. D.; Stavrakas, C.; Mao, Z.; Raja, A.; Musumeci, P.; Tan, L. Z.; Minor, A. M.; Filippetto, D.; Kaindl, R. A. Ultrafast optical melting of trimer superstructure in layered 1T'-TaTe2. 2020. arXiv:2009.02891. arXiv.org e-Print archive. https://arxiv.org/abs/2009.02891.

(178) Ligges, M.; Avigo, I.; Golež, D.; Strand, H. U. R.; Beyazit, Y.; Hanff, K.; Diekmann, F.; Stojchevska, L.; Kalläne, M.; Zhou, P.; Rossnagel, K.; Eckstein, M.; Werner, P.; Bovensiepen, U. Ultrafast Doublon Dynamics in Photoexcited $1T$-TaS_2. *Phys. Rev. Lett.* **2018**, *120*, No. 166401.

(179) Storeck, G.; Gerrit Horstmann, J.; Diekmann, T.; Vogelgesang, S.; von Witte, G.; Yalunin, S.; Rossnagel, K.; Ropers, C. Structural Dynamics of Incommensurate Charge-Density Waves tracked by Ultrafast Low-Energy Electron Diffraction. 2019, arXiv:1909.10793. arXiv.org e-Print archive. https://arxiv.org/abs/1909.10793.

(180) Starykh, O. A. Unusual ordered phases of highly frustrated magnets: a review. *Rep. Prog. Phys.* **2015**, *78*, No. 052502.

(181) Tapavicza, E.; Bellchambers, G. D.; Vincent, J. C.; Furche, F. Ab initio non-adiabatic molecular dynamics. *Phys. Chem. Chem. Phys.* **2013**, *15*, 18336−18348.

(182) Curchod, B. F. E.; Martínez, T. J. Ab Initio Nonadiabatic Quantum Molecular Dynamics. *Chem. Rev.* **2018**, *118*, 3305−3336.

(183) Tamm, A.; Caro, M.; Caro, A.; Samolyuk, G.; Klintenberg, M.; Correa, A. A. Langevin Dynamics with Spatial Correlations as a Model for Electron-Phonon Coupling. *Phys. Rev. Lett.* **2018**, *120*, No. 185501.

(184) Belinicher, V. I.; Sturman, B. I. The photogalvanic effect in media lacking a center of symmetry. *Sov. Phys. Usp.* **1980**, *23*, 199−223.

(185) Sipe, J. E.; Shkrebtii, A. I. Second-order optical response in semiconductors. *Phys. Rev. B* **2000**, *61*, 5337−5352.

(186) Tan, L. Z.; Zheng, F.; Young, S. M.; Wang, F.; Liu, S.; Rappe, A. M. Shift current bulk photovoltaic effect in polar materials—hybrid and oxide perovskites and beyond. *npj Comput. Mater.* **2016**, *2*, No. 16026.

(187) Fregoso, B. M.; Muniz, R. A.; Sipe, J. E. Jerk Current: A Novel Bulk Photovoltaic Effect. *Phys. Rev. Lett.* **2018**, *121*, No. 176604.

(188) Andrade, X.; Hamel, S.; Correa, A. A. Negative differential conductivity in liquid aluminum from real-time quantum simulations. *Eur. Phys. J. B* **2018**, *91*, No. 229.

(189) Parker, D. E.; Morimoto, T.; Orenstein, J.; Moore, J. E. Diagrammatic approach to nonlinear optical response with application to Weyl semimetals. *Phys. Rev. B* **2019**, *99*, No. 045121.

(190) Rajpurohit, S.; Das Pemmaraju, C.; Ogitsu, T.; Tan, L. Z. A Non-perturbative Study of Bulk Photovoltaic Effect Enhanced by an Optically Induced Phase Transition. 2021, arXiv:2105.11310. arXiv.org e-Print archive. https://arxiv.org/abs/2105.11310.

(191) Schleder, G. R.; Padilha, A. C.; Acosta, C. M.; Costa, M.; Fazzio, A. From DFT to machine learning: recent approaches to materials science—a review. *J. Phys.: Mater.* **2019**, *2*, No. 032001.

(192) Kasim, M. F.; Vinko, S. M. Learning the Exchange-Correlation Functional from Nature with Fully Differentiable Density Functional Theory. 2021, arXiv:2102.04229. arXiv.org e-Print archive. https://arxiv.org/abs/2102.04229.

(193) Bartlett, R. A.; Gay, D. M.; Phipps, E. T. In *Automatic Differentiation of C++ Codes for Large-Scale Scientific Computing*, International Conference on Computational Science, 2006; pp 525−532.

(194) Hogan, R. J. Fast reverse-mode automatic differentiation using expression templates in C++. *ACM Trans. Math. Software* **2014**, *40*, 1−16.