Computer Physics Communications 210 (2017) 181–192

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

PCTDSE: A parallel Cartesian-grid-based TDSE solver for modeling laser–atom interactions☆

Yongsheng Fu [a,*], Jiaolong Zeng [a,b], Jianmin Yuan [a,b]

[a] College of Science, National University of Defense Technology, Changsha 410073, People's Republic of China
[b] IFSA Collaborative Innovation Center, Shanghai Jiao Tong University, Shanghai 200240, People's Republic of China

ARTICLE INFO

Article history:
Received 12 February 2016
Received in revised form
28 August 2016
Accepted 24 September 2016
Available online 3 October 2016

Keywords:
Cartesian grid
TDSE
Laser–atom interactions
2D decomposition

ABSTRACT

We present a parallel Cartesian-grid-based time-dependent Schrödinger equation (TDSE) solver for modeling laser–atom interactions. It can simulate the single-electron dynamics of atoms in arbitrary time-dependent vector potentials. We use a split-operator method combined with fast Fourier transforms (FFT), on a three-dimensional (3D) Cartesian grid. Parallelization is realized using a 2D decomposition strategy based on the Message Passing Interface (MPI) library, which results in a good parallel scaling on modern supercomputers. We give simple applications for the hydrogen atom using the benchmark problems coming from the references and obtain repeatable results. The extensions to other laser–atom systems are straightforward with minimal modifications of the source code.

Program summary

Program title: PCTDSE

Catalogue identifier: AFBM_v1_0

Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AFBM_v1_0.html

Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland

Licensing provisions: BSD 3 - Clause

No. of lines in distributed program, including test data, etc.: 398662

No. of bytes in distributed program, including test data, etc.: 6355779

Distribution format: tar.gz

Programming language: Fortran 2003.

Computer: Distributed memory machines.

Operating system: Unix-like system.

RAM: Depends on the size of the Cartesian grid

Classification: 2.1, 2.2, 2.5.

External routines: 2DECOMP&FFT; FFTW; MPI

Nature of problem:

Simulate the single-electron dynamics of atoms under the strong fields of modern lasers according to the time-dependent Schrödinger equation.

Solution method:

The package solves the TDSE in the FFT-split-operator method, employing the split-operator method to approximate the time evolution operator and fast Fourier transforms to calculate the spatial derivatives.

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail addresses: fuyongsheng@nudt.edu.cn (Y. Fu), jlzeng@nudt.edu.cn (J. Zeng), jmyuan@nudt.edu.cn (J. Yuan).

http://dx.doi.org/10.1016/j.cpc.2016.09.016

Restrictions:

The code is restricted to problems where the atoms are in the single-active-electron approximation and the lasers are in the dipole approximation. It is also limited by the CPU time and memory that one can afford.

Unusual features:

We adopt the parallel strategy where the Cartesian grid is distributed among processors using a 2D decomposition, which has no limitation for large-scale simulations.

Running time:

The running time depends on the size of the grid, the number of time step, the number of processors, and the choice of the processor grid, ranging from a few hours to several days.

1. Introduction

The interactions between atoms and laser fields are of fundamental interest in atomic physics [1] since the first realization of the laser in 1960. The present laser technologies can easily provide the electric field amplitudes of the same order as the Coulomb field in atoms, which brings the electron dynamics into the non-perturbative regime and promotes the experimental studies on various strong-field processes including Above-threshold ionization (ATI) [2], high harmonic generation (HHG) [3], Terahertz (THZ) radiation [4]. In the theoretical aspect, the time-dependent Schrödinger equation (TDSE) governs the behavior of electrons in the complex fields of atom and laser. The quantum-mechanical wave function contains all the information of the systems, compared with other approximately analytic models [5].

The numerical solutions of the TDSE give *ab initio* simulations of the electron dynamics, and have been extensively carried out for single- [6], two- [7], and many-electron atoms [8], for long-wavelength optical lasers [9] and short-wavelength X-ray lasers [10]. Different levels of approximates have been made for the mentioned situations. The single-electron (hydrogen) atom allows an exact numerical solution of the three-dimensional (3D) TDSE. There also exists the recently developed six-dimensional two-electron TDSE program for the He atom [11]. For many-electron atoms, in addition to the time-dependent Hartree–Fock (TDHF) method [12], time-dependent density functional theory (TDDFT) [13], the single-active-electron (SAE) approximation [14] is widely used when multiple electron excitations are not important. When describing the effect of the tightly bounded inner electrons on the valence electron as a model potential, the problem can reduce to the single-electron situation. In the aspect of interaction with lasers, the dipole approximate is always valid for optical lasers which are treated as homogeneous time-dependent electric fields. The corrections by non-dipole terms [15] should be considered when X-ray sources are used.

In spite of the increase of computing power, it is difficult to numerically solve the 3D TDSE in a large scale, even for the single-electron situation, hampered by the high demands on resources. Parallel calculations are necessary. Numerical approaches like the finite-difference method [16], finite-element method [17], matrix iteration method [18], split-operator method [19] have been used in many attempts to solve the equation. Not only spherical coordinates but also Cartesian coordinates appear in the respective implementations. It is accepted that Cartesian coordinates are more convenient for parallelization using the domain decomposition strategies [20].

In this work, we present a parallel Cartesian-grid-based TDSE solver for modeling laser–atom interactions. We restrict the atom in the SAE approximation interacting with the laser in the dipole approximate. We make use of the split-operator (SO) method assisted by Fast Fourier Transforms (FFT), which was firstly introduced by Feit et al. [21]. Similar method was just adopted by Mocken and Keitel [22] to solve the time-dependent Dirac equation in $2+1$ dimensions and by Dion et al. [23] to simulate the wave-packet dynamics according to the TDSE. Different from the work of Dion et al., the Cartesian grid here is distributed among cores using a 2D decomposition which has no limitation for large-scale simulations.

The remaining part of this paper is arranged as follows. In Section 2, we introduce the split-operator method in Cartesian coordinates, which lays the theoretical foundation of our numerical solution of the TDSE. In Section 3, the parallel strategy in the implementation is exhibited. The calculation details are present in Section 4. In Section 5, we give a brief introduction on the external library which our code strongly relies on. Following that, an elaborate description of the code package is presented in Section 6. The parallel scaling is tested in Section 7. Four examples illustrating the applications of the code are given in Section 8. We draw the conclusions in the last section. Atomic units are used throughout the paper.

2. Split-operator method in the Cartesian grid

2.1. Split-operator method

In quantum mechanics, the time-dependent Schrödinger equation (TDSE)

$$i\frac{\partial}{\partial t}\Psi(\mathbf{r},t)=\hat{H}(t)\Psi(\mathbf{r},t) \tag{1}$$

gives a description of the system evolving with time. Here $\Psi(\mathbf{r},t)$ is the spatial wave function with $\mathbf{r}$ and t being the space–time coordinates, and $\hat{H}(t)$ is the Hamiltonian operator.

Considering the interaction of an atomic electron with the classical electromagnetic radiation in the Coulomb gauge, the Hamiltonian in general reads

$$\hat{H}(t)=\frac{1}{2}\left[\hat{\mathbf{p}}+\frac{1}{c}\mathbf{A}(\mathbf{r},t)\right]^2+V(r), \tag{2}$$

where $\mathbf{A}(\mathbf{r},t)$ are the electromagnetic vector scalar potential, $\hat{p}=-i\nabla$ is the canonical momentum operator, c is the speed of light in vacuum, and $V(r)$ is the atomic potential.

A formal solution to Eq. (1) is expressed by the time-evolution operator

$$\hat{U}(t,t_0)=\hat{T}\exp\left(-i\int_{t_0}^{t}\hat{H}(t')dt'\right), \tag{3}$$

which connects the wave function at any time t to the one at the initial time t_0 by an unitary transformation as

$$\Psi(\mathbf{r}, t) = \hat{U}(t, t_0)\Psi(\mathbf{r}, t_0). \tag{4}$$

$\hat{T}$ denotes the time ordering operator.

Due to the generally spatial dependence of the vector potential, there is no way to make a division between the spatial-dependent dependent part and the momentum-dependent part of the Hamiltonian. Things change when we consider now the so-called dipole approximation $\mathbf{A}(\mathbf{r}, t) \approx \mathbf{A}(t)$, i.e., a purely time-dependent vector potential. A split of the Hamiltonian is thus possible

$$\hat{H}(t) = \hat{H}_1(t) + \hat{H}_2, \tag{5}$$

where

$$\hat{H}_1(t) = \frac{1}{2}\hat{\mathbf{p}}^2 + \frac{1}{c}\hat{\mathbf{p}} \cdot \mathbf{A}(t) + \frac{1}{2c^2}\mathbf{A}^2(t) \tag{6}$$

and

$$\hat{H}_2 = V(r) \tag{7}$$

have the respective momentum and spatial dependence merely.

The task of the split-operator method is to factorize the time-evolution operator of Eq. (3) at an enough short time interval into products of a series of operators. With the split scheme in Eq. (5), the time-evolution operator for a small time step Δt is factorized as

$$U(t + \Delta t) = \exp(-i\hat{H}_2\Delta t/2) \exp\left(-i\int_t^{t+\Delta t} \hat{H}_1(t')dt'\right) \times \exp(-i\hat{H}_2\Delta t/2) + O\left((\Delta t)^3\right), \tag{8}$$

which allows the propagation of a wave function accurate to second-order

$$\Psi(\mathbf{r}, t + \Delta t) \approx \exp(-i\hat{H}_2\Delta t/2) \exp\left(-i\Delta t\hat{H}_1(t + \Delta t/2)\right) \times \exp(-i\hat{H}_2\Delta t/2)\Psi(\mathbf{r}, t). \tag{9}$$

Note that we have replaced the integral term in Eq. (8) by the midpoint formula. The resulting exponential operators become simple multiplication factors in the position or momentum space, rending the operations trivial in their respective spaces. By virtue of this, Eq. (9) can be rewritten as

$$\Psi(\mathbf{r}, t + \Delta t) \approx \exp(-i\hat{H}_2\Delta t/2)\mathcal{F}^{-1} \times \exp\left(-i\Delta t\hat{H}_1(t + \Delta t/2)\right)\mathcal{F} \exp(-i\hat{H}_2\Delta t/2)\Psi(\mathbf{r}, t) \tag{10}$$

where $\mathcal{F}$ and its inverse $\mathcal{F}^{-1}$ are the Fourier transforms from the position into momentum space and vice versa.

Eq. (5) and (10) together make up the main aspect of the SO method. In a numerical implementation, the initial wave function is discretized on a Cartesian grid. Its time propagation is calculated by applying Eq. (10) successively until the final time is reached. The Fourier transforms are accomplished by Fast-Fourier-transform (FFT) algorithms.

2.2. Cartesian grid setup

We solve the TDSE on the Cartesian grid. Both the spatial and momentum grids are involved in the implementation of the SO method. Suppose that the simulation is restricted in a finite box $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}]$ in the position space, the spatial wave functions, being 3D complex arrays, are defined on the grid points

$$\begin{aligned} x_i &= x_{\min} + (i-1)\Delta x, \quad i = 1, \ldots, n_x \\ y_j &= y_{\min} + (j-1)\Delta y, \quad j = 1, \ldots, n_y \\ z_k &= z_{\min} + (k-1)\Delta z, \quad k = 1, \ldots, n_z \end{aligned} \tag{11}$$

where n_x, n_y, n_z are the numbers of the grid points, and

$$\begin{aligned} \Delta x &= (x_{\max} - x_{\min})/(n_x - 1) \\ \Delta y &= (y_{\max} - y_{\min})/(n_y - 1) \\ \Delta z &= (z_{\max} - z_{\min})/(n_z - 1) \end{aligned} \tag{12}$$

are the grid steps.

The momentum wave functions are connected with the spatial ones by the complex discrete Fourier transforms (DFTs). For human viewing of a spectrum, it is often convenient to put the zero-momentum component at the center of the output array and choose the corresponding momentum grid

$$\begin{aligned} p_{x,i} &= 2\pi i/(n_x\Delta x), \quad i = -n_x/2 + 1, \ldots, n_x/2 \\ p_{y,j} &= 2\pi j/(n_y\Delta y), \quad j = -n_y/2 + 1, \ldots, n_y/2 \\ p_{z,k} &= 2\pi k/(n_z\Delta z), \quad k = -n_z/2 + 1, \ldots, n_z/2. \end{aligned} \tag{13}$$

However, this arrangement does not obey the storage order of the arrays expected by the FFT routines that the zero-frequency component is in the first element. As a result, we need to deal with another momentum grid as

$$\begin{aligned} p_{x,i'} &= 2\pi i'/(n_x\Delta x), \quad i' = 0, \ldots, n_x/2, -n_x/2 + 1, \ldots, -1 \\ p_{y,j'} &= 2\pi j'/(n_y\Delta y), \quad j' = 0, \ldots, n_y/2, -n_y/2 + 1, \ldots, -1 \\ p_{z,k'} &= 2\pi k'/(n_z\Delta z), \quad k' = 0, \ldots, n_z/2, -n_z/2 + 1, \ldots, -1. \end{aligned} \tag{14}$$

Note that Eq. (14) can build bridge to Eq. (13) by reordering the indices of the momentum coordinates along each dimension. To match the latter momentum grid, a half-space swapped array is used at all the intermediate time during the propagation except that it is time for visualization.

3. Parallel strategy

In this section, we present the scheme of our parallel implementation of the SO method on multi-core computer systems that support the Message Passing Interface (MPI) library [24]. The crucial points are the distribution of the discrete wave functions among the cores and FFTs of the distributed data.

3.1. Domain decomposition strategies

The domain decomposition strategies are suitable to divide the three-dimensional Cartesian grid where the wave functions live. The quite simple one is a one-dimensional (1D) decomposition, also called a slab decomposition. Here the 3D domain is split along only one direction (such as the x direction), making the whole y–z plane localized on every core, as shown in Fig. 1. The main drawback comes from the maximum number of cores limited by the grid size in the decomposed direction. For example, the limitation is N for a given cubic grid with N^3 points. The supercomputers nowadays always have more than ten thousands of cores. 1D decompositions not only confine the scale of parallelization, but also induces too much workload per core.

A 2D decomposition, also known as a pencil decomposition, is thought as superior to 1D decompositions. It decomposes a 3D domain by the process grid

$$N_{proc} = P_{row} \times P_{col}, \tag{15}$$

where N_{proc} is the total number of cores (processes) separated into two groups, see Fig. 2 for example. Different from that in Fig. 1, the 3D Cartesian grid is decomposed in both the x and y directions. The resulting state is called a z-pencil where the grid points along the z direction are local for every core. x- and y-pencils can be obtained in the similar manner. Given a cubic grid of size N^3, the maximum number of cores grows up to N^2. There is opportunity for our program which uses a 2D decomposition to make use of

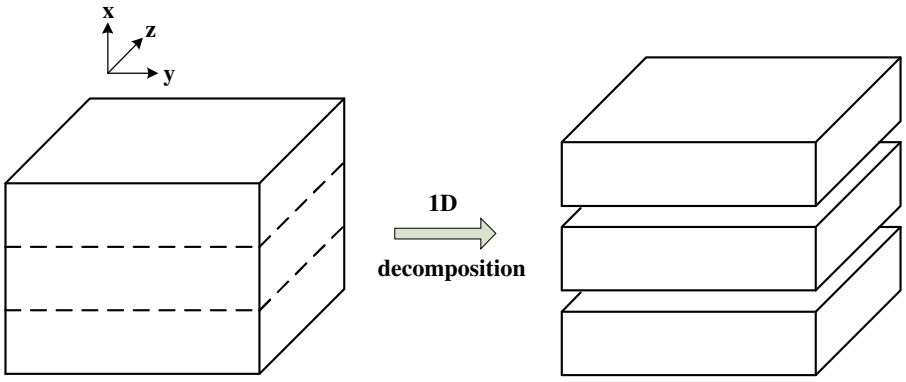

Fig. 1. A sketch of the 1D decomposition in the x direction using 3 cores: The cuboid on the left representing a 3D Cartesian grid that will be decomposed at the dashed lines; the resulting three parts on the right are distributed on the three cores.

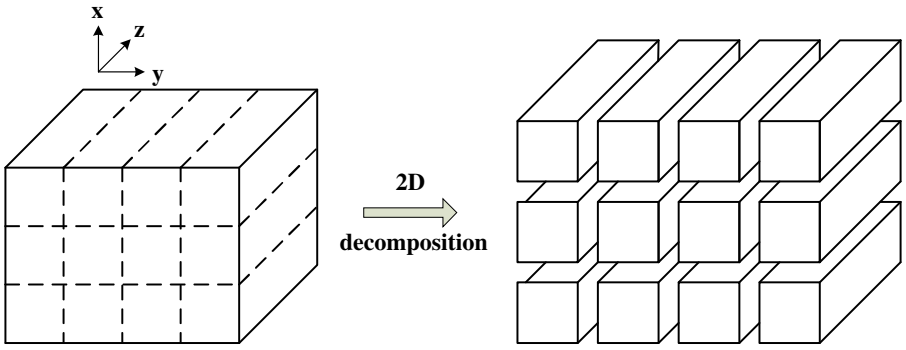

Fig. 2. A sketch of the 2D decomposition in the x and y directions using a 3×4 process grid: The cuboid on the left representing a 3D Cartesian grid that will be decomposed at the dashed lines; the resulting twelve parts on the right are distributed on the process grid.

as many cores as possible which are available from the modern supercomputers.

In practice, the numerical implementations of 2D decompositions can be realized using the MPI Cartesian Topology. 1D decompositions can be categorized as the special cases into the same framework.

3.2. Parallel FFT

All FFT algorithms compute the DFT of a 3D complex array as

$$\tilde{X}_{u,v,w} = \left(\sum_{n=0}^{N-1} \left(\sum_{m=0}^{M-1} \left(\sum_{l=0}^{L-1} X_{l,m,n} e^{-2\pi i u l/L} \right)_{(1)} \times e^{-2\pi i v m/M} \right)_{(2)} e^{-2\pi i w n/N} \right)_{(3)}, \tag{16}$$

and need the data on the global grid across all the cores.

The parallelization of FFT on the distributed data is motivated by the usual calculation of Eq. (16) in sequential three stages (as shown by the parentheses with the markers):

(1) 1D Fourier transform along the x direction;
(2) 1D Fourier transform along the y direction;
(3) 1D Fourier transform along the z direction.

For the 3D Cartesian grid using a 2D decomposition, each stage allows the application of serial algorithms on local memories, assisted by dynamically transposing data among cores from x- to y-pencil and from y- to z-pencil afterwards.

Highly optimized serial FFT algorithms have been implemented by many major hardware vendors, including Intel's MKL [25], IBM's ESSL [26], AMD's ACML [27], etc. There is also the most popular open-source FFTW library [28]. The transpositions are realized by the MPI_ALLTOALLV routine of the MPI library.

4. Calculation details

4.1. Ground state preparation

The ground state always serves as an initial condition of the time evolution of an atomic system according the TDSE, i.e., Eq. (1). It is the lowest eigenstate of the unperturbed Hamiltonian

$$\hat{H}_0 = \frac{\hat{\mathbf{p}}^2}{2} + V(r), \tag{17}$$

and is exactly known only for the hydrogenic potential $V(r) = -Z_{nuc}/r$ where Z_{nuc} is the atomic number. Numerical approaches are needed to find the ground states of arbitrary potentials. There are many ways to calculate the ground state for a given scalar potential. The simple one is the imaginary-time-propagation (ITP) method which allows a minimum of additional routines within our code. The scheme is to replace the real-time step Δt in Eq. (10) by an imaginary-time step $\Delta t \rightarrow -i\Delta\tau$, and implement a propagation in the imaginary time τ from a test wave function $\Psi(\mathbf{r}, 0)$.

The principle of the ITP method is easy to understand. Assume that we have $\hat{H}_0\phi_n(\mathbf{r}) = e_n\phi_n(\mathbf{r})$ with e_n and ϕ_n the eigenvalues and eigenstates. The wave function at any imaginary time τ can be expanded as

$$\Psi(\mathbf{r}, \tau) = e^{-\hat{H}_0\tau}\Psi(\mathbf{r}, 0) = \sum_n c_n e^{-e_n\tau}\phi_n(\mathbf{r}), \tag{18}$$

where c_n is the expansion coefficients. We note that the wave function is a summation of the complete eigenstates weighted by

the factor $e^{-e_n\tau}$. At large τ, the terms with $e_n > 0$ die out, while that with $e_n < 0$ diverge. The fastest divergent one corresponds to the minimum e_n. Thus the normalized $\Psi(\mathbf{r}, \tau)$ converges to the ground state, i.e., eigenstate with the minimal total energy.

4.2. High harmonic generation

The HHG spectrum of an atom along the direction of the unit vector $\mathbf{e}$ can be described in three alternative forms associated with the Fourier transforms of the time-dependent expectation values of the dipole moment $\mathbf{r}$, the dipole velocity $\dot{\mathbf{r}}$, and the dipole acceleration $\ddot{\mathbf{r}}$ (let $\zeta = \{\mathbf{r}, \dot{\mathbf{r}}, \ddot{\mathbf{r}}\}$):

$$S_\zeta^{\mathbf{e}}(\omega) = \left| \frac{1}{T_1} \int_0^{T_1} \mathbf{e} \cdot \langle \zeta(t) \rangle \, e^{-i\omega t} dt \right|^2, \tag{19}$$

where

$$\langle \mathbf{r}(t) \rangle = \langle \Psi(\mathbf{r}, t) | \mathbf{r} | \Psi(\mathbf{r}, t) \rangle, \tag{20}$$

$$\begin{aligned} \langle \dot{\mathbf{r}}(t) \rangle &= \left\langle \Psi(\mathbf{r}, t) | \hat{\mathbf{p}} + \frac{1}{c}\mathbf{A}(t) | \Psi(\mathbf{r}, t) \right\rangle \\ &= \left\langle \Psi(\mathbf{r}, t) | \mathcal{F}^{-1} \left[\mathbf{p} + \frac{1}{c}\mathbf{A}(t) \right] \mathcal{F} | \Psi(\mathbf{r}, t) \right\rangle, \end{aligned} \tag{21}$$

$$\langle \ddot{\mathbf{r}}(t) \rangle = \langle \Psi(\mathbf{r}, t) | -\nabla V(r) - \mathbf{E}(t) | \Psi(\mathbf{r}, t) \rangle. \tag{22}$$

The last equation is obtained from the Ehrenfest theorem [29].

4.3. Absorbing boundary

When numerically solving the TDSE in a fixed-size box, an absorbing function is need to avoid the unphysical reflection of the wave function at the boundary. The widely used form is [30]

$$M(r) = \begin{cases} 1, & r \le r_0 \\ \cos^{1/8}\left(\dfrac{\pi(r - r_0)}{2(r_{\max} - r_0)} \right), & r_0 < r < r_{\max} \\ 0, & \text{otherwise} \end{cases} \tag{23}$$

which forces the damping of the wave function in the range of r_0 to $r_{\max}$. The damped zone has to be far enough away from the nucleus so that the physical system is undisturbed by the absorbing boundary.

Quite often, we are only interested in the part of the wave function in the vicinity of the nucleus and discard the part absorbed by the boundary. The decrease of the norm of the wave function within the box can be interpreted as ionization. Suppose that we have propagated the wave function up to the final time t_f, the ionization probability is calculated as

$$P = 1 - \left\langle \Psi(\mathbf{r}, t_f) | \Psi(\mathbf{r}, t_f) \right\rangle. \tag{24}$$

We turn to the trick for the calculation of the photoelectron spectra in the same framework. Oppositely, it is now we care only for the part of the wave function absorbed by the boundary. The information about the photoelectron spectrum at the final time t_f is obtained from gathering the contributions from the absorbed electron flux

$$\Psi_{flux}(\mathbf{r}, t_c) = (1 - M(r))\Psi(\mathbf{r}, t_c) \tag{25}$$

at each intermediate time t_c. The photoelectron momentum distribution is expressed explicitly as [31]

$$\begin{aligned} f(\mathbf{p}, t_f) &= \left| \Psi_{total}(\mathbf{p}, t_f) \right|^2 \\ &= \left| \sum_{t_c} e^{-i \int_{t_c}^{t_f} \frac{[\mathbf{p} + \mathbf{A}(t)/c]^2}{2} dt} \mathcal{F} \Psi_{flux}(\mathbf{r}, t_c) \right|^2. \end{aligned} \tag{26}$$

Here the exponential part represents the Volkov evolution operator which propagates the wave function absorbed at the time t_c to t_f, $\mathcal{F}$ is the Fourier transform. Note that we have ignored the influence of the ion core on the ionized part.

For the calculations, the final time t_f has to be large enough to ensure that the ionized part and the bound part of the wave function are well separated.

5. Introduction to the 2DECOMP&FFT library

The 2DECOMP&FFT library, published by Li and Laizet [32], is designed for large-scale applications using 3D Cartesian meshes and spatially implicit numerical algorithms. It is written in Fortran and implements a general-purpose 2D decomposition along with 3D parallel FFTs depending on a choice of the external FFT libraries as mentioned in Section 3.2. It defines a set of global variables that can be used directly in applications and provides many user-friendly interfaces to developers by hiding communication details relying on MPI. Benefiting from the merits, the library is easily applied to our program making the code well scalable to hundreds of thousands of cores on recent supercomputers. In order to facilitate the use of our code, it is worthwhile to give a brief introduction on the main aspects of the library.

Following is the list of the global variables that are used in our code.

- mytype: integer variable to define the KIND of floating-point data. Double precision is turned on by using the pre-precessing flag -DDOUBLE_PREC at compile time.
- real_type, complex_type: MPI datatypes for real and complex numbers respectively.
- nproc: number of MPI processes.
- nrank: rank of the current MPI process.
- xsize(i), ysize(i), zsize(i), i=1,2,3: 1D arrays to store the sizes of the sub-domains held by the current process. The first letter refers to the pencil orientation and the three elements contain the sizes in the x, y and z directions, respectively.
- xstart(i), ystart(i), zstart(i), xend(i), yend(i), zend(i), i=1,2,3: the starting and ending indices for each sub-domains, as in the global coordinate system. The first letter refers to the pencil orientation.

The key subroutines that serve as the basic interfaces for 2D decomposition, 3D FFT and parallel I/O are also available.

– decomp_2d_init(nx, ny, nz, p_row, p_col)

  initializes the global variables of the library. nx, ny, nz are the size of 3D data on the Cartesian grid to be distributed over a 2D process grid nproc = p_row*p_col. The library can handle non-evenly distributed data with the only constraints as p_row $\leqslant$ min(nx, ny) and p_col $\leqslant$ min(ny, nz).
– decomp_2d_finalize

  cleans up the memory.
– transpose_x_to_y(in, out)

  performs the data transposition from x- to y-pencil. The input 3D array in and output array out can be either real or complex in ordinary ijk-order. Similar subroutines such as transpose_y_to_z(in, out), transpose_z_to_y(in, out), transpose_y_to_x(in, out) are also in the library.
– decomp_2d_fft_init

  prepares plans for the underlying FFT engines and defines temporary work spaces.
– decomp_2d_fft_finalize

  releases the memory used by the FFT interface.

– decomp_2d_fft_3d(in, out, direction)

  performs a 3D complex-to-complex FFT. direction can be either DECOMP_2D_FFT_FORWARD for forward transforms, or DECOMP_2D_FFT_BACKWARD for backward transforms. The input array in and output array out are both complex and have to be either a x- and z-pencil combination or vice versa, correspondingly.
– decomp_2d_write_one(ipencil, var, filename)

  writes a single 3D array to a unformatted file. ipencil describes the state of distributed (valid value are: 1 for x-pencil; 2 for y-pencil and 3 for z-pencil). var is the data array which can be either real or complex. filename is the name of the file to be written.
– decomp_2d_write_plane(ipencil, var, iplane, n, filename)

  writes a 2D slice of data from a 3D array to a unformatted file. iplane defines the direction of the desired 2D slice (valid value are: 1 for y–z plane; 2 for z–x plane and 3 for x–y plane). n specifies the n-th plane in global coordinate system. The other arguments have the same meanings as before.

In the library, all related variables and procedures are grouped in the three modules: decomp_2d, decomp_2d_fft, and decomp_2d_io which should be used by applications.

6. Description of the package

The whole package is written in Fortran, being consistent with the 2DECOMP&FFT library and utilizing the object-oriented design.

6.1. Modules

6.1.1. The params module

The params module contains all the basic constants that are used throughout the package.

6.1.2. The laser module

The laser module defines the laser vector potential and electric field. It will be used by the schrodinger module and must be coded depending on the practical problem considered.

The contained variables are:

- f0: electric field amplitude of the laser, in atomic units.
- omega: angular frequency of the laser, in atomic units.
- ipulse: integer to specify the pulse envelopes. Currently, this module implements three kinds of envelopes corresponding to the examples in Section 8. If ipulse=1, the laser vector potential has a 3-cycle $\sin^2$ envelope. If ipulse=2, the laser vector potential has a $\cos^4$ envelope. If ipulse=3, the laser vector potential has a 10-cycle $\sin^2$ envelope.

The contained subroutines

– electric_field(t, f1, f2, f3)

  gives the electric field components f1, f2 and f3 at the time t.
– vector_potential(t, v1, v2, v3)

  gives the vector potential components v1, v2 and v3 at the time t.

6.1.3. The atom module

The atom module provides the atom potential and its partial derivatives. It must be coded according the atomic system in study.

The two functions

– atom_potential(x, y, z)

  returns the atom potential at the Cartesian coordinates. Currently, the pure Coulomb potential is adopted for modeling the hydrogen atom.
– diff_pot(dir, x, y, z)

  returns the partial derivatives of the atom potential at the Cartesian coordinates. dir=1, 2 or 3 specifies the partial derivative in the x, y or z direction, respectively.

6.1.4. The grid module

The grid module defines the spatial and momentum grids where the wave functions live. In the parallel implementation, they are distributed among cores by the 2D process grid. Considering the outputs from the FFT routines, a second momentum grid with the coordinates in a reversed order is also defined.

All the related variables are:

- p_row, p_col: 2D process grid.
- nx, ny, nz: number of grid points in each dimension.
- min_x, max_x, min_y, max_y, min_z, max_z: bounds of the simulation box $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}]$.
- dx, dy, dz: grid steps Δx, Δy, and Δz in the position space, see Eq. (12).
- dk1, dk2, dk3: grid steps $2\pi/(n_x\Delta x)$, $2\pi/(n_y\Delta y)$, and $2\pi/(n_z\Delta z)$ in the momentum space.
- r1, r2, r3: 1D Arrays containing the spatial coordinates x_i, y_j, and z_k, see Eq. (11).
- k1, k2, k3: 1D Arrays containing the momentum coordinates of in-order $p_{x,i}$, $p_{y,j}$, and $p_{z,k}$, see Eq. (13).
- k1_tmp, k2_tmp, k3_tmp: 1D Arrays containing the momentum coordinates of reversed order $p_{x,i'}$, $p_{y,j'}$, and $p_{z,k'}$, see Eq. (14).

The main subroutines

– grid_int

  initializes the spatial and momentum grids.
– grid_finalize

  destroys the spatial and momentum grids.
– visualize_one(filename1, filename2, fid1, fid2, cplx, flag)

  visualizes the 3D array that was written to disk by the interface decomp_2d_write_one (see Section 5). filename1 is the name of the existent unformatted file. filename2 is the name of the objective ASCII file. fid1 and fid2 are the unit identifiers of the two files. cplx is the integer to specify a complex (cplx=1) or real (cplx=0) array. flag is the integer to decide the array in the position (flag=0) or momentum (flag=1) space.
– visualize_plane(filename1, filename2, fid1, fid2, iplane, cplx, flag)

  visualizes the 2D slice of a 3D array that was written to disk by the interface decomp_2d_write_plane (see Section 5). The same parameters are used except that iplane defines the direction of the desired 2D slice (1, 2 or 3 for the y–z, z–x or x–y plane).

6.1.5. The schrodinger module

The schrodinger module includes the main aspects of the split-operator method in solving the TDSE numerically.

Following is the list of the variables in the module. Considering that the parallel FFT interface involves the input and output arrays with different pencil orientations, as described in Section 5, both the x-pencil and z-pencil arrays are defined properly.

- psi_r: 3D array in x-pencil to store the spatial wave function.
- psi_k: 3D array in x-pencil to store the momentum wave function, with the zero-momentum component at the center.
- psi_tmp: 3D array in z-pencil to store the momentum wave function, with the zero-momentum component in the first element.
- imtime: logical variable to distinguish a real- or imaginary-time propagation. ITP is implemented by setting the value true.
- dt: time step Δt of the time evolution.

- t_init, t_final: initial and final times.
- width: width of the absorbing boundary in each dimension. It is defined as dimensionless fraction on each side.
- photoelectron: logical variable to decide whether the ionized part is accumulated.
- psi_flux: 3D array in x-pencil to store the absorbed part $\Psi_{flux}(\mathbf{r}, t_c)$, see Eq. (25). It is only allocated when photoelectron is set true.
- psi_flux_total: 3D array in z-pencil to store the total of the absorbed parts $\Psi_{total}(\mathbf{p}, t_f)$, see Eq. (26). It is only allocated when photoelectron is set true.

The contained subroutines and functions

- schrodinger_init

  initializes the schrodinger propagation.
- schrodinger_finalize

  terminates the schrodinger propagation.
- propagator(t)

  propagates the wave function in one time step from t to $t + dt$.
- k_operator(t)

  applies the momentum-dependent operator exp $\left(-i\Delta t\hat{H}_1\right)$.
- r_operator

  applies the position-dependent operator exp $\left(-i\Delta t\hat{H}_2/2\right)$.
- damping_operator

  applies the absorbing boundary.
- sum_flux(t)

  gathers the absorbing part at the intermediate time t. It is called only when photoelectron is set true.
- get_psi_k(visu)

  generates the momentum wave function. if visu=0, only psi_tmp is generated for intermediate calculations. if visu=1, the data is saved in the new order to psi_k, which allows an easy visualization.
- swap

  rearranges the 3D array psi_tmp to psi_k.
- integrate(psi1, psi2, flag)

  being a function returns the inner product $\langle\Psi_1|\Psi_2\rangle$. flag=0 or 1 stands for the calculation in the position (psi1 and psi2 are in x-pencil) or momentum space (psi1 and psi2 are in z-pencil).
- norm(flag)

  being a function returns the norm of the wave function $\sqrt{\langle\Psi|\Psi\rangle}$. flag has the same meaning as before.
- renormalize(flag) renormalizes the wave function. flag has the same meaning as before.
- avg_r(x, y, z)

  calculates the expectation value of the position operator, given the spatial wave function psi_r. the three components are returned in x, y, and z.
- avg_p(t, px, py, pz)

  calculates the expectation value of the momentum operator, given the momentum wave function psi_tmp, the three components are returned in px, py, and pz.
- avg_a(t, ax, ay, az)

  calculates the expectation value of the acceleration operator, given the spatial wave function psi_r. the three components are returned in ax, ay, and az.
- get_energy()

  being a function returns the expectation value of the unperturbed Hamiltonian of Eq. (17). It is called when to prepare the ground state.

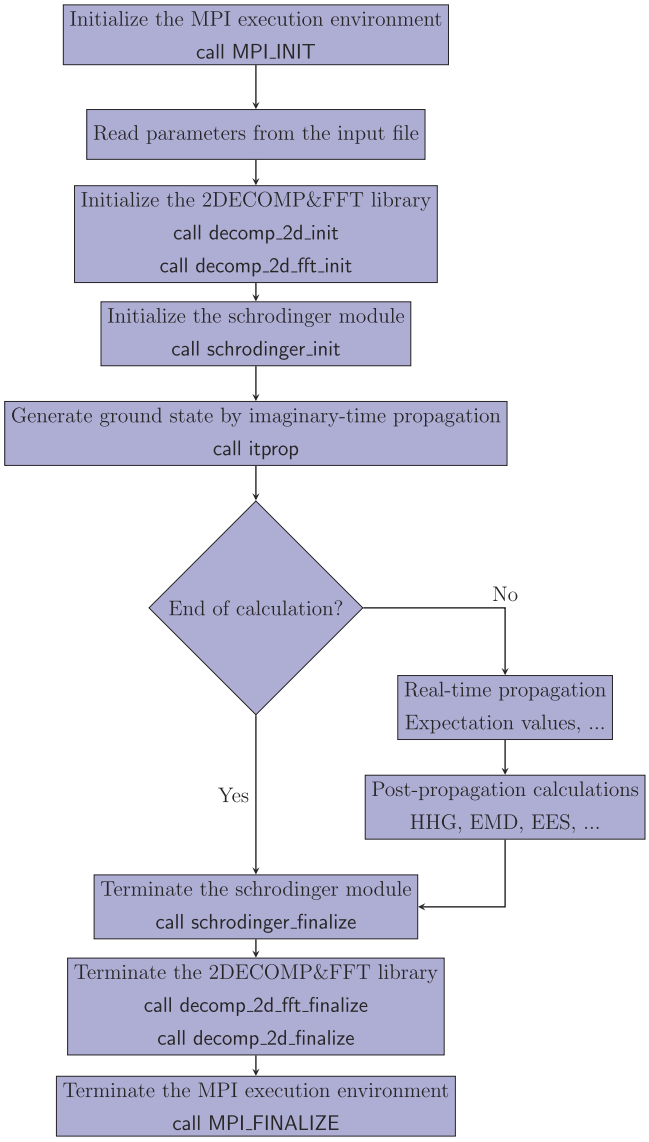

Fig. 3. Flow chart of the code. See text for details.

6.2. The main program

Four alternative main programs driver_itp, driver_hhg, driver_emd and driver_ees are present in the package. The program driver_itp gives a test on the imaginary-time propagation. The other three main programs are designed to model laser–atom interactions. The program driver_hhg calculates the high harmonic generation (HHG) of the hydrogen atom under a linearly polarized laser field. The program driver_emd simulates the strong-field ionization of the hydrogen atom by an circularly polarized laser in the attoclock configuration [7] and extracts the electron momentum distribution (EMD). The program driver_ees produces the electron energy spectrum (EES) from the ionization of atomic hydrogen in a strong infrared laser pulse. Fig. 3 shows a flow chart of the code.

6.3. The input data

Input data to the program take the form of namelist data. The following three namelists with the variables introduced in the previous subsection are read from standard input.

- namelist /grid_info/ p_row, p_col, nx, ny, nz, min_x, max_x, min_y, max_y, min_z, max_z

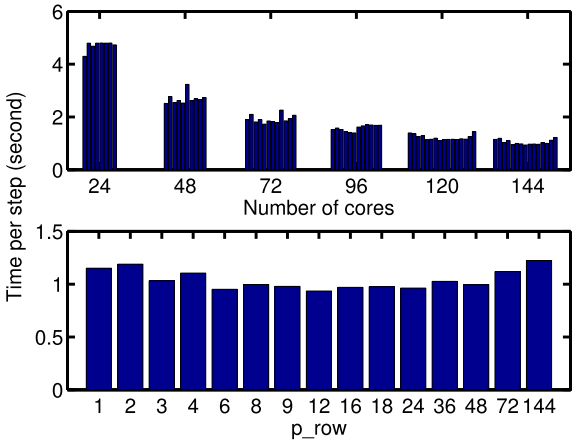

Fig. 4. (Color online) Top: Machine time per step in second as a function of the number of cores for all the options of process grid. Bottom: The magnifying part at 144 cores as a function of the process grid (only P_{row} is shown). The benchmark was performed on the Tianhe II supercomputer system at NUDT using the grid size 512^3.

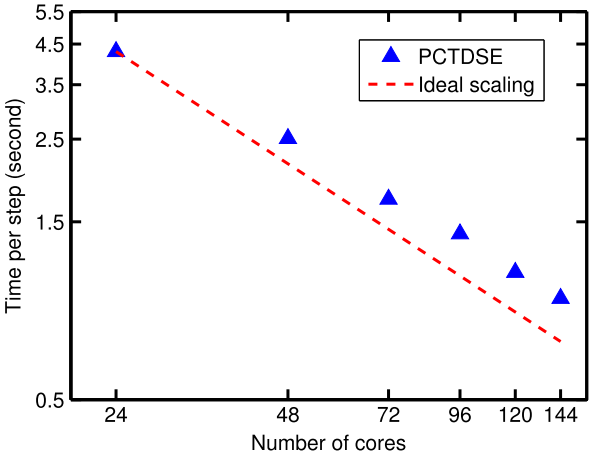

Fig. 5. (Color online) Parallel scaling of PCTDSE. The benchmark was performed on the Tianhe II supercomputer system at NUDT using the grid size 512^3.

- namelist /laser_info/ f0, omega, ipulse
- namelist /prop_info/ t_init, t_final, dt, width

6.4. Installation and user guide

PCTDSE is distributed as a source code and designed to run on most Unix-like systems. The distributed archive contains the *src* directory of the source files, the *bin* directory for the executable programs, the *test* directory, the main *Makefile* and a *README* file. The 2DECOMP&FFT source files (the usage of the library can be found in [32]) are also provided with the permission of the authors in the subdirectory *2decomp_fft* in *src*. FFTs are realized by the FFTW engines.

Compiling for the target environment is needed before the implementation. Suppose the package is extracted to the *./PCTDSE* directory. There is no need to change anything in the main *Makefile*. Instead, users should modify the platform-dependent *Makefile* in the *./PCTDSE/src* directory, including the compiler name (F90), flags (F90FLAGS) and FFTW library locations (INC and LIB). A fortran 2003 compiler is necessary to use the allocatable enhancement features. Once all set-up is done, type

make

from the main directory which will build the following four executable programs in the *./PCTDSE/bin* directory: *driver_itp*, *driver_hhg*, *driver_emd* and *driver_ees*.

Each input file *Input.in* of the examples is located in a separate subdirectory under the *./PCTDSE/test* directory. To run the programs, simply use the MPI rule

mpirun -np N program

where N is the number of cores.

7. Parallel scaling

The parallel performance of PCTDSE has been studied on the Tianhe II supercomputer system at National University of Defense Technology (NUDT), with each computer node comprising two 12-core Intel Xeon E5-2692 of 2.2 GHz. In addition to the number of cores, users also have the freedom to choose the 2D process grid $P_{row} \times P_{col}$. Depending on the hardware architecture, some options take on much better performance than others.

The benchmarks are performed using a typical grid size 512^3. We examine the machine time spent to implement the real-time propagation of one time step. The test programs run in fully-populated mode (full 24 cores per node) using from 1 to 6 nodes respectively, the corresponding number of cores being 24 to 144. For each number of cores, all the options of process grid are enumerated. The results are collected in the bar graph in Fig. 4. It can be seen that there is the best process grid at runtime. So application users are responsible for selecting the best options.

The minimum of the time among all process-grid options at each number of cores is selected and shown in Fig. 5. Good scaling can be achieved on up to 144 cores.

8. Examples

In this section, four examples corresponding to the four alternative main programs are shown to facilitate the applications of the code. All calculations are performed on the Tianhe II supercomputer system at NUDT.

8.1. Ground state via imaginary-time propagation

In this example, we calculate the non-relativistic ground state of the hydrogen atom by means of imaginary-time propagation, as explained in Section 4.1. Despite the fact that the solution is analytically known, we do this to validate the usefulness of this method.

We start the imaginary-time propagation from a test wave function that was initialized by random numbers. During the propagation, the expectation value of the unperturbed Hamiltonian, i.e., Eq. (17), is examined at each time step. The propagation stops when convergence to the lowest energy is satisfied.

The input file *Input.in* is prepared in the *./PCTDSE/examples/itp* subdirectory. To run this example, the user can enter this directory and type

mpirun -np N../../bin/driver_itp Input.in & > itp.log

which writes the calculation details to the file *itp.log*.

The typical calculation is made using a 3D grid of size 512^3 decomposed on a 4×4 process grid, covering ± 10 a.u. in each dimension. The time step is set 0.001 a.u. Below is part of the output file which shows the details of the imaginary-time propagation.

Step	Current energy	Current error
1	10.3113432435	0.7951196426E+003
2	1.7523638063	0.8558979437E+001
3	0.5618636882	0.1190500118E+001

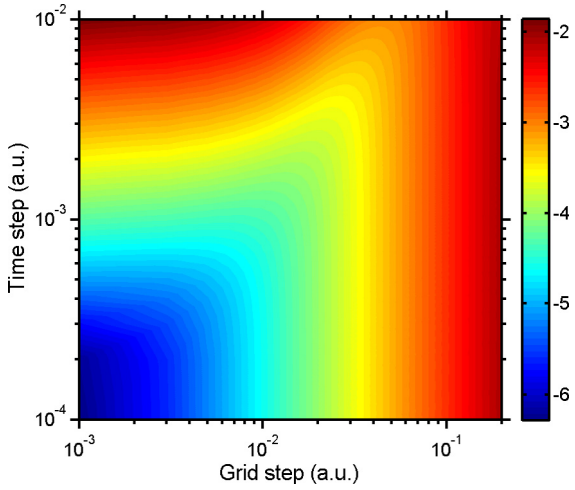

Fig. 6. (Color online) Logarithm of the absolute error of the ground-state energy as a function of grid and time steps.

4	0.2131478709	0.3487158173E+000
5	0.0713470329	0.1418008380E+000
6	0.0017957928	0.6955124014E−001
7	−0.0367483846	0.3854417733E−001
8	−0.0600296786	0.2328129402E−001
9	−0.0750227976	0.1499311898E−001
10	−0.0851696466	0.1014684905E−001
...	...	...
...	...	...
24536	−0.4996209148	0.1008941819E−009
24537	−0.4996209149	0.1009186623E−009
24538	−0.4996209150	0.1006702499E−009
24539	−0.4996209151	0.1004742400E−009
24540	−0.4996209152	0.1008646500E−009
24541	−0.4996209153	0.1005221462E−009
24542	−0.4996209154	0.1003587768E−009
24543	−0.4996209155	0.1001606575E−009
24544	−0.4996209156	0.1004450412E−009
24545	−0.4996209157	0.9990558381E−010

We can see that the energy converges with a precision of 10^{-10} after 24 545 steps of imaginary time. The machine time spent is about 24 h.

Because of the present implementation based on the Cartesian grid, we inevitably face the singularity of the Coulomb potential. The grid should avoid the singular point, but at the same time approach it as closely as possible. In this work, we have carefully tested the dependence of the results on the grid size and time steps. The tests were performed by varying both the grid size from 128^3 to 1024^3 (different grid steps) and the time steps from 0.0001 to 0.01 a.u. Given that the exact value of the ground-state energy of the hydrogen atom is −0.5 a.u., Fig. 6 shows a pseudocolor plot of the logarithm of the absolute error as a function of grid and time steps. We can see that more accurate results emerge when the grid and time steps are both small. It requires a greater reduction in the time step to get a converged result when reducing the grid step.

In the sample calculation above, we have used a symmetry grid where n_x, n_y, n_z are even and identical. The criterion always involves a big grid size and a huge number of time steps, which decreases the utilities in a realistic problem. Fortunately, things change when we turn to an asymmetry grid where n_x, n_y, n_z are an odd–odd–even combination. (Three odd numbers are not allowed because otherwise the grid will touch the singular point.) We keep $n_x \approx n_y \approx n_z$ in order to insure an approximately unbiased spatial resolution ($\Delta x \approx \Delta y \approx \Delta z$). For example, we can use the grid size $(n-1)\times(n-1)\times n$ ($n = 128, 256, \ldots, 1024$). The absolute error as a function of grid step (at the converged time steps accordingly) is plotted in Fig. 7 with a comparison with the symmetric situation. As is shown, the accuracy of the ground-state energy can be improved by a factor of ten.

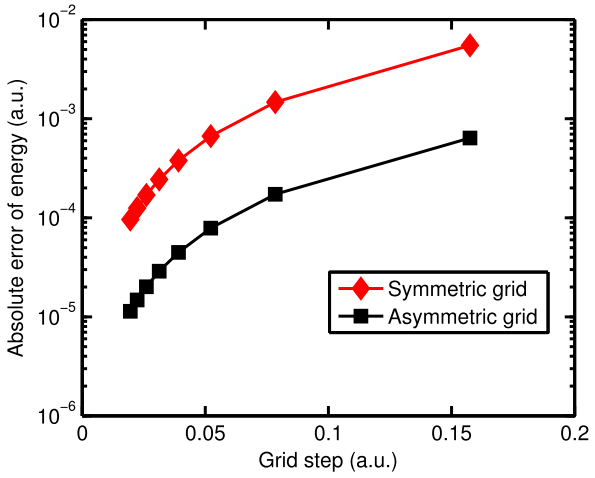

Fig. 7. (Color online) A comparison of the absolute error of the ground-state energy as a function of grid step (at the converged time steps accordingly) using the symmetric grid n^3 (red diamond) and the asymmetric grid $(n-1)\times(n-1)\times n$ (black square), where $n = 128, 256, \ldots, 1024$.

8.2. High harmonic generation

We turn to the second example where the hydrogen atom interacts with a linearly polarized laser pulse. The laser vector potential is chosen to have a sine-square envelope

$$\mathbf{A}(t) = \begin{cases} c\dfrac{E_0}{\omega}\sin^2\left(\dfrac{\pi t}{T_1}\right)\sin(\omega t)\mathbf{e}_z, & 0 \le t \le T_1 \\ 0, \quad \text{otherwise} \end{cases} \tag{27}$$

where E_0 is the electric field amplitude, ω is the angular frequency, $T_1 = 3T$ is the total pulse duration ($T = 2\pi/\omega$ is an optical period, corresponding to 110.32 a.u. for the laser wavelength of 800 nm). The same system was used by Y.C. Han and L.B. Madsen [9] in their calculations of high harmonic generation where the TDSE was solved in spherical coordinates. We present the high harmonic spectra at an intensity of 0.3×10^{14} W/cm^2 ($E_0 = 0.0292$ a.u.), based on three different forms, the dipole, dipole velocity and acceleration forms, which are comparable to the reference.

The ionization dynamics are simulated by real-time propagations. During the propagation, the time-dependent expectation values of the dipole moment, dipole velocity, and dipole acceleration are calculated and saved at each time step. The high harmonic spectra are obtained from Fourier transforms of the former at the end of the propagation.

The input file *Input.in* is prepared in the *./PCTDSE/test/hhg* subdirectory. To run this example, the user can enter the directory and type

mpirun -np N../../bin/driver_hhg Input.in & > hhg.log

The program writes the calculation details to the file *hhg.log*. It also writes the time-dependent expectation values of the dipole moment, dipole velocity, and dipole acceleration to the files *Avg_r.txt*, *Avg_p.txt* and *Avg_a.txt* respectively. The high harmonic spectra based on the three forms are saved into the files *SptrM.txt*, *SptrV.txt* and *SptrA.txt*.

The typical calculation is made using a symmetric grid of size $1536\times1536\times1536$ decomposed on a 12×12 process grid, covering ±125 a.u. in each dimension. The time step is set 0.02 a.u. resulting in 17500 time steps that spends a machine time of 74.5 h. The results are shown in Fig. 8. The time-dependent expectation values

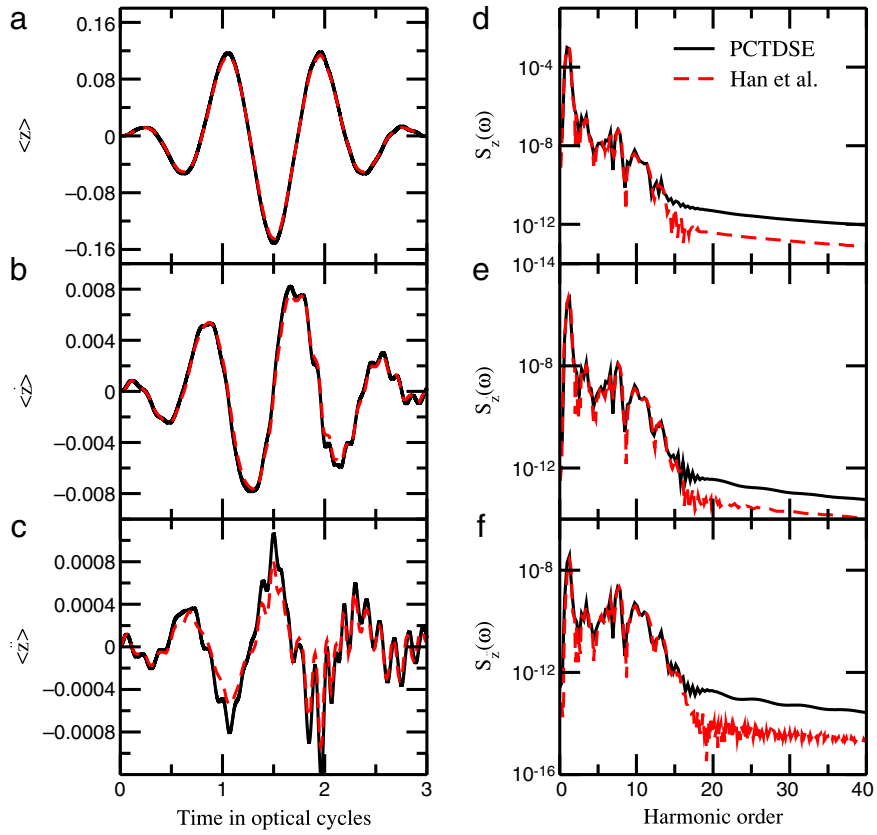

Fig. 8. (Color online) Expectation values of dipole moment (a), dipole velocity (b), and dipole acceleration (c) as a function of time, along with the respective harmonic spectra (d), (e), and (f) versus harmonic order, for a laser of wavelength $\lambda = 800$ nm and an intensity of 0.3×10^{14} W/cm^2. The solid black and dashed red curves refer to the results from our calculation and from Ref. [9].

of the dipole moment, dipole velocity and dipole acceleration are shown in the left panels (a)–(c), and the corresponding harmonic spectra are shown to the right, (d)–(f). We can see that our code can deliver repeatable results to some extent. The biggest difference appears in the curves of the dipole acceleration. This is because the acceleration operator $\ddot{z} = -z/r^3 - E(t)$ has a strong singularity which is very difficult to capture using the uniform grid.

8.3. Electron momentum distribution

For a long time, the timing of the tunneling process has been a controversy. Recent works [33–35] have been dedicated to extract the tunneling time from the angular offset of the maximum in the 2D photoelectron momentum distribution, measured by ionization in strong, nearly circularly polarized fields. In the theoretical aspect, the offset angle can be read from the 2D photoelectron momentum distribution which can be obtained from directly solving the 3D TDSE. Different from the standard method where one need to project the wave function at the end of the laser pulse on the ingoing scattering states [7], we use the trick described in the Section 4.3 to calculate the electron momentum distribution. We have proved the applicability of our code in this aspect.

In this example, a numerical attoclock experiment is carried out using a circularly polarized laser pulse with central wavelength 800 nm. The vector potential being the same as the choice of L. Torlina et al. [36] reads

$$\mathbf{A}(t) = \begin{cases} -A_0 \cos^4(\omega t/4) \\ \quad \times (\cos(\omega t)\mathbf{e}_x + \sin(\omega t)\mathbf{e}_y), \quad -T \le t \le T \\ 0, \quad \text{otherwise} \end{cases} \tag{28}$$

where $A_0 = cE_0/\omega$ is the amplitude of the vector potential. The offset angles are calculated at a serial of laser intensities and a comparison with the results from the reference is presented.

During the real-time propagation, the wave function absorbed by the boundary is accumulated and evolutes under the laser field only. Approximately, it contain all the ionization information at the final time. The photoelectron momentum distribution in the polarization plane can then be obtained by integrating Eq. (26) over the p_z coordinate, i.e., $f(p_x, p_y) = \int f(\mathbf{p}, t_f) dp_z$. The offset angle can be calculated from the average electron momentum $\bar{\mathbf{p}} = (\bar{p}_x, \bar{p}_y, \bar{p}_z) = \int f(\mathbf{p}, t_f)\mathbf{p}d\mathbf{p}$ by $\theta = \tan^{-1}(\bar{p}_y/\bar{p}_x)$.

The input file *Input.in* is prepared in the *./PCTDSE/test/emd* subdirectory which gives a sample calculation at the intensity of 0.8×10^{14} W/cm^2 (circular field $E_0 = 0.0338$ a.u.). To run this example, the user can enter this directory and type

mpirun -np N../../bin/driver_emd Input.in & > emd.log

The program writes the calculation details to the file *emd.log*. The average electron momentum of the absorbed part at all immediate times and the final electron momentum distribution are saved into the files *Avg_p.txt* and *Emd.txt*, respectively.

The typical calculation is made using an asymmetric grid of size $511 \times 511 \times 512$ decomposed on a 8×8 process grid, covering ± 100 a.u. in each dimension. The time step is set 0.02 a.u., resulting in 18 000 time steps that spends a machine time of 17.5 h. The 2D electron momentum distribution is shown in Fig. 9. We can see that there is an angular offset of the maximum in the distribution, as expected.

At last, we give the offset angle as a function of laser intensity in Fig. 10. The results for other intensities are obtained by modifying the input file properly and implementing the calculations repeatedly. A bigger time step of 0.05 a.u. is enough at large intensities.

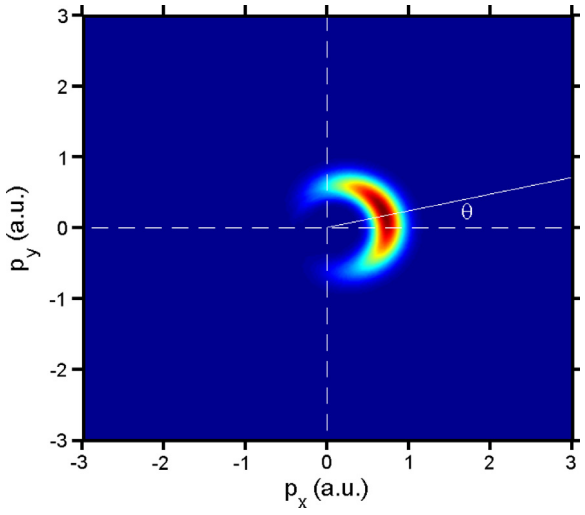

Fig. 9. (Color online) Photoelectron momentum distribution in the polarization (x–y) plane at the laser intensity of 0.8×10^{14} W/cm^2. θ is the offset angle relative to the horizontal p_x direction.

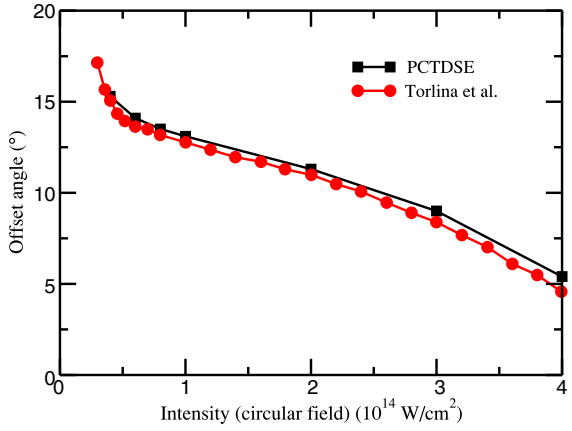

Fig. 10. (Color online) A comparison of the offset angles as a function of laser intensity, obtained from the present code (black square) and Ref. [36] (red circle).

However, It must reduce to 0.01 a.u. for convergence at the lowest intensity. We can see that the two curves agree within 0.5°.

8.4. Electron energy spectrum

We now move on to the last example showing the capabilities of the present code in calculating the photoelectron energy spectrum. The electron energy spectrum is an angle-integrated probability density which can build a bridge with the electron momentum distribution according to the energy–momentum relationship $e = \mathbf{p}^2/2$. We have proved the calculations of the electron momentum distribution in the above subsection, using the trick in the Section 4.3.

We consider the ionization of atomic hydrogen in a 10-cycle laser pulse with a $\sin^2$ envelope for the vector potential

$$\mathbf{A}(t) = \begin{cases} c\dfrac{E_0}{\omega}\sin^2\left(\dfrac{\omega t}{20}\right)\sin(\omega t)\mathbf{e}_z, & 0 \le t \le 20\pi/\omega \\ 0, \quad \text{otherwise.} \end{cases} \tag{29}$$

Three peak intensities 4×10^{14} W/cm^2, 6×10^{14} W/cm^2 and 1×10^{15} W/cm^2 at the wavelength of 780 nm ($\omega = 0.057$ a.u.) are selected in order to compare directly with the results presented by A. N. Grum-Grzhimailo et al. [18].

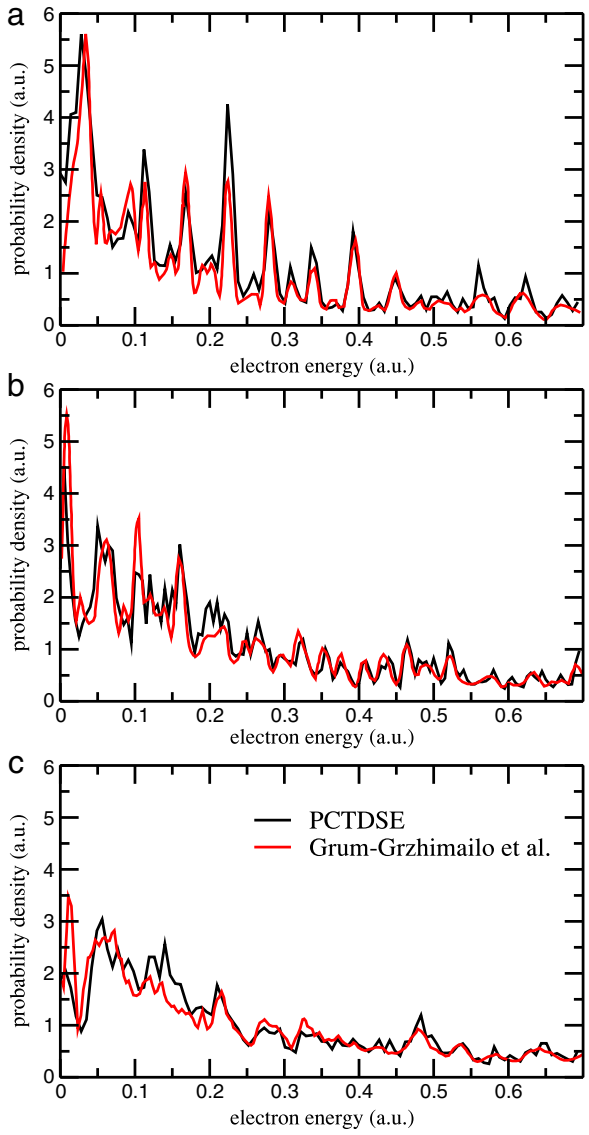

Fig. 11. (Color online) Photoelectron energy spectrum of atomic hydrogen in a laser pulse as Eq. (29) at the three peak intensities of 4×10^{14} W/cm^2 (a), 6×10^{14} W/cm^2 (b) and 1×10^{15} W/cm^2 (c), at the wavelength of 780 nm. There is good agreement between the spectra by A. N. Grum-Grzhimailo et al. [18] (red) and ours (black).

The input file *Input.in* is prepared in the *./PCTDSE/test/ees* subdirectory which gives a sample calculation at the intensity of 1×10^{15} W/cm^2 ($E_0 = 0.1688$ a.u.). To run this example, the user can enter this directory and type

mpirun -np N../../bin/driver_ees Input.in & > ees.log

The program writes the calculation details to the file *ees.log*. The average electron momentum of the absorbed part at all immediate times and the final electron energy spectrum are saved into the files *Avg_p.txt* and *EES.txt*, respectively.

The typical calculation is made using an asymmetric grid of size $675 \times 675 \times 676$ decomposed on a 8×8 process grid, covering ± 160 a.u. in each dimension. The time step is set 0.02 a.u., resulting in 75 000 time steps that spends a machine time of five days. Fig. 11 shows our results for the three peak intensities, each of which gives a comparison with the result from the Ref. [18]. Qualitatively we obtain very good agreement with their predictions. The biggest difference appears in the low-energy regime, especially at the highest intensity of 1×10^{15} W/cm^2. This is because the simulation

box in our calculations is relatively small, as compared with the reference.

9. Conclusions

In this paper we have presented the PCTDSE package for modeling laser–atom interactions. It solves the time-dependent Schrödinger equation on a 3D Cartesian using the split-operator method and a 2D domain decomposition strategy. It is devised relying on the 2DECOMP&FFT library which supports large-scale simulations making use of thousands of cores on major supercomputers. A detailed description of the package is included. Four examples using the hydrogen atom are given to illustrate the applications of the package. With a modification of the atom and laser modules, the package is easily extended to other systems.

Acknowledgments

This work was supported by the National Basic Research Program of China (973 program) under Grant No. 2013CB922203 and National Natural Science Foundation of China under Grant Nos. 11274382, 11274383 and 11674394.

References

[1] M. Protopapas, C.H. Keitel, P.L. Knight, Rep. Progr. Phys. 60 (1997) 389–486.
[2] H.G. Muller, Phys. Rev. A 60 (2) (1999) 1341–1350.
[3] M. Lewenstein, P. Balcou, M.Y. Ivanov, A. L'Huillier, P.B. Corkum, Phys. Rev. A 49 (3) (1994) 2117–2132.
[4] W.B. Chen, Y.D. Huang, C. Meng, J.L. Liu, Z.Y. Zhou, D.W. Zhang, J.M. Yuan, Z.X. Zhao, Phys. Rev. A 92 (2015) 033410.
[5] H.R. Reiss, Phys. Rev. A 42 (3) (1990) 1476–1486.
[6] A.D. Bandrauk, S. Chelkowski, D.J. Diestler, J. Manz, K.J. Yuan, Phys. Rev. A 79 (2009) 023403.
[7] I.A. Ivanov, A.S. Kheifets, Phys. Rev. A 89 (2014) 021402(R).
[8] X.X. Guan, C.J. Noble, O. Zatsarinny, K. Bartschat, B.I. Schneider, Comput. Phys. Comm. 180 (2009) 2401–2409.
[9] Y.C. Han, L.B. Madsen, Phys. Rev. A 81 (2010) 063430.
[10] M. Dondera, H. Bachau, Phys. Rev. A 85 (2012) 013423.
[11] J.M.N. Djiokap, S.X. Hu, L.B. Madsen, N.L. Manakov, A.V. Meremianin, A.F. Starace, Phys. Rev. Lett. 115 (2015) 113004.
[12] K.C. Kulander, Phys. Rev. A 36 (6) (1987) 2726–2738.
[13] G. Onida, L. Reining, A. Rubio, Rev. Modern Phys. 74 (2002) 601–659.
[14] X.M. Tong, C.D. Lin, J. Phys. B: At. Mol. Opt. Phys. 38 (2005) 2593–2600.
[15] M. Førre, A.S. Simonsen, Phys. Rev. A 90 (2014) 053411.
[16] C.O. Broin, L.A.A. Nikolopoulos, Comput. Phys. Comm. 185 (2014) 1791–1807.
[17] S.X. Hu, L.A. Collins, B.I. Schneider, Phys. Rev. A 80 (2009) 023426.
[18] A.N. Grum-Grzhimailo, B. Abeln, K. Bartschat, D. Weflen, Phys. Rev. A 81 (2010) 043408.
[19] D. Bauer, P. Koval, Comput. Phys. Comm. 174 (2006) 396–421.
[20] I.K. Gainullin, M.A. Sonkin, Comput. Phys. Comm. 188 (2015) 68–75.
[21] M.D. Feit, J.A. Fleck, A. Steiger, J. Comput. Phys. 47 (1982) 412–433.
[22] G.R. Mocken, C.H. Keitel, Comput. Phys. Comm. 178 (2008) 868–882.
[23] C.M. Dion, A. Hashemloo, G. Rahali, Comput. Phys. Comm. 185 (2014) 407–414.
[24] URL http://www.mpi-forum.org/.
[25] Intel Math Kernel Library. URL http://software.intel.com/en-us/articles/intel-mkl/.
[26] Engineering and Scientific Subroutine Library. URL http://www-03.ibm.com/systems/software/essl/.
[27] AMD Core Math Library. URL http://developer.amd.com/libraries/acml/pages/default.aspx.
[28] Fastest Fourier Transform in the West. URL http://www.fftw.org/.
[29] K. Burnett, V.C. Reed, J. Cooper, P.L. Knight, Phys. Rev. A 45 (1992) 3347.
[30] J.L. Krause, K.J. Schafer, K.C. Kulander, Phys. Rev. A 45 (7) (1992) 4998.
[31] S.X. Hu, C.H. Keitel, Phys. Rev. A 63 (2001) 053402.
[32] N. Li, S. Laizet, 2DECOMP&FFT—A highly scalable 2D decomposition library and FFT interface, Cray User Group 2010 conference, Edinburgh, 2010. URL http://www.2decomp.org/pdf/17B-CUG2010-paper-Ning_Li.pdf.
[33] P. Eckle, Science 322 (2008) 1525–1529.
[34] P. Eckle, Nat. Phys. 4 (2008) 565–570.
[35] R. Boge, C. Cirelli, A.S. Landsman, S. Heuser, A. Ludwig, J. Maurer, M. Weger, L. Gallmann, U. Keller, Phys. Rev. Lett. 111 (2013) 103003.
[36] L. Torlina, F. Morales, J. Kaushal, I. Ivanov, A. Kheifets, A. Zielinski, A. Scrinzi, H.G. Muller, S. Sukiasyan, M. Ivanov, O. Smirnova, Nat. Phys. 11 (2015) 503–508.