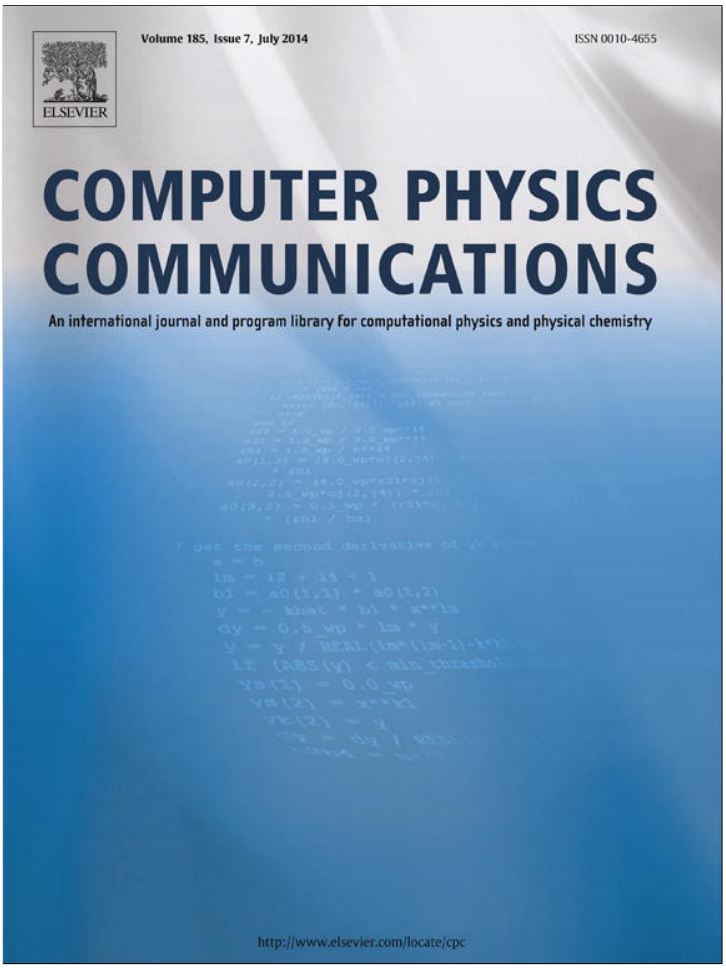

Computer Physics Communications 185 (2014) 1955–1964

Contents lists available at ScienceDirect

Computer Physics Communications

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

Accelerating spectral atomic and molecular collisions methods with graphics processing units

F.D. Colavecchia

División Física Atómica, Molecular y Óptica, Centro Atómico Bariloche, and CONICET, 8400 S. C. de Bariloche, Río Negro, Argentina

ARTICLE INFO

Article history:
Received 10 January 2014
Received in revised form
14 March 2014
Accepted 26 March 2014
Available online 3 April 2014

Keywords:
Ionization
Spectral methods
Sturmian functions
GPU computing

ABSTRACT

We present a computation method to accelerate the calculation of the Hamiltonian of a three-body time independent Schrödinger equation for collisions. The Hamiltonian is constructed with one dimensional (basis overlaps) and two dimensional (interparticle interaction) integrals that are mapped into a computational grid in a Graphics Processing Unit (GPU). We illustrate the method for the case of an electron impact single ionization of a two electron atom. This proposal makes use of a Generalized Sturmian Basis set for each electron, which are obtained numerically on a quadrature grid that is used to compute the integrals in the GPU. The optimal computation is more than twenty times faster in the GPU than the calculation in CPU. The method can be easily scaled to computers with several Graphics Processing Units or clusters.

1. Getting started

The properties of atoms and molecules can be determined by their wave function, which is obtained as the solution of the Schrödinger equation for non-relativistic energies. There are many methods to solve this second order, partial differential equation, from simple approximations that can give a hint of the physics, to the complete ab-initio numerical solution to predict physical quantities with a high degree of precision. When the wave function of an atomic or molecular system is written in a basis expansion, the Schrödinger equation transforms into a linear problem described by a dense matrix [1]. To obtain this matrix, it is necessary to compute integrals between two elements of the basis (called overlaps) and integrals of the interparticle potentials, that usually involve four basis functions.

The details of the calculation of each element of this matrix (called the Hamiltonian matrix) greatly depend on two factors. First, the system of coordinates chosen to represent the positions of the particles and their interactions, and second, the election of the basis set for each particle. These choices are determined not only by the number or type of particles involved, but also for the kind of experiment and/or physical property under investigation. For example, the simpler method makes use of Slater Type Orbitals (exponentials times powers of the electronic coordinates relative to the nuclei) as a basis set [2]. This is well suited to study bound atomic systems, but not to analyze problems where particles can be spread out through the space, like ionization processes. Quantum chemistry calculations usually employ Gaussian functions to obtain properties of molecules, and take great advantage from the fact the systems under scrutiny are bound states [3–5].

Unlike these calculations of the quantum chemistry arena, collisional problems deal with particles that can be far from each other. In fact, the most important feature of a collision, the cross sections, are defined in those regions, assuming that the interactions among the particles are no longer effective and that they are far away from where the collision took place [6]. Therefore, the basis in collision problems should be able to accurately expand the full wave function for large interparticle distances. Moreover, if charged particles are present, the basis should take into account the long-range asymptotic behavior of Coulomb fields [7,8]. Finally, possibly the main shortcoming is that it is not possible to use simple analytic basis sets for collision problems. All these factors pose several challenges to perform a numerically accurate calculation of wave functions and cross sections in atomic and molecular collisions [9].

In the last few years, the computer hardware and software have been moving fast to an heterogeneous world [10]. In the desktop market, this meant going from a simple one-core desktop computer to an aggregate of one to several multicore CPUs with their corresponding accelerators. This also replicates in High Performance Computing clusters, such as the Titan supercomputer at Oak Ridge National Laboratory [11]. Nowadays it is not possible to program a scientific code thinking only about the pure speed of the calculation (i.e., floating points operations per second). In-core and out-of-core communication layers, memory hierarchies,

E-mail address: flavioc@cab.cnea.gov.ar.

http://dx.doi.org/10.1016/j.cpc.2014.03.026
0010-4655/

latency factors have to be taken into account at the time of modeling a numerical calculation. One major advantage within this picture is that all this heterogeneity is exposed to the programmer by software abstractions. Within the Graphical Processing Units accelerators, Computer Unified Device Architecture (CUDA) has been developed as a massively parallel programming tool, and is currently considered a de-facto standard. CUDA is a C language extension that enables one to program some GPUs for scientific purposes [12,13]. Besides, the OpenCL [14] initiative strives to provide an open framework to work not only with NVIDIA GPUs, but also with any aggregate of cores and accelerators from any hardware provider. It is still unclear whether any of these architectures will prevail, or if they will converge into an broader standard.

The aim of this work is to demonstrate that a careful analysis of the structure of the Hamiltonian leads to an efficient mapping of the calculations into the software abstractions of GPUs, resulting in codes that can run more than twenty times faster than their CPU's counterparts. The paper is organized as follows. In Section 2 we review the structure of the simplest three body collisional system, making use of a spectral method based on Generalized Sturmian Functions [15]. In Section 3 we detail the procedure to compute the Hamiltonian in this basis, and analyze the calculations in GPU. In Section 4 we summarize our results, and envision further enhancements that can be subject of future research. Atomic units are used thorough the paper.

2. The structure of the three-body Hamiltonian

There is a vast amount of different collisions and reactions that involve atoms and molecules. However, the simplest system, yet still an active area of research, is the Three Body problem: one particle collides with a two-body bound target, resulting in the excitation of the target, the capture of one of the particles of the target by the projectile, or even the dissociation of it, leading to three free particles after the collision. Single ionization, excitation or capture of atoms by electrons or ions, reactive scattering or recombination chemical reactions are examples of Three Body problems. If the particles of the system are charged, the long range of Coulomb interactions should be taken into account [6].

Let us consider an atomic system of two electrons and a heavy atomic nucleus. This is a simple atomic system, but showcases most of the difficulties associated with the collisional problem. Moreover, if one is interested in the double ionization of a multielectronic atom, one can always reduce the problem to the three-body one, including the screening effect of the non-active electrons in a model potential. In any case, we can assume that the nucleus is at rest, and refer the motion of the electron relative to it. The interactions between the particles are Coulomb ones, hence one can write the electronic Hamiltonian as:

$$H = \nabla^2_{\mathbf{r}_1} + \nabla^2_{\mathbf{r}_2} + V(r_1) + V(r_2) + V(r_{12}) \tag{1}$$

which is simply the sum of the kinetic and potential energies of the system. The coordinates set $\{\mathbf{r}_1, \mathbf{r}_2\}$ determines the positions of the electrons 1 and 2 in space, relative to the fixed nucleus, while $r_{12} = |\mathbf{r}_1 - \mathbf{r}_2|$ is the interelectronic distance. The interactions are Coulomb potentials given by ($i = 1, 2$)

$$V(r_i) = -\frac{Z}{r_i} \quad \text{and} \quad V(r_{12}) = \frac{1}{r_{12}},$$

where Z is the charge of the nucleus. Solutions to the time-dependent Schrödinger equation

$$H\Phi = i\frac{\partial \Phi}{\partial t} \tag{2}$$

determine all the quantum mechanical properties of the system. Different solutions to this equation depend on the boundary conditions set in the coordinate domain. There are basically two different classes of solutions related to this equation: bound states, that decay exponentially for large distances, and continuum states, that behave as waves when particles are far from each other.

Bound states are the eigenvectors of the Hamiltonian. Replacing $\Phi = \Psi \exp(-iEt)$ in (2), one ends up with the eigenvalue problem

$$(H - E)\,\Psi = 0 \tag{3}$$

where E is the (unknown) energy of state Ψ.

Continuum states result from collisions, therefore, one can assume that the complete state of the system is the sum of the initial state Ψ_0 and the scattered part Ψ:

$$\Phi = \Psi_0 e^{-iEt} + \Psi e^{-iEt} \tag{4}$$

and can be obtained from the solution of the time-independent problem

$$(H - E)\,\Psi = -(H - E)\,\Psi_0 \tag{5}$$

where Ψ_0 is the initial unperturbed state of the process, and E is the total energy of the system. One can split H such that

$$H = H_0 + W \quad \text{with } (H_0 - E)\Psi_0 = 0$$

because the wave function Ψ_0 is an eigenvalue of the unperturbed system represented by the Hamiltonian H_0. Therefore, we have

$$(H - E)\,\Psi = W\Psi_0. \tag{6}$$

The, possibly differential, operator W is responsible for the transitions from the initial, known state Ψ_0 to the continuum, collisional state Ψ. The initial state in a ionization collision is described by the product of the bound state of the target atom and a plane wave describing the impinging projectile. The boundary conditions for the unknown scattering state Ψ are very complex [8,16]: it is customary to split the asymptotic regions in Ω_0, where all the particles are far from each other, and Ω_i, $i = 1, 2, 3$, where particle i is far from the rest of the system. When one deals with ionization problems, the most important of these regions is Ω_0. This boundary condition can be written as an hyperspherical wave [16]:

$$\Psi(\mathbf{r}_1, \mathbf{r}_2) \underset{\rho\to\infty}{\to} \frac{1}{(2\pi)^{5/2}} \frac{K^{3/2}}{\rho^{5/2}} e^{iK\rho + i\lambda_0 \ln(2K\rho) + i\pi/4} \tag{7}$$

where the hyper-radius is $\rho = \sqrt{r_1^2 + r_2^2}$ while $K = \sqrt{k_1^2 + k_2^2}$ is the hyper-momentum and λ_0 is a Coulomb parameter involving charges and relative momenta of the particles.

In this work, we will compute the time-independent wave function. The solution to either (3) or (6) can be computed analytically only in very few cases, and the Hamiltonian (1) has to be represented and solved numerically. One possible numerical path is to describe the Schrödinger equation in a numerical grid, using finite differences or finite elements methods [17,18]. The representation of the Hamiltonian in these methods is quite straightforward, but the mesh in both cases results in huge, sparse matrices, that can be difficult to handle unless a powerful computer cluster is at hand. Besides, scaling to more particles is cumbersome, and management of boundary conditions, specially in the continuum case, is tricky [19].

Another way to tackle this problem numerically is to use a spectral method, where one introduces a suitable basis set to represent the solution Ψ. The resulting Hamiltonian matrix is usually much smaller than direct methods, and one has the benefit that can introduce physical features directly in the basis elements [20–22].

For the atomic system in consideration, a spectral method in the electronic coordinates introduces a basis set $\{\Xi_{nm}(\mathbf{r}_1, \mathbf{r}_2)\}$ to expand the wave function Ψ as

$$\Psi(\mathbf{r}_1, \mathbf{r}_2) = \sum_{nm} a_{nm} \Xi_{nm}(\mathbf{r}_1, \mathbf{r}_2)$$

such that the Hamiltonian matrix is defined as

$$\mathbf{H}_{n'm'nm} = \langle \Xi_{n'm'} | H - E | \Xi_{nm} \rangle ,$$

which is a square matrix of $N_H \times N_H$ elements. Therefore, the solution of the linear system $\mathbf{Ha} = \mathbf{b}$ gives the set of coefficients $\mathbf{a} = \{a_{nm}\}$ that determines the solution Ψ of the Schrödinger equation (6). The matrix $\mathbf{b}$ is the projection into the basis of the action of operator W on the initial state Ψ_0.

The elements of the Hamiltonian matrix $\mathbf{H}_{n'm'nm}$ are a priori six dimensional integrals in the whole space spanned by $\{\mathbf{r}_1, \mathbf{r}_2\}$. However, they can be reduced if physical symmetries are taken into account. Up to this point no assumption about the angular symmetries has been included in the basis, and the coupling between the angular momenta of each particle can be introduced in many different ways. Let us assume that we make use of spherical coordinates for each electron: $\mathbf{r}_i = (r_i, \theta_i, \varphi_i)$ for $i = 1, 2$. The elements of the basis Ξ_{nm} include the product of two one-electron functions depending on the distance to the nucleus times a function that couples the angular variables [23]

$$\Xi_{nm}(\mathbf{r}_1, \mathbf{r}_2) = \frac{S_{n_a l_a}(r_1)}{r_1} \frac{S_{n_b l_b}(r_2)}{r_2} \mathcal{Y}^{LM}_{l_a l_b}(\widehat{\mathbf{r}}_1, \widehat{\mathbf{r}}_2).$$

The elements S_{nl} are Generalized Sturmian Functions (GSF), eigenvectors of a one dimensional radial Schrödinger equation with angular momentum l. That is, for $i = 1, 2$ we define the one dimensional Hamiltonian

$$h_i = -\frac{1}{2}\frac{d^2}{dr_i^2} + \frac{l(l+1)}{2r_i^2} + V(r_i) - E_i \tag{8}$$

such that

$$h_i S_{nl}(r_i) = -\beta_{nl} \mathcal{V}(r_i) S_{nl}(r_i), \tag{9}$$

where β_{nl} are the eigenvalues of the problem. The functions S_{nl} are orthonormal respect to the *generating* potential $\mathcal{V}$ [24]. This is a short range, physically sound potential, chosen according to the problem under scrutiny: for bound state calculations, one chooses a Yukawa potential; for scattering problems, a square well potential is usually the best choice. The generating potential must vanish for distances larger than a certain value R_{max}, that also depends on the features of the physical problem. The eigenvalues β_{nl} of the Sturmian equation (9) and can be complex numbers for outgoing or incoming boundary conditions and real and positive energy E_i, which are the conditions for ionization problems. The energy of each electron E_i can be selected arbitrarily, although we have found that best convergence for ionization problems is achieved when $E_a = E_b = E$ [23].

The coupling among the angular variables is described by the bipolar spherical harmonic $\mathcal{Y}^{LM}_{l_a l_b}(\widehat{\mathbf{r}}_1, \widehat{\mathbf{r}}_2)$ defined as:

$$\mathcal{Y}^{LM}_{l_a l_b}(\widehat{\mathbf{r}}_1, \widehat{\mathbf{r}}_2) = \sum_{m_a, m_b} \langle l_a, l_b; m_a, m_b | L, M \rangle \, Y_{l_a, m_a}(\theta_1, \varphi_1) Y_{l_b, m_b}(\theta_2, \varphi_2)$$

in terms of Clebsch–Gordan coefficients and the usual spherical harmonics [25]. We end up with the following spectral expansion:

$$\Psi(\mathbf{r}_1, \mathbf{r}_2) = \sum_{n_a, n_b, l_a, l_b} a_{l_a l_b n_a n_b} \frac{S_{n_a l_a}(r_1)}{r_1} \frac{S_{n_b l_b}(r_2)}{r_2} \mathcal{Y}^{LM}_{l_a l_b}(\widehat{\mathbf{r}}_1, \widehat{\mathbf{r}}_2). \tag{10}$$

There some subtleties about the lasts equations that should be addressed. First, indexes $\{n, m\}$ that label the basis set become composite ones $\{n_a, l_a, n_b, l_b\}$. This has some implications in the order of the elements of the Hamiltonian matrix according to them, that, as we will show in the next sections, can impact in the numerical performance of the code.

Second, all these angular functions have a defined angular momenta L and projection into a fixed axis M. This does not affect the computation of a bound state, since they are eigenstates with well defined angular momentum and projection. However, this could be not the case in a collision process. It could be possible to have a very general transition operator W which mixes states of different angular momenta. Therefore, summation over the indexes $\{n, m\}$ of the general Hamiltonian should be also extended to include different L and M. Throughout this paper we will assume that the perturbation W does not mix angular momenta, and one can solve the Schrödinger equation for each pair $\{L, M\}$ independently.

Finally, it is important to note that the particular selection of the radial functions S_{nl} only determines the specific expression of the Hamiltonian matrix in terms of the basis set, eigenvalues β_{nl} and potentials V and $\mathcal{V}$. Hence, the method presented here is completely general and will be suited for any basis set chosen.

2.1. Hamiltonian blocks

At this point, it is clear that for each pair $\{L, M\}$ one needs to define how many elements are going to be included in the basis set. Let us assume that we include N^{LM} pairs $\{L, M\}$. For each fixed value of (L and M), one choses a set of P^{LM} pairs

$$(l_a, l_b)^{LM} = \left\{ (l_a, l_b)_1, (l_a, l_b)_2, (l_a, l_b)_3, \ldots, (l_a, l_b)_{P^{LM}} \right\}.$$

Even though the number P^{LM} of pairs is not known a priori, it can be determined by convergence properties on the wave function, and is different for each process under consideration. Besides, for each pair (l_a, l_b), one defines also the size of the radial basis set for each electron. For simplicity, we assume that the basis size is the same for all pairs $(l_a, l_b)_{LM}$, that is to say, the basis size depends only on L and M. We will define N_a^{LM} (N_b^{LM}) as the number of functions $S_{n_a l_a}$ ($S_{n_b l_b}$) included in the basis for each electron with angular momentum l_a (l_b), respectively. With all these definitions, one can compute the size $N_H \times N_H$ of the Hamiltonian Matrix with:

$$N_H = \prod_j^{N^{LM}} P^j \left(N_a^j \times N_b^j \right),$$

where the index j runs across all the possible values of LM included in the problem. This simply resembles the block structure of the Hamiltonian matrix that has $N_{L'M'} \times N_{LM}$ sub-matrices $\mathbf{H}^{L'M'LM}$, which are also divided in $P^{LM} \times P^{LM}$ blocks $\mathbf{h}^{l'_a l'_b l_a l_b}$ of size $\left(N_a^j \times N_b^j \right)^2$ each. Fortunately, in many interesting cases the hypothesis that the perturbation does not mix the angular momenta holds, and one has to deal with the diagonal blocks of the Hamiltonian $\mathbf{H} = \mathbf{H}^{L'=L, M'=M}$ independently.

To fix the ideas, let us show a few of examples. First, let us consider an hypothetical process for which the Hamiltonian needs to be computed for $L = 0, 1, 2$, and $M = 0$ for all L (see Fig. 1). Therefore $N_{L'=2,M'=0} = N_{L=2,M=0} = 3$ and the Hamiltonian contains nine blocks sub-matrices $\mathbf{H}^{L'0L0}$. Let us suppose that the perturbation is such that only the diagonal blocks of the Hamiltonian matrix are needed. Hence, we need to compute three sub-matrices $\mathbf{H}^{0000}$, $\mathbf{H}^{1010}$ and $\mathbf{H}^{2020}$. Fixing the number of (l_a, l_b) pairs to $P^{LM} = 4$ for all L, and taking into account selection rules, we would have the pairs[1]

$$(l_a, l_b)^{00} = \{(0, 0), (1, 1), (2, 2), (3, 3)\},$$

[1] The selection of which pairs (l_a, l_b) are included in the set is given by convergence tests, and usually unknown a priori.

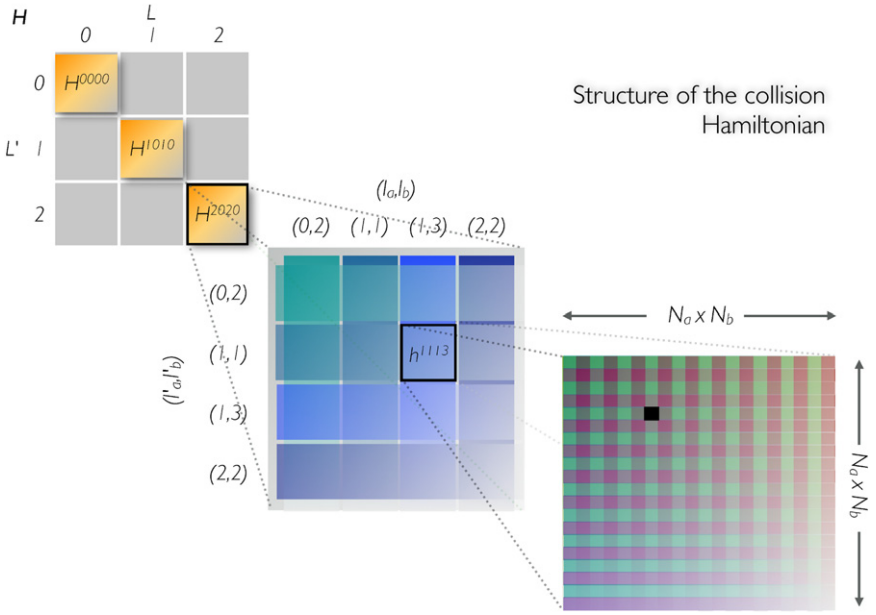

Fig. 1. Structure of the Hamiltonian matrix. Here we show a Hamiltonian matrix for $L = 0, 1, 2$, $M = 0$, that is to say $N_{L'=2,M'=0} = N_{L=2,M=0} = 3$. Each of the $\mathbf{H}^{L'0L0}$ blocks contains 16 sub-blocks $\mathbf{h}^{l'_a l'_b l_a l_b}$. We illustrate the case of the $\mathbf{H}^{22}$ block having a grid of $P^{20} \times P^{20}$ sub blocks, with $P^{20} = 4$. Finally, we show the $\mathbf{h}^{1113}$ tile, which has $(N_a N_b)^2 = (4 \times 5)^2 = 400$ complex numbers. Each one of these values (black square) is obtained from a sum of two two-dimensional integrals, and six one dimensional ones, see Eq. (17).

$$(l_a, l_b)^{10} = \{(0, 1), (1, 2), (2, 3), (3, 4)\},$$
$$(l_a, l_b)^{20} = \{(0, 2), (1, 1), (1, 3), (2, 2)\}.$$

Each of the three diagonal sub-matrices will have $P^{L0} \times P^{L0} = 4 \times 4 = 16$ sub blocks $\mathbf{h}^{l'_a l'_b l_a l_b}$. A detailed representation for $\mathbf{H}^{2020}$ is depicted in the center of Fig. 1. If we set, for example, $N_a = 4$ and $N_b = 5$, each block $\mathbf{h}^{l'_a l'_b l_a l_b}$ will have $(N_a N_b)^2 = (4\ 5)^2 = 20^2 = 400$ complex numbers. Therefore, the total memory required to keep the Hamiltonian will be $3 \times 16 \times 400 \times 16$ bytes, which is about 300 kB.

Second, if one is interested in the calculation of the fundamental state of He, one has to solve Eq. (3) for $L = 0$ and $M = 0$. Due to selection rules, $l_a = l_b$. In Ref. [26] we make use of thirteen pairs of (l_a, l_b), therefore $P^{00} = 13$. Besides, we employed 35 functions for each electron, that is to say, $N_a^{00} = N_b^{00} = 35$. The size of each $\mathbf{h}^{l'_a l'_b l_a l_b}$ is a matrix of $(35 \times 35) = 1225$ rows and columns, with $1225^2 = 1500625$ complex numbers, which occupy 23.45 MB. Since the complete Hamiltonian has a total of 13×13 of these blocks, 3.96 GB of memory storage are needed for it, which can be accommodated with ease in nowadays computers.

Finally, the calculation of a collision process is a different story. Let us take for example the high energy (e, 3e) calculation. We have shown that this process can be modeled as a three body one, thus the Hamiltonian is the one given by Eq. (1) [27]. However, the exact wave function contains all angular momenta L and should be converged respect to it. Taking into account $L = 0, 1, \ldots, 4$, (that is, $N^{LM} = 5$) and five pairs of (l_a, l_b) for each L, we have $P^{LM} = 5$. We employ 84 functions for each electron, $N_a^{LM} = N_b^{LM} = 84$. Each $\mathbf{h}^{l'_a l'_b l_a l_b}$ is a matrix of $(84 \times 84) = 7056$ rows and columns, with $7056^2 = 49,79 \times 10^6$ complex numbers, occupying 0.78 GB. The perturbation does not mix angular momenta, and therefore the Hamiltonian has a diagonal structure with $N^{LM} = 5$ blocks, each one containing $(P^{LM} \times P^{LM}) = 25$ $\mathbf{h}^{l'_a l'_b l_a l_b}$ blocks, or 97.5 GB for the total Hamiltonian.

From now on, we will assume that L and M are fixed, and drop the superindex from the notation. Therefore, we need to find the solution $\mathbf{x}$ of the linear system $\mathbf{Hx} = \mathbf{b}$, where $\mathbf{H} = \mathbf{H}^{L'=L,M'=M}$. The block structure of this matrix corresponds to the distribution of different pairs of one-particle angular momenta defined by $\mathbf{h}^{l'_a l'_b l_a l_b}$ (see Fig. 1).

2.2. Integrals in the Hamiltonian

The specific structure of the Hamiltonian is determined by the coordinate system, and by the basis functions selected in the description of the atomic process. Each choice would lead to integrals of different dimensions, on particular domains defined by the relations among coordinates. In the case of the Generalized Sturmian Functions method, application of Hamiltonian (1) to the wave function defined by (10), and using Eq. (8) to remove the partial derivatives, results in one-dimensional and two-dimensional integral of the basis functions sets. If the one electron radial energies are E_a and E_b for each electron, the Hamiltonian matrix $\mathbf{H}$ has elements

$$[\mathbf{H}^{l'_a l'_b l_a l_b}]_{n'_a n'_b n_a n_b} = \left\langle \mathcal{Y}^{LM}_{l'_a l'_b} \frac{S_{n'_a l'_a}}{r_1} \frac{S_{n'_b l'_b}}{r_2} \middle| H - E \middle| \frac{S_{n_a l_a}}{r_1} \frac{S_{n_b l_b}}{r_2} \mathcal{Y}^{LM}_{l_a l_b} \right\rangle \tag{11}$$

that can be computed after some algebra as

$$\begin{aligned}[\mathbf{H}^{l'_a l'_b l_a l_b}]_{n'_a n'_b n_a n_b} = &\Big[-\beta_{n_a l_a} \mathcal{V}_{a\, n'_a n_a}^{l_a l_a} O^{l_b l_b}_{n'_b n_b} - \beta_{n_b l_b} \mathcal{V}_{b\, n_b n_b}^{l_b l_b} O^{l_a l_a}_{n_a n_a} \\ &+ (E_a + E_b - E)\, O^{l_a l_a}_{n'_a n_a} O^{l_b l_b}_{n'_b n_b} \Big] \delta_{l_a l_a} \delta_{l_b l_b} \\ &+ \sum_{l=0}^{\infty} \frac{4\pi}{2l+1} R^{l'_a l'_b l_a l_b l}_{n'_a n'_b n_a n_b} A^{LM}_{l'_a l'_b l_a l_b l},\end{aligned} \tag{12}$$

in terms of one-dimensional

$$O^{l'l}_{n'n} = \int_0^\infty dr S_{n'l'}(r) S_{nl}(r) \tag{13}$$

$$\mathcal{V}^{l'l}_{i\,n'n} = \int_0^\infty dr S_{n'l'}(r) \mathcal{V}_i(r) S_{nl}(r) \tag{14}$$

and two-dimensional

$$R^{l'_a l'_b l_a l_b l}_{n'_a n'_b n_a n_b} = \int_0^\infty dr_1 \int_0^\infty dr_2 S_{n'_a l'_a}(r_1) S_{n'_b l'_b}(r_2) \frac{r_<^l}{r_>^{l+1}} S_{n_a l_a}(r_1) S_{n_b l_b}(r_2) \tag{15}$$

$$A^{LM}_{l'_a l'_b l_a l_b l} = \sum_{m=-l}^{l} \int d\hat{\mathbf{r}}_1 \int d\hat{\mathbf{r}}_2 \times \mathcal{Y}^{LM*}_{l'_a l'_b}(\hat{\mathbf{r}}_1, \hat{\mathbf{r}}_2) Y^*_{lm}(\hat{\mathbf{r}}_1) Y_{lm}(\hat{\mathbf{r}}_2) \mathcal{Y}^{LM}_{l_a l_b}(\hat{\mathbf{r}}_1, \hat{\mathbf{r}}_2) \tag{16}$$

integrals of the basis functions. The generating potentials for each electron are $\mathcal{V}_a$ and $\mathcal{V}_b$, see the RHS of Eq. (9). The matrix $\mathbf{H}$ involves the electron–electron interaction that was evaluated using the standard spherical harmonics expansion

$$\frac{1}{r_{12}} = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \frac{4\pi}{2l+1} \frac{r_<^l}{r_>^{l+1}} Y^*_{lm}(\theta_1, \phi_1) Y_{lm}(\theta_2, \phi_2), \tag{17}$$

with $r_< = \min(r_1, r_2)$ and $r_> = \max(r_1, r_2)$. Angular integrals $A^{LM}_{l'_a l'_b l_a l_b l}$ can be computed in terms of Clebsch–Gordan coefficients:

$$\begin{aligned} A^{L,M}_{l'_a, l'_b, l_a, l_b l} &= \langle l'_a, l; 0, 0 | l_a, 0\rangle \langle l'_b, l; 0, 0 | l_b, 0\rangle \\ &\times \sqrt{\frac{(2l'_a+1)(2l'_b+1)}{(2l_a+1)(2l_b+1)}} \\ &\times \sum_{\substack{m_a=\max[-l_a, M-l_b] \\ m'_a=\max[-l'_a, M-l'_b]}}^{\substack{\min[l_a, M+l_b] \\ \min[l'_a, M+l'_b]}} \langle l'_a, l'_b; m'_a, M-m'_a | L, M\rangle \\ &\times \langle l_a, l_b; m_a, M-m_a | L, M\rangle \\ &\times \sum_{m=-l}^{l} (-1)^m \langle l'_a, l; -m'_a, m | l_a, -m_a\rangle \\ &\times \langle l'_b, l; m'_a - M, -m | l_b, m_a - M\rangle. \end{aligned} \tag{18}$$

Only one angular integral is required for each set of values $\{L, M, l'_a, l'_b, l_a, l_b, l\}$, and they do not depend on the index labels $\{n'_a, n'_b, n_a, n_b\}$. Therefore, for each block of repulsion integrals, one has to compute only one angular integral. Besides, they are sums of Clebsch–Gordan coefficients times normalization factors, which can be precomputed easily and stored in a table or file. Since the number of these integrals is small, the effect in the overall run-time calculation is negligible.

Note that Eq. (11) is the direct term of the Hamiltonian, and exchange $a \leftrightarrow b$ should be considered to take into account the symmetry of the system if needed.

It should be clear by now that there are two stages in the calculation of the solution Ψ of the problem, provided that one has a numerical representation of the radial basis set S_{nl} [28]. First, one needs to compute the Hamiltonian of the problem, Eq. (11). Second, one has to solve a linear algebra problem: an eigenvalue calculation for bound states, or a linear system for continuum ones. This last step is straightforward, and can be performed with different computational libraries, such as LaPACK [29], MAGMA or PLASMA [30], FLAME [31] etc. in one desktop computer. It is important to note that the only constrain at this point is that the complete matrix should be in the memory of the machine to make use of one of these advanced numerical linear algebra methods. If this is not the case, one can make use of parallel MPI implementations of these packages, and solve the second stage in a computer cluster.

This work is devoted to the optimization of the calculation of the matrix elements of the Hamiltonian in a GPU. Since we make use of CUDA capable cards, we will summarize some aspects of the CUDA parallel abstraction, further details can be found elsewhere [32]. The parallel work in CUDA is organized in kernels, which are execution units that runs on a grid of blocks of execution threads. These kernels are functions written in an extended C Language code, that are started directly in the GPU by the card scheduler. Kernels can take advantage of the memory hierarchy in the GPU, that has a small number of registers, shared memory (a small user-managed cache) per block and global memory (up to 6 GB in Tesla series cards). The grid of blocks and threads is defined by the code at runtime, and can be organized as a one, two or three-dimensional array of blocks, each one composed by one, two- or three-dimensional array of threads. The maximum number of threads per block is 1024, however, the number of blocks available is huge (up to 65 535 per dimension of a 2D grid, for compute capability 2.0). Therefore, the mapping of the calculation into the grid of blocks and arrays is critical in the optimization of the code.

3. Computing the Hamiltonian

The calculation of the Hamiltonian requires the computation of one- and two-dimensional integrals. There are few aspects related with the specific basis functions that concerns both of them. First, all integrals range from zero to infinity, however, the basis set is defined up to the maximum radial value R_{max}. Therefore, we will replace the upper integral limit by this value, and test whether convergences is achieved or not. This is specially important for overlap integrals (13) and (14), since they are generally related to the orthogonality of the basis set. In the case of Generalized Sturmian Functions, orthogonality is in fact given by (14), which provides a strict test for the calculation of these integrals. Moreover, since GSF have all the same correct asymptotic condition

$$S^{asympt}_{nl}(r) = \lim_{r\to\infty} S_{nl}(r) \propto e^{ikr - i\frac{Z}{k}\ln(2kr)},$$

we can easily compute these integral between 0 and R_{max} numerically, and add the asymptotic part of the integral between R_{max} and infinity that can be computed analytically. We have shown that adding the asymptotic integral accelerates the convergence at the wave function level [33].

Second, we will assume that the numerical method can provide accurate basis functions in an arbitrary radial grid. This is the case of Generalized Sturmian Functions, that can be computed directly in a very dense grid, and interpolated in a smaller grid, such as Gaussian quadratures abscissas.

From a computational point of view, it is evident that the two-dimensional integrals are the most time-consuming part of the whole calculation, since they involve four GSF compared to the two GSF in the overlap integrals (13) and (14). Besides, there is an explicit sum in l that is associated with the interelectronic repulsion expansion (17). This sum runs up to the maximum value of the pairs (l_a, l_b) admitted by selection rules, for a given L. Since mapping of integrals to the CUDA abstraction is the same for all angular integrals, we will restrict our attention to the particular case of a fixed value of l in Eq. (17). This example will provide all the features of the optimized calculation in the GPU.

3.1. Overlap integrals

Computation of overlap integrals can be performed in two stages. First we compute the one electron integrals, and then we compute the products of them present in the first three terms of the Hamiltonian, Eq. (12).

Since GSF can be obtained accurately in a numeric grid, we will compute one-dimensional integrals as a sum over Gauss–Legendre

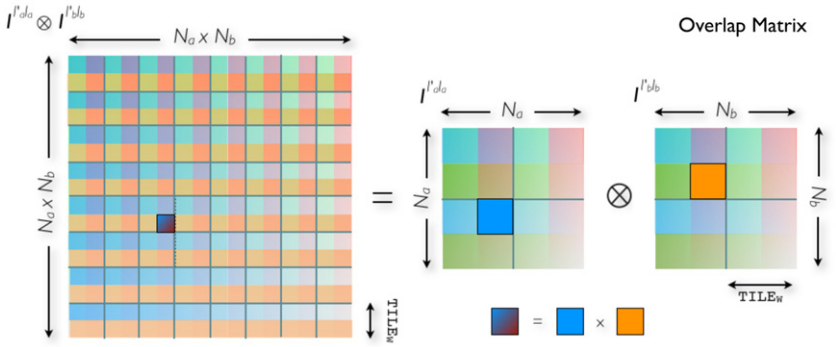

Fig. 2. An illustration of the overlap matrices Eq. (20). The full overlap matrix is constructed as a Kronecker product of one-electron overlaps. In this Illustration, assume that $N_a = N_b = 4$, and $\mathtt{TILE}_W = 2$ the kernel that computes one-electron overlaps $\mathbf{I}^{l'_a l_a}$ and $\mathbf{I}^{l'_b l_b}$ make use of a grid of $(N_i/\mathtt{TILE}_W \times N_i/\mathtt{TILE}_W = 2 \times 2)$ blocks. Once these matrices are obtained in the GPU, the kernel that computes the Kronecker product is run in a grid of (8×8) blocks of 2×2 threads per block. For example, the element $(10, 6)$ of the two-electron overlap (dark blue) is the product of $I^{l'_a l_a}_{32}$ and $I^{l'_b l_b}_{22}$ is computed by thread $(1, 1)$ of block $(4, 2)$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

N_q quadrature points $\{r_j\}$ with weights $\{w_j\}$, with $j = 1, \ldots, N_q$

$$I^{l'l}_{n'n}[f] = \int_0^{R_{max}} S_{n'l'}(r) f(r) S_{nl}(r) dr = \sum_{j=1}^{N_q} S_{n'l'}(r_j) f(r_j) S_{nl}(r_j) w_j. \quad (19)$$

In this expression the basis sets are computed at Gaussian points $\{r_j\}$, and the functional $I^{l'l}_{n'n}$ is equal to the overlaps for $f(r) = 1$, $(I^{l'l}_{n'n}[1] = O^{l'l}_{n'n})$ or to the orthogonality relation for $f(r) = \mathcal{V}_i(r)$, $(I^{l'l}_{n'n}[\mathcal{V}_i] = \mathcal{V}_{i\,n'n}{}^{l'l})$. Therefore, the products of one-electron integral in the Hamiltonian can be written as:

$$O^{l_a l_a}_{n'_a n_a} O^{l_b l_b}_{n'_b n_b} = I^{l_a l_a}_{n'_a n_a}[1] I^{l_b l_b}_{n'_b n_b}[1]$$
$$\mathcal{V}_{a\,n'_a n_a}{}^{l_a l_a} O^{l_b l_b}_{n'_b n_b} = I^{l_a l_a}_{n'_a n_a}[\mathcal{V}_a] I^{l_b l_b}_{n'_b n_b}[1]$$
$$\mathcal{V}_{b\,n'_b n_b}{}^{l_b l_b} O^{l_a l_a}_{n'_a n_a} = I^{l_b l_b}_{n'_b n_b}[\mathcal{V}_b] I^{l_a l_a}_{n'_a n_a}[1].$$

Each one of these one-electron integrals can be organized in a matrix $\mathbf{I}^{l'l}[f]$ of size $N_i \times N_i$ with $i = a, b$ labeling each of the two electrons of the system. This identification into a one-electron matrix enables a straightforward mapping to a grid of blocks and threads in the GPU.

We divide the matrix $\mathbf{I}^{l'l}[f]$ in blocks of size $\mathtt{TILE}_W \times \mathtt{TILE}_W$. This naturally defines a two-dimensional grid of $(N_i/\mathtt{TILE}_W \times N_i/\mathtt{TILE}_W)$ blocks of $\mathtt{TILE}_W \times \mathtt{TILE}_W$ threads per block. We will also assume that $N_i \bmod (\mathtt{TILE}_W) = 0$, that is, the size of the one-electron basis N_i is a multiple of the thread tile size $\mathtt{TILE}_W$ per block. This is not usually a very stringent restriction, but if this is not the case, the simplest solutions is to pad the basis and matrix with zeros until the next multiple of $\mathtt{TILE}_W$ is reached, a common strategy in massively parallel computing. Even though this seems to be a waste of memory, GPUs excel in very uniform calculations, and often is better to compute a block with many zeros than conditioning the code to deal with the, possibly exceptional, case of rectangular blocks of threads. Note also that since the maximum number of threads per block is 1024, then $\mathtt{TILE}_W \times \mathtt{TILE}_W < 1024$, therefore, one can choose a smaller $\mathtt{TILE}_W$ to mitigate the effect of the padding.

Another way to see this calculation scheme is to note that the N_i basis functions were computed in N_q quadrature points. Then, one can organize the numerical basis in a matrix $\mathbf{S}$ of N_q rows by N_i columns. Then, the matrix representing the elements (19) can be computed as a weighted matrix product

$$\mathbf{I}^{l'l}[f] = \mathbf{S}^T \mathbf{F} \mathbf{S}$$

where $\mathbf{F}$ is a diagonal matrix with elements $\mathbf{F}_{jj} = f(r_j) w_j$, $j = 1, \ldots, N_q$. The product of two matrices in a GPU is one of the textbook examples of a typical massively parallel calculation, and can be applied here with small modifications. Note also that one can compute both the overlap and the orthogonality matrices with one kernel, since the only difference between them is the specific value of the function f, which is precomputed over the grid of quadrature abscissas.

Once the one-electron integrals are computed, one can use them to obtain the corresponding Hamiltonian terms. We make use of the matrices $\mathbf{I}^{l'l}[f]$ and write the products $I^{l'l}_{n'n}[f] I^{l'l}_{n'n}[f]$ as Kronecker tensor products, for example:

$$O^{l_a l_a}_{n'_a n_a} O^{l_b l_b}_{n'_b n_b} = \left(\mathbf{I}^{l'_a l_a}[1] \otimes \mathbf{I}^{l'_b l_b}[1]\right)_{n'_a n_a n'_b n_b}, \quad (20)$$

and so on. Note that the resulting matrices have $N_a \times N_b$ rows and columns, i.e., the same size as each Hamiltonian block $\mathbf{h}^{l'_a l'_b l_a l_b}$. Care must be taken to address the sorting order of the basis elements in the Hamiltonian, as stated in previous section. The calculation of these Kronecker products in the GPU proceeds as follows.

The two-electron matrix is computed in a 2D grid of $(N_a \times N_b/\mathtt{TILE}_W) \times (N_a \times N_b/\mathtt{TILE}_W)$ blocks of $\mathtt{TILE}_W \times \mathtt{TILE}_W$ threads per block in the GPU. In this way, each thread computes one element of the two-electron overlap (see Fig. 2). For example, let us assume that the one-electron basis has 96 elements. The Hamiltonian, as well as the two-electron overlap matrices, has $96 \times 96 = 9216$ rows and columns. If we choose 16×16 threads per block, the kernel would need to process 576×576 blocks. Even for the biggest basis sets (256 elements per electron) and smallest number of threads per block, the block grid is well below the limits of the architecture.

The flux of the calculation starts by transferring the basis sets, quadrature weights, eigenvalues and potentials to the GPU. Memory is allocated for two one-electron overlap matrices, and one two-electron matrix. Then, a loop over the three overlap terms in (17) is performed. In each step, one call to the overlap kernel and one call to the Kronecker product kernel is included, obtaining the two-dimensional overlap in GPU. Then, this two-dimensional overlap is transferred back to the CPU, and added up to the Hamiltonian. All the calculation is performed in double precision. The number of bytes sent to GPU can be computed from the $N_q(N_a+N_b)+(N_a+N_b)$ complex double precision numbers (two one-electron basis and eigenvalues), and $3N_q$ double precision numbers corresponding to potentials and Gauss weights, totaling $[(N_q + 1)(N_a + N_b) \times 16 + 3N_q \times 8)$ bytes. The number of bytes send back to CPU is $(N_a N_b)^2 \times 16$ bytes. For example, for the biggest calculation performed, $N_q = N_a = N_b = 256$, about 2 MB are sent to GPU, and

Table 1
Computing times, speed-ups for Overlaps calculation, for different basis sizes and computing grid configuration in the GPU, with $N_q = 256$. The computing times in CPU are 44 s. For $N_a = N_b = 128$ and 277 s. For $N_a = N_b = 256$.

Basis size $N_a = N_b$	Tile size $\mathtt{TILE}_W$	Overlap matrix size $N_a \times N_b$	Threads per block $\mathtt{TILE}_W^2$	Number of blocks $\frac{N_a \times N_b}{\mathtt{TILE}_W^2}$	GPU time s	Speed up
128	8	16 384	64	256	0.92	47
128	16	16 384	256	64	1.32	33
128	32	16 384	1024	16	2.31	19
256	8	65 536	64	256	3.41	82
256	16	65 536	256	64	4.73	58
256	32	65 536	1024	16	5.84	47

Table 2
Computing times, speed-ups for tensorial Kronecker product calculation, for different basis sizes and computing grid configuration in the GPU. The computing times in CPU are 0.035 s. For $N_a = N_b = 128$, 0.92 s. For $N_a = N_b = 64$, 5.32 s. For $N_a = N_b = 96$ and 32.94 s. For $N_a = N_b = 256$.

Basis size $N_a = N_b$	Tile size $\mathtt{TILE}_W$	Full overlap matrix size $(N_a \times N_b)^2$	Threads per block $\mathtt{TILE}_W^2$	Number of blocks $\frac{(N_a \times N_b)^2}{\mathtt{TILE}_W^2}$	GPU time s	Speed up
32	16	1024^2	16×16	64	0.003	12
64	16	4096^2	16×16	256	0.015	61
96	16	9216^2	16×16	1296	0.057	93
128	16	16384^2	16×16	4096	0.147	224
32	32	1024^2	32×32	1	0.003	12
64	32	4096^2	32×32	16	0.021	44
96	32	9216^2	32×32	81	0.075	71
128	32	16384^2	32×32	256	0.219	150

4 GB are dispatched back to CPU. Clearly, the size in memory of the basis and precomputed potentials, eigenvalues and weights is negligible compared to the full Hamiltonian size.

In Tables 1 and 2 we present the computing time of these two kernels compared to the performance in one CPU core. GPU is a Tesla C2070 with 6 GB of RAM, running in a computer with a Intel i7 960 CPU and 24 GB of RAM. We explore different $\mathtt{TILE}_W$ sizes, basis sizes, which are the two limiting parameters in the calculation. For all cases, the speed up shown by the calculation in the GPU are excellent.

There are several aspects that deserve further discussion. First, the $\mathtt{TILE}_W$ parameter is tightly related to the number of threads that can concurrently run in the GPU. For the overlap case, best results are obtained with $\mathtt{TILE}_W = 8$, with a total of 64 threads per block, while the worst one is for $\mathtt{TILE}_W = 32$, for a total of 1024 threads per block. In the last case, only one block can be computed at a time in the GPU, while the best case can accommodate many more at a given time. This is consistent with the GPU operation: its better to saturate it with a lot of blocks, without reaching the limit of the maximum number of threads, 1024 in this architecture. The Kronecker product results are also along this line, obtaining the best calculation with less threads (256 total), but a bigger number of blocks (1024) for the one-electron basis set of 128 elements.

Second, note that the relative time of the Kronecker product compared to the overlap calculation is also reduced compared to the CPU: while in the CPU case, the Kronecker product consumes about 43% of the whole calculation (overlap plus Kronecker), this reduces to a mere 14% in the GPU. Third, note that the product of the eigenvalues β_{nl} that appear in the Hamiltonian are also included in these performance times.

Note that the times included in the tables are calculation times only, and does not include the overhead of starting the kernels, and the transfer time to/from the GPU.

3.2. Repulsion integrals

Calculation of two-dimensional repulsion integrals is a lot different story. First, there are several ways to decouple the two-dimensional integral into two nested integrals. We use the following decoupling

$$\begin{aligned} R^{l'_a l'_b l_a l_b l}_{n'_a, n'_b, n_a, n_b} &= \int_0^\infty dr_1 \int_0^\infty dr_2 S_{l'_a, n'_a}(r_1) S_{l'_b, n'_b}(r_2) \\ &\quad \times \frac{r_<^l}{r_>^{l+1}} S_{l_a, n_a}(r_1) S_{l_b, n_b}(r_2) \\ &= \int_0^\infty dr_2 S_{l'_b, n'_b}(r_2) S_{l_b, n_b}(r_2) r_2^l \\ &\quad \times \int_{r_2}^\infty dr_1 \frac{1}{r_1^{l+1}} S_{l'_a, n'_a}(r_1) S_{l_a, n_a}(r_1) \\ &\quad + \int_0^\infty dr_1 S_{l'_a, n'_a}(r_1) S_{l_a, n_a}(r_1) r_1^l \\ &\quad \times \int_{r_1}^\infty dr_2 \frac{1}{r_2^{l+1}} S_{l'_b, n'_b}(r_2) S_{l_b, n_b}(r_2) \end{aligned} \tag{21}$$

which is more stable from a numerical point of view [34]. Recall that these integrals are computed up to a maximum value of the radial coordinates R_{max}. Let us define the inner integral as

$$g^{l'_i l_i l}_{n'_i n_i}(r) = r^l \int_r^{R_{max}} du \frac{1}{u^{l+1}} S_{n'_i l'_i}(u) S_{n_i l_i}(u).$$

Then, the two-dimensional integral can be cast in terms of the functional (19)

$$R^{l'_a l'_b l_a l_b l}_{n'_a, n'_b, n_a, n_b} = I^{l'_b l_b}_{n'_b n_b}[g^{l'_a l_a l}_{n'_a n_a}] + I^{l'_a l_a}_{n'_a n_a}[g^{l'_b l_b l}_{n'_b n_b}] \tag{22}$$

which enables one to use the same Gauss–Legendre quadrature to compute the external integral of the two-dimensional calculation. The inner integral is a function of the lower limit of the integration domain. Since the functionals in (22) are evaluated at the Gauss–Legendre abscissas, we need to compute the inner integral at those values. Let us consider the extended set of $N_q + 1$ abscissas defined by $T = \{r_1, r_2, \ldots, r_{N_q}, R_{max}\}$, that is, the Gauss–Legendre set including the upper limit of the integrals R_{max}. We define the

Table 3
Computing times, speedups and number of repulsion integrals computed per second. The GPU calculation was performed in Tesla C2070 card, in double precision. The CPU calculation was performed in one core of an Intel i7-960 processor. Computing times are were averaged over 10 runs. The number of quadrature points is $N_q = 128$, and the number of Simpson points is $N_s = 9$.

Basis size $N_a = N_b$	Tile size $\mathtt{TILE}_W$	CPU time ms	GPU time ms	Speed up	Number of integrals	Number of integrals in CPU (10^6/s)	Number of integrals in GPU (10^6/s)
16	2	109	21	5	2^{16}	0.60	3.12
16	4	109	8	14	2^{16}	0.60	8.19
16	8	109	6	18	2^{16}	0.60	10.92
32	2	1 852	252	7	2^{20}	0.57	4.16
32	4	1 852	86	22	2^{20}	0.57	12.19
32	8	1 852	88	21	2^{20}	0.57	11.92
64	2	29 270	3 720	8	2^{24}	0.57	4.51
64	4	29 270	1 229	24	2^{24}	0.57	13.65
64	8	29 270	1 491	20	2^{24}	0.57	11.25
128	2	475 910	58 712	8	2^{28}	0.56	4.57
128	4	475 910	19 389	25	2^{28}	0.56	13.84
128	8	475 910	24 728	19	2^{28}	0.56	10.86

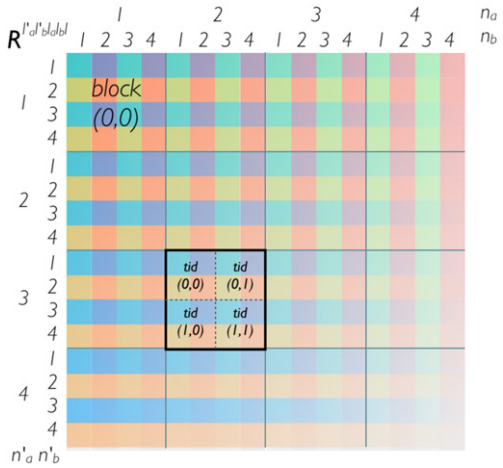

Fig. 3. Scheme of the mapping of the repulsion matrix onto the computing grid. This is an example of the mapping of the first term of Eq. (21), where the inner integrals corresponds to basis elements of the first electron, labeled by subindex a. The matrix in the figure is $\mathbf{R}^{l'_a l'_b l_a l_b l}$, with fixed values of l_a, l_b, l'_a, l'_b and l, while the basis set has a size of $N_a \times N_b = 4 \times 4 = 16$ elements. Rows are labeled by the pairs (n'_a, n'_b) whereas columns by (n_a, n_b). The computational grid has 4×4 blocks of 2×2 threads per block, as shown. The distribution of threads per block is illustrated for block (2, 1) where each thread is labeled by `tid`. In the first stage of the calculation, all the threads in each block collaborate to compute an inner integral $g^{l'_a l_a l}_{n'_a n_a}(r_j)$, which is saved in shared memory. In the second stage, each block uses this integral to compute the outer ones, and each thread takes care of 4 integrals. Note that the computational grid on the GPU starts at block (0, 0).

integral between these two adjacent abscissas in T:

$$h^{l'_i l_i l}_{n'_i n_i}(r_j) = r_j^l \int_{r_j}^{r_{j+1}} du \frac{1}{u^{l+1}} S_{n'_i l'_i}(u) S_{n_i l_i}(u), \tag{23}$$

so we can compute

$$g^{l'_i l_i l}_{n'_i n_i}(r_j) = \sum_{k=j}^{N_q+1} h^{l'_i l_i l}_{n'_i n_i}(r_j) \tag{24}$$

which is simply the cumulative sum of integrals between adjacent points from the upper limit R_{max}. Note that the domain of integration in (23) is usually small, and the integrand is smooth enough such that a simple Simpson rule with a very small number of points N_s is very accurate. However, all the basis set should be interpolated at these Simpson points to obtain the integrand. Therefore, we assume that we have each one-electron basis set S_{nl} precomputed in $N_q + 1 + (N_q N_s)$ points.

Calculation of all two-dimensional repulsion integrals runs over four Sturmians, two coming from the expansion of the wave function, and two from the projection onto the basis set. Since the integrals do not factor, four loops are needed in a CPU. The best performance is achieved when one computes one inner integral, and use that result with all the possible outer counterparts. Let us assume that we need to compute $N_O \times N_O$ outer integrals, and $N_I \times N_I$ inner integrals, such that for the first term in (21) we have $N_O = N_b$, $N_I = N_a$, while $N_O = N_a$, $N_I = N_b$, for the second term.

The calculation in the GPU has two stages. In the first one the inner integrals are computed, while the outer integral is evaluated in the second one. We define a grid of $N_I \times N_I$ blocks, with $\mathtt{TILE}_W \times \mathtt{TILE}_W$ threads per block. This setup is useful for both stages of the calculation. In the first stage, each block of threads computes one inner-integral in two steps. In the first step, each thread of the block computes the Simpson integral (23) between adjacent points, filling in an array in shared memory with them. In the second step, all threads in the block collaborate to compute the prefix-sum (24) also in shared memory. When this first stage ends, each block will have the inner integral at hand in shared memory, and ready to obtain the outer integral. In this second phase, each thread computes $N_O/\mathtt{TILE}_W \times N_O/\mathtt{TILE}_W$ outer integrals using a quadrature, similar to the calculation of the overlap matrices. An schematic of this mapping is shown in Fig. 3.

The total memory required to keep all the data is $[(N_q N_s + N_q + 1)(N_a + N_b) \times 16]$ bytes that account for all the complex basis set values computed at the Legendre quadrature and Simpson's rule abscissas plus $[2(N_q N_s + N_q + 1) \times 8]$ bytes, which corresponds to the factors r^l and $1/r^{l+1}$ and $(N_q + N_s) \times 8$ bytes for the weights of the integral scheme. Using a large basis set, for example, 128 elements for each one electron basis, $N_q = 256$ and $N_s = 9$, we have just 3 MB in memory for these data, which is negligible compared to $(N_a N_b)^2 \times 16$ bytes of the matrix Hamiltonian, which is about 4 GB that are returned to CPU.

In Tables 3 and 4 we present the calculation of these repulsion integrals for different basis sizes, using two values for the number of quadrature abscissas, $N_q = 128$ or $N_q = 256$. For all cases we get double digits speed ups respect to one-core CPU calculation. The only exception is the case for $\mathtt{TILE}_W = 2$, and $N_a, N_b < 32$. In that case, blocks of threads are too small to be managed efficiently by the GPU. For a given basis size, the best performance is obtained using $\mathtt{TILE}_W = 8$, that is to say, 64 threads per each block.

Assuming one makes use of the best mappings to compute each kind of integral, and for the largest basis set used ($N_a = N_b = 128$), most of the time of the code is spent in the calculation of repulsion integrals (92% of the time, 45.23 s). Comparing the results from all tables, one can see that the best computing times of Overlaps

Table 4
Same as Table 3, for $N_q = 256$ and $N_s = 9$.

Basis Size $N_a = N_b$	Tile size $\mathtt{TILE}_W$	CPU time ms	GPU time ms	Speed up	Number of integrals	Number of integrals in CPU (10^6/s)	Number of integrals in GPU (10^6/s)
16	2	240	44	5	2^{16}	0.27	1.47
16	4	240	15	16	2^{16}	0.27	4.29
16	8	240	10	25	2^{16}	0.27	6.84
16	16	240	9	25	2^{16}	0.27	6.93
32	2	3720	614	6	2^{20}	0.28	1.71
32	4	3720	213	17	2^{20}	0.28	4.92
32	8	3720	172	22	2^{20}	0.28	6.08
32	16	3720	186	20	2^{20}	0.28	5.63
64	2	59 880	9414	6	2^{24}	0.28	1.78
64	4	59 880	3221	19	2^{24}	0.28	5.21
64	8	59 880	2706	22	2^{24}	0.28	6.20
64	16	59 880	3336	18	2^{24}	0.28	5.03
128	2	1006 350	149 706	7	2^{28}	0.27	1.79
128	4	1006 350	51 490	20	2^{28}	0.27	5.21
128	8	1006 350	45 232	22	2^{28}	0.27	5.93
128	16	1006 350	59 153	17	2^{28}	0.27	4.54

integrals and Kronecker is $1.32 + 0.147 = 1.46$ s, or about 3% of the total time. The remaining time corresponds to the transfer from GPU to CPU of the Hamiltonian is about 2.6 s, which accounts for 5% of the run. Time of copies from CPU to GPU are negligible compared to the total time of calculation.

3.3. Overall performance

In the precedent sections it has been shown that the performance of the calculation of the Hamiltonian for a general three-body process can be dramatically increased by the use of GPUs. Let us illustrate the overall gain in performance with two examples. First, let us go back to the calculation of a ground state of Helium, a two-electron atom. In this case, $L = 0, M = 0$ and we choose five different values of single-electron angular momenta, $P^{LM} = 5$:

$$(l_a, l_b)^{00} = \{(0, 0), (1, 1), \ldots, (4, 4)\}.$$

Therefore, the full Hamiltonian matrix contains $5 \times 5 = 25$ $\mathbf{h}^{l'_a l'_b l_a l_b}$ blocks. Angular momenta selection rules determine the number of repulsion integrals that are needed for each block. For example, $\mathbf{h}^{00 l_a l_b}$ require only one repulsion integral, while five are needed to compute in $\mathbf{h}^{4444}$. The total number of repulsion blocks needed is 55. For a moderate size basis set, $N_a^{00} = N_b^{00} = 64$, the calculation reduces from more than one day (28 h) in CPU to a mere 1.45 h in GPU. These times also includes overheads due to I/O operations, but do not consider the diagonalization of the Hamiltonian performed by LaPACK.

The second example is the calculation of double ionization of He by high energy electron impact. This is a full four body process, but can be reduced to a three-body one within the First Born Approximation [27]. The Hamiltonian has three blocks of $L = 0$, 1 and 2, and we choose $P^{00} = 5$, $P^{1M} = 4$ and $P^{2M} = 4$. Again, number of repulsion blocks for $L = 0$ is 55 (this does not depend on the process), while for $L = 1$ is 50, and for $L = 2$ is 34, such that the total number of repulsion integral blocks is 139. The calculation in one CPU takes 3 days, while the GPU version lasts 4 h, for a basis size of 64 elements, and also including I/O operations.

Finally, let us summarize some details about the code. The main code (which is called IonExc, for Ionization–Excitation processes) of this collision calculation is being written in Fortran 90. To connect with the C Language Code of CUDA we make use of `iso_c_bindings` module, which is a feature of almost all new Fortran compilers [35]. This module provides a clean way to wrap C functions and call them from Fortran. Besides, copying to and from the devices is performed within the Fortran code making use of FortCuda [36]. This is also a Fortran module that provides wrappers to all CUDA API functions using `iso_c_bindings`. In this way, the code can be compiled easily with or without GPU support. IonExc uses double precision calculations, for both CPU and GPU. Intel Fortran Compilers were used for CPU, and Nvidia `nvcc` for GPU CUDA. Serial code on CPU was compiled with all optimizations enabled by the `-O3` option.

4. Summary and outlook

In this work we have shown an effective way to map the calculation of the collision Hamiltonian matrix into a GPU. This calculation implies the computation of millions of one and two-dimensional integrals with functions obtained numerically over a coordinate grid. One dimensional integrals are computed by Legendre Gaussian quadratures, while two-dimensional ones make use of a Simpson rule for the inner part of the integral. A careful analysis of the structure of this matrix shows that it has very uniform patterns of computation that can be conveniently adapted to massively parallel processors, such as GPUs. One dimensional integrals related to the overlaps among basis elements are easily mapped to a scheme of one-integral per thread, and can be related to a weighted product of matrices. Two dimensional repulsion integrals are computed with only one kernel, that progresses along the calculation in two stages. The first one computes the inner part of the integral and saves their results in shared memory. The calculation of outer integrals make use of these explicitly cached results. We have developed different kernels for each calculation, because the mapping of the data is different for overlaps and repulsion matrices.

The speed ups in performance are excellent for all cases, cutting computing times at least by an order of magnitude compared to one-core CPU calculations. Using the best values for each of the kernels for the biggest basis set, the CPU complete time is about 1068 s (almost 18 min), that transforms into a mere 49 s in GPU. Repulsion integrals take most of these times, 93% in CPU, and 92% in GPU. This means that the speed ups of overlap matrices is greater than the repulsion ones. Therefore, the percentage of time spent in the overlap calculation is smaller in GPU compared to CPU. The same behavior is observed for the Kronecker product respect to the overlap calculation: while in CPU this tensor product takes 43% of the overlap calculations, it reduces to 14% in GPU.

The optimization procedure for GPUs was presented for one particular block $\mathbf{h}^{l'_a l'_b l_a l_b}$ of the Hamiltonian, and taking into account only one term of expansion in Eq. (17). In one desktop computer with one GPU, one can compute all the l terms in the GPU one

after another, and add up the blocks in CPU. If a cluster is available, one can compute each of the terms in different processors, and combine them using MPI routines. The present approach in this case has the benefit that the load of the work is perfectly balanced among nodes, since all the calculations for each l term of Eq. (17) demand the same memory and computing resources. Therefore, there is no need to increase the complexity of the kernels to include other parts of the Hamiltonian.

The present optimization scheme can be applied to any three-body problem, regardless the masses of the particles or the coordinate system chosen to describe the position of them. For three-body systems of general masses, a non-orthogonal kinetic energy term emerges in the Hamiltonian. However, it is the product of one dimensional integrals that involve the gradients of the basis functions, which can be cast into overlap-style integrals. The method presented here to compute integrals is unaware of these details, since the inputs are interpolated functions in a quadrature grid.

A possible drawback one can find in the present approach is that for moderately large basis sets (60–100 elements per electron), one runs out of memory in the GPU. Even though GPUs devised for scientific purposes usually have 6 GB of RAM or more, this is not the case for consumer cards. In that case, it is easy to split each block $\mathbf{h}^{l'_a l'_b l_a l_b}$ in smaller tiles, and dispatch them to the GPU.

Our method is fully implemented in our code for calculation of bound states, double photoionization, and electron single ionization of atoms. Also, it is being used in the double ionization of atoms by high energy electron impact, which can be reduced to a three-body problem [27]. The extension to four or more bodies is cumbersome, however, one should be able to reuse the basic codes for integration in the radial variables. Angular integrals become more complex, due to the coupling among them.

The Sturmian method is being successfully applied to compute Fully Differential Cross Sections (FDCS) of the processes mentioned above. This cross section is differential in the ejected angles of the electrons, and their momenta, and in the total energy of the collision. Other cross sections (double differential or single differential) can be computed integrating the FDCS. Integration on the angles do not require the recalculation of the Hamiltonian, however, for different ejected momenta and total energies, different basis should be used to optimize convergence, and therefore, different Hamiltonians have to be computed. The calculation of many Hamiltonians can be easily parallelized in a cluster, for example, setting each CPU+GPU node of the cluster to compute one of them. It is not clear that this straightforward parallelization scheme would be the optimal one.

Our aim in this work was to show that finding a good map between the data and the abstraction layer of the CUDA architecture, it is possible to increase the performance of a collision code significantly. However, there is still room for more improvements. For example, even though the computation of overlaps is extremely fast in GPU, one can compute them in the CPU while the repulsion integrals are obtained in GPU. Also, one can also proceed further and perform the calculation of the basis directly at the GPU. Besides, overlapping computation and transfer of data from the GPU to CPU can also contribute to further accelerate the code. One important step in speeding up the code further would be to use single precision arithmetics in the calculation, whenever is possible, since GPUs can perform single precision calculations sometimes three times faster than double precision ones. Some results coming from the quantum chemistry arena are promising in mixing single and double precision calculation without losing accuracy [37]. Whether these methods can be applied to collisions will be a matter of subsequent studies.

Acknowledgments

The author would like to acknowledge the financial support of a grant from Fundación Balseiro and NVIDIA Corporation through its Academic Partnership Program. This work is part of projects PICT 0934/08 from the Agencia Nacional de Promoción Científica y Tecnológica (ANPCyT), PIP 200901/552 from the Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) and 06/C348 of Universidad Nacional de Cuyo.

References

[1] C. Cohen-Tannoudji, B. Diu, F. Laloë, Quantum Mechanics, Wiley, 2006.
[2] G.W.F. Drake, Springer Handbook of Atomic, Molecular, and Optical Physics, Springer, New York, 2005.
[3] G. Kresse, J. Hafner, Phys. Rev. B 47 (1993) 558.
[4] G. Kresse, J. Hafner, Phys. Rev. B 49 (1994) 14251.
[5] M.J. Frisch, et al., Gaussian 09, Revision A1, Gaussian Inc., Wallingford CT, 2009.
[6] R.G. Newton, Scattering Theory of Waves and Particles, Dover Publications, New York, 2002.
[7] R.K. Peterkop, Theory of Ionization of Atoms by Electron Impact, Colorado Associated University Press, Boulder, 1977.
[8] F.D. Colavecchia, G. Gasaneo, C.R. Garibotti, Phys. Rev. A 57 (1998) 1018.
[9] T.N. Rescigno, M. Baertschy, W.A. Isaacs, C.W. McCurdy, Science 286 (1999) 2474.
[10] Y. Furukawa, R. Koga, K. Yasuda, International Conference on Modeling and Simulation Technology, 2011.
[11] https://www.olcf.ornl.gov/computing-resources/titan-cray-xk7/.
[12] J. Sanders, E. Kandrot, CUDA by example, in: An Introduction to General-Purpose GPU Programming, Addison-Wesley Professional, 2010.
[13] NVIDIA, CUDA C PROGRAMMING GUIDE (Jul. 2013).
[14] http://www.khronos.org/opencl/.
[15] G. Gasaneo, A. Frapiccini, L.U. Ancarani, D.M. Mitnik, J.M. Randazzo, F.D. Colavecchia, Adv. Quantum Chem. 67 (2013) 153.
[16] A.S. Kadyrov, A.M. Mukhamedzhanov, A.T. Stelbovics, I. Bray, F. Pirlepesov, Phys. Rev. A 68 (2003) 022703.
[17] M.S. Pindzola, D.R. Schultz, Phys. Rev. A 53 (1996) 1525.
[18] M. Baertschy, T.N. Rescigno, W. Isaacs, X. Li, C. McCurdy, Phys. Rev. A 63 (2) (2001) 022712.
[19] M.S. Pindzola, F. Robicheaux, S.D. Loch, J.C. Berengut, T. Topcu, J. Colgan, M. Foster, D.C. Griffin, C.P. Ballance, D.R. Schultz, T. Minami, N.R. Badnell, M.C. Witthoeft, D.R. Plante, D.M. Mitnik, J.A. Ludlow, U. Kleiman, J. Phys. B: At. Mol. Opt. Phys. 40 (7) (2007) R39.
[20] I. Bray, A. Kheifets, D.V. Fursa, J. Phys. B: At. Mol. Opt. Phys. 35 (2002) R117.
[21] M.S. Mengoue, M.G.K. Njock, B. Piraux, Y.V. Popov, S.A. Zaytsev, Phys. Rev. A 83 (2011) 052708.
[22] A. Frapiccini, J.M. Randazzo, G. Gasaneo, F.D. Colavecchia, J. Phys. B: At. Mol. Opt. Phys. 43 (10) (2010) 101001.
[23] J.M. Randazzo, A. Frapiccini, L.U. Ancarani, G. Gasaneo, F.D. Colavecchia, Phys. Rev. A 81 (2010) 042520.
[24] J. Avery, Generalized Sturmians and Atomic Spectra, World Scientific, Singapore, 2006.
[25] D.A. Varshalovich, A.N. Moskalev, V.K. Khersonskiĭ, Quantum Theory of Angular Momentum: Irreducible Tensors, Spherical Harmonics, Vector Coupling Coefficients, 3nj Symbols, World Scientific, 1988.
[26] A. Frapiccini, J.M. Randazzo, F.D. Colavecchia, G. Gasaneo, Phys. Rev. A 79 (2009) 022507.
[27] G. Gasaneo, J.M. Randazzo, D.M. Mitnik, L.U. Ancarani, F.D. Colavecchia, Phys. Rev. A 87 (2013) 042707.
[28] D.M. Mitnik, F.D. Colavecchia, G. Gasaneo, Computational methods for generalized sturmians basis, Comput. Phys. Comm. 182 (2011) 1145.
[29] J.J. Dongarra, E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, Lapack User's Guide, third ed.
[30] J. Kurzak, E. Agullo, J. Demmel, J.J. Dongarra, B. Hadri, J. Langou, H. Ltaief, P. Luszczek, S. Tomov, J. Phys. Conf. Ser. 180 (2009) 012037.
[31] G. Quintana-Ortí, E.S. Quintana-Ortí, E. Chan, R.A. Geijn, F.G. Van Zee, Design and scheduling of an algorithm-by-blocks for the lu factorization on multithreaded architectures FLAME working note# 26, FLAWN 26.
[32] NVIDIA. CUDA parallel computing platform [online].
[33] M.J. Ambrosio, Simple y doble ionización de Helio por impacto de electrones (Ph.D. thesis), Departamento de Física, Universidad Nacional del Sur, Departamento de Física, Universidad Nacional del Sur (Nov. 2013).
[34] E. Foumouo, G.L. Kamta, G. Edah, B. Piraux, Phys. Rev. A 74 (2006) 063409.
[35] M. Metcalf, J. Reid, M. Cohen, Modern Fortran explained, in: Numerical Mathematics and Scientific Computation, Oxford Univ. Press, Oxford, 2011.
[36] D. Car, FortCUDA.
[37] A. Asadchev, M.S. Gordon, Comput. Phys. Comm. 183 (2012) 1563–1567.