GPU Accelerated Implementation of Density Functional Theory for Hybrid QM/MM Simulations

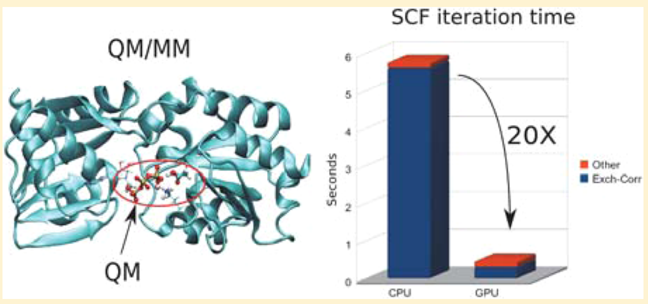
Matías A. Nitsche,[†] Manuel Ferreria,[‡] Esteban E. Mocskos,[‡,†] and Mariano C. González Lebrero*,[§]

[†]Departamento de Computación, FCEN, UBA, Buenos Aires C1428EGA, Argentina

[‡]Centro de Simulación Computacional para Aplicaciones Tecnológicas, CSC, CONICET, Buenos Aires C1425FQD, Argentina

[§]Instituto de Química y Fisicoquímica Biológicas, IQUIFIB, CONICET, Buenos Aires C1425FQD, Argentina

ABSTRACT: The hybrid simulation tools (QM/MM) evolved into a fundamental methodology for studying chemical reactivity in complex environments. This paper presents an implementation of electronic structure calculations based on density functional theory. This development is optimized for performing hybrid molecular dynamics simulations by making use of graphic processors (GPU) for the most computationally demanding parts (exchange-correlation terms). The proposed implementation is able to take advantage of modern GPUs achieving acceleration in relevant portions between 20 to 30 times faster than the CPU version. The presented code was extensively tested, both in terms of numerical quality and performance over systems of different size and composition.



1. INTRODUCTION

The simulation of chemical properties in complex systems (solution, proteins, etc.) with electronic detail generally requires treatment by means of computationally expensive methods. One approach consists of treating these systems using hybrid (QM/MM) methods. In this approach, the system is divided into a subsystem treated with a Hamiltonian based on quantum mechanics whereas the rest is modeled with a classical Hamiltonian. This methodology allows for the treatment of complex systems with many degrees of freedom. However, the computational cost associated with the resolution the self-consistent electronic problem remains a major constraint when applying this type of model.

Nevertheless, given the immense computing capabilities of the current graphics processing units (GPU), these appear as attractive alternatives in the area of high-performance computing. In particular, the use of GPUs in quantum chemistry has allowed us to obtain interesting results. There are several works that employ GPUs in diverse electronic-structure calculations,[1−7] and there is even a commercial software developed exclusively for this kind of hardware.[8] A particularly relevant work is Yasuda's,[9] in which an algorithm for the exchange-correlation calculations related to self-consistent field iterations (SCF) is presented. It is important to take into consideration that the possibility of obtaining efficient algorithms depends strongly, aside from the hardware to be used, on the type of system that have to be solved (size, type of atoms, basis-functions, etc.).

In our case, the use of hybrid simulation techniques (QM/MM) is of great interest to study bio-molecules' active sites: the presented implementation is oriented toward these systems, which usually are not bigger than 50 or 100 atoms and may include relatively heavy elements such as iron, sulfur, copper, etc.[10−17] Molecular dynamics simulations involve performing complete density functional theory (DFT) calculations a large number of times, each with only a few iterations. This implies that the initialization time has a greater relative weight in this kind of calculation than in a single-point SCF computation.

In summary, we propose a GPU implementation oriented toward QM/MM molecular dynamics calculations focused on the most computationally demanding steps of a DFT calculation with Gaussian basis. This work is based on the code Molecole[18] and includes novel approaches having a positive impact on parallelization and performance without affecting numerical quality. One of these differences consists of including a new partitioning strategy for the set of quadrature points, which results in a more efficient grouping of computational batches in terms of performance and significative functions. Another aspect involves using a low-cost classification criteria for determining these significative functions, which does not require computing the actual function values. Although other implementations[19] proposed the recalculation of function values several times at each iteration, in our implementation, we precompute these, obtaining notable performance improvements.

A CPU implementation was also developed and compared to the GPU version. The CPU version is not simply a translation of the GPU implementation because specific CPU features (like SSE4.1/4.2 or AVX) were used to obtain the best performance possible. Finally, we present and test a hybrid code made by the coupling of our DFT implementation with AMBER 12.[20]

Received: April 16, 2013

2. METHOD

In the DFT approach, the energy is written as a functional of the density:

$$E[\rho] = T_s[\rho] + V_{ne}[\rho] + \frac{1}{2}\int\int \frac{\rho(\vec{r}_1)\rho(\vec{r}_2)}{r_{12}} d\vec{r}_1 d\vec{r}_2 + E_{xc}[\rho] \quad (1)$$

where the first term is the kinetic energy associated with the density, the second is the interaction between the density and the nuclei, the third one is the Coulombs repulsion of the density with itself and the last is the exchange and correlation energy.[21]

The global exchange-correlation portion is the most expensive in terms of computational cost. The energy corresponding is calculated by the integral of the local exchange-correlation energy as the following:

$$E_{XC} = \int \rho(r)\varepsilon_{xc}(\rho(r))dr \quad (2)$$

Equation 2 can be computed as a discrete sum over a grid:[22]

$$E_{XC} \cong \sum_j \rho(r_j)\varepsilon_{xc}(\rho(r_j)) \quad (3)$$

where the electronic density ρ over each grid point j is defined from the molecular orbitals ψ_i as the following:

$$\rho(r_j) = \sum_i |\psi_i(r_j)|^2 \quad (4)$$

with

$$\psi_i(x, y, z) = \sum_{k=1}^{n} c_i^k \chi_k \quad (5)$$

where c_i are the variational coefficients, and the orbitals ψ_i are constructed by expanding them in a basis of contracted Cartesian Gaussian functions as the following:

$$\chi_k = (x - x_0)^{n_x^k}(y - y_0)^{n_y^k}(z - z_0)^{n_z^k} \sum_j k_j^k e^{-\alpha_j(\vec{r}-\vec{r}_0)^2} \quad (6)$$

where

$$(\vec{r} - \vec{r}_0)^2 = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 \quad (7)$$

An alternative way to compute the electronic density is using the density matrix:

$$\rho(\vec{r}) = \sum_{i=1}^{m}\sum_{j\geq i}^{m} p_{ij}\chi_i(\vec{r})\chi_j(\vec{r}) \quad (8)$$

where m is the number of basis functions (or basis functions with a non negligible value) and

$$p_{ij} = \sum_{k=1}^{n} C_i^k \times C_j^k \quad (9)$$

where n is the number of occupied orbitals.

The computation of the exchange and correlation energy (and the corresponding Khon–Sham matrix elements) involves several steps, for all of which a linear-scaling algorithm exists. Still, all steps exhibit a degree of parallelism inherent to their mathematical formulation. Exploiting this aspect gives a notable advantage over sequential procedures. Nevertheless, there is not a well-established parallel implementation because the computation can be resolved in different ways. One possible solution consists in parallelizing these steps independently and determining the most adequate parallelization strategy for each one. In relation with the diagram shown in Figure 1, the main computational steps are (a) quadrature-point positions, (b) quadrature-point weights, (i) function values, (j) density at each point and (k) Kohn–Sham matrix elements.

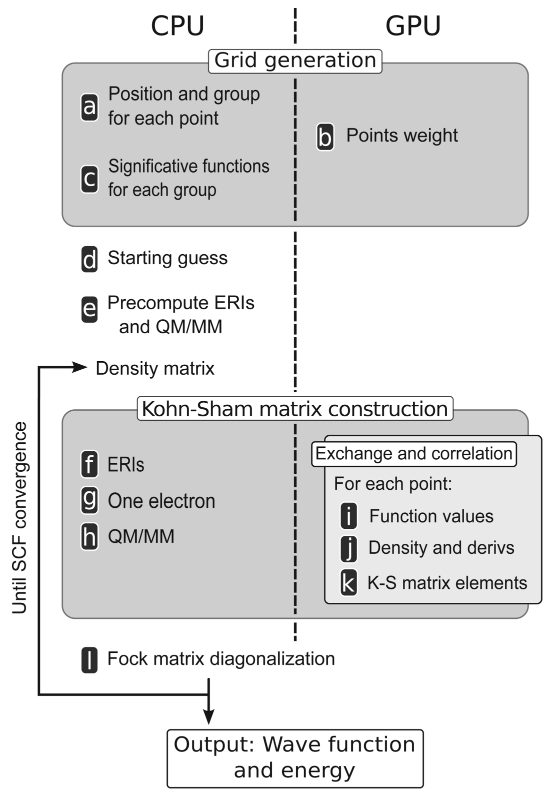


Figure 1. Calculations steps. The white box inside Kohn–Sham matrix construction module is the optimized code for GPU.

To achieve a linear-scaling implementation, Stratman et al. propose several strategies.[23] The main idea consists in grouping quadrature-points instead of solving the computation for each one. This grouping allows determining which basis-functions have a significative contribution to the final computation, which are referred to as significative functions. Given the rapid decay of Gaussian functions, the size of the set of significative functions associated with each group of quadrature points does not depend on the number of atoms of the system. In other words, this size has constant order in terms of computational complexity. As a consequence, it is possible to subdivide the complete DFT calculation by computing each of these groups independently.

A simplified scheme of the calculation procedure is shown in Figure 1. The steps (a) to (e) correspond to the initialization stage; they are computed once at the start of simulation. The SCF iteration is composed of steps (f) to (l); this cycle is repeated until the density matrix changes less than a previously established tolerance. The selected steps parallelized to be run in GPU are (b), (i), (j) and (k), which are the most computationally expensive parts of the code.

2.1. Grid. An important aspect of the calculation is the shape of the grid on which eq 3 is applied. The usual practice is to generate a grid for the molecule via atomic overlapping grids.

These atomic grids come from the superposition of layers derived by scaling a reference layer (see Figure 2).

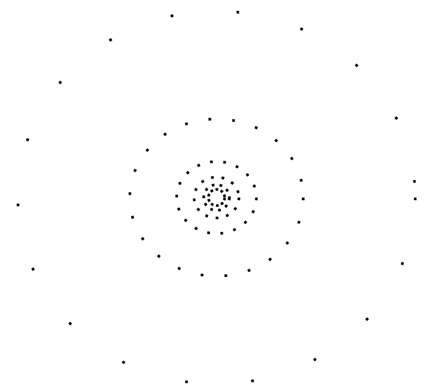


Figure 2. Schematic atomic grid. Layers are not equidistant but are most concentrated close to the nuclei.

These layers are not equidistant but are most concentrated close to the nuclei (see Figure 3), where the electron density

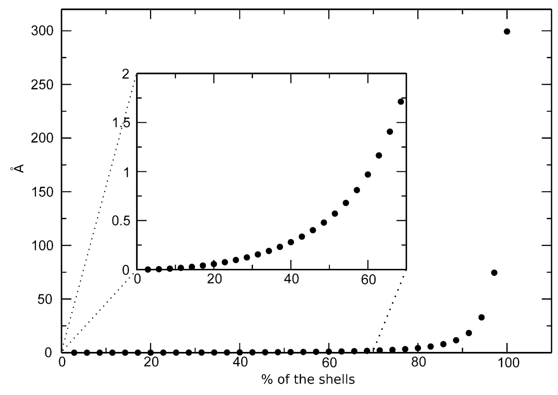


Figure 3. Radii in Angstroms of the grid layers for the oxygen atom.

changes more abruptly, and are more spaced away from them. In a molecule the overlap of atomic grids causes that the relative weight of a given point depends on the position of the grid points from other atoms, making the calculation scale quadratically.[22] However algorithms that scale linearly have been developed.[23]

2.2. Partitioning and Function Selection Criteria. The simplest partitioning scheme consists of dividing the whole system volume into fixed-size cubes, therefore grouping neighboring points. However, the distribution of points in space is not homogeneous as a result of the grid shape, which concentrates a large number of points near the nuclei where the electronic density changes more (see Figure 3).

These spheres occupy a small space, so there are a reduced amount of significative functions to compute despite having a large quantity of points. This produces a reduction in the computational cost, both in GPU and CPU. Then, after excluding these already grouped points from the complete point cloud, the traditional cube-based partitioning is applied. Using this hybrid partitioning, the cube sizes can be incremented, producing fewer groups. The result of this hybrid partitioning scheme is a more homogeneous distribution of points in terms of computational cost due to having fewer significative functions in the sphere groups (see Figure 4).

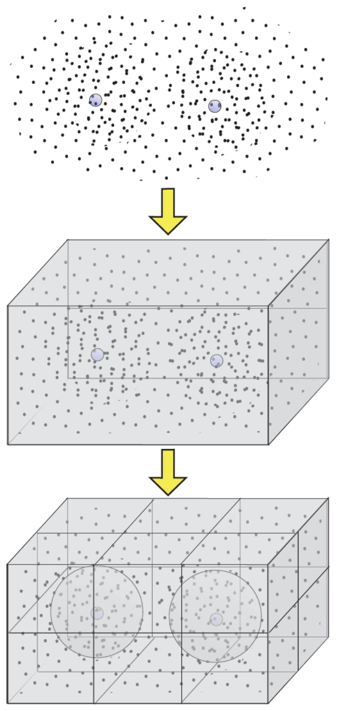


Figure 4. Grid partition. Combining spheres and cubes (hybrid partition) produces fewer groups and a more homogeneous distribution of points in terms of computational cost.

Another optimization implemented in this work consists of a simpler selection criteria for significative functions similar to other packages (for example, Gaussian). Independent of the shape of a group, the selection criteria determines which functions are to be computed over the contained points. A simple criteria consists of determining, for each basis function, if the value for that point is greater than a specific threshold δ. However, this gives the partitioning and selection criteria a considerable computational cost, because it implies computing all the function values for all points. Even further, the cost of computing a basis-function value is not negligible because it involves calculating several Gaussian functions.

The used selection criteria is based on finding a distance of influence for each Gaussian function. Outside this radius of influence the function has a negligible value (lower than a parameter: $e^{-\delta}$). The distance between the Gaussian function center and the group border is calculated, and if the function has a radius of influence larger than this distance the function is discarded.

This condition simplifies the computation by not requiring the actual function value on every point in the group. It should be noted that this inclusion condition is only sufficient and not necessary. This means that some basis functions not meeting this exclusion criteria are still computed but contribute with a negligible value to the calculation. Although, the number of these functions is small and do not impact on the resulting computational cost.

As a consequence, with this selection method the computation time for an iteration can be slightly higher than with a more rigorous method but the grid generation time is considerable lower.

2.3. Computation Kernels. During group partitioning, the positions of the points have to be computed to determine in which group they will be contained. After the partition is obtained, the list of significative functions of each group is computed. These steps (a and c in Figure 1) are computed in the CPU, because the corresponding execution time is negligible in both cases.

The weights of the grid points are computed by a GPU kernel (b in Figure 1), because this is the most computational demanding portion of the grid generation step. This kernel maps one thread to one grid point. Shared memory is used as a cache to store the atom positions and other relevant parameters.

The exchange-correlation calculation in the SCF cycle starts with computing the function values (i in Figure 1) in GPU. Again, one thread is mapped to a single grid point. The necessary Gaussian parameters are first loaded into shared memory because these do not depend on the grid point position. The function values and their first and second derivatives are stored in GPU global memory to be used during the remaining steps of the calculation for each group. The size of the memory present in the current GPUs allows storing a great part of these function values (or even all of them) in GPU memory during all the SCF iterations. This allows reusing them and drastically diminishes the time of this calculation for the successive steps. When the next group of points is computed, the value of the functions (and its gradients and Hessians) are recalculated only for the groups that do not have them already stored in GPU global memory.

This approach falls between a fully caching algorithm, where all functions are precalculated and stored for all groups, and a fully recalculating algorithm, where none of the function values are stored but instead are computed on demand for every iteration. The latter approach corresponds to Yasuda's work,[9] where it was shown that the raw processing power of GPUs could be applied in such way to gain significant performance improvements. In our case, we took advantage of newer GPU boards, which feature faster memory access and increased memory capacity.

The next step of the algorithm consists of computing the density value, the gradient and the Hessian of the density (j in Figure 1). These steps are the most intensive in terms of floating points operations. For this reason, a more detailed explanation of the algorithm is presented.

The calculation performed by this kernel is for all points in the grid using the eq 8. The gradient and Hessian are obtained using a similar expression.

$$\rho(\text{point}) = \sum_{i=1}^{m} \chi_i(\text{point}) \sum_{j\geq i}^{m} p_{ij}\chi_j(\text{point}) \tag{10}$$

where m is the number of significative functions, p_{ij} is the reduced density matrix and χ_i (χ_j) are the values of the i (j) function on the considered point. The values of the functions (χ_j) are shared for each point, but the density matrix (p_{ij}) is different for every pair of functions.

In the development of GPU scientific applications, reusing data reads from global memory as much as possible is desirable to minimize latency. The dependence of the terms in eq 10 makes it difficult to do so. There are two possible approaches for implementing step j from Figure 1: a point-based parallelization that may share elements of the density matrix but not the function values and its derivatives (see strategy 1 in Figure 5) or a function-based parallelization that can share the function values but not the density matrix elements. Initially, a point-based parallelization was implemented, in which each thread computes the entire calculation for a single point. However, the performance of this method was not sufficiently acceptable and was discarded.

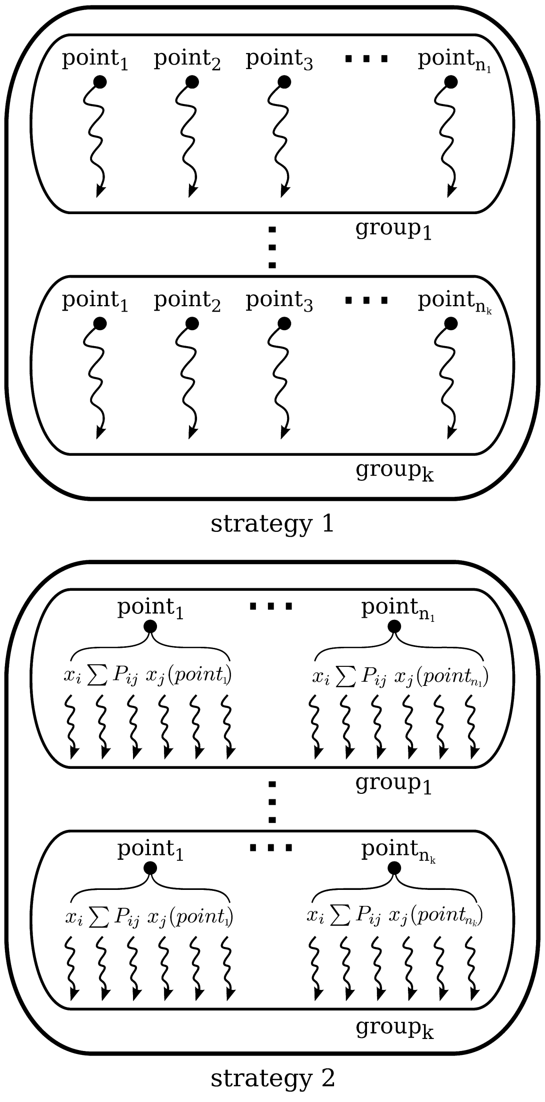

point$_1$ point$_2$ point$_3$ point$_{n_1}$
group$_1$
point$_1$ point$_2$ point$_3$ point$_{n_k}$
group$_k$
strategy 1
point$_1$ point$_{n_1}$
$x_i \sum P_{ij}\, x_j(point_1)$
$x_i \sum P_{ij}\, x_j(point_{n_1})$
group$_1$
point$_1$ point$_{n_k}$
$x_i \sum P_{ij}\, x_j(point_1)$
$x_i \sum P_{ij}\, x_j(point_{n_k})$
group$_k$
strategy 2

Figure 5. Two possible approaches for implementing step j (Figure 1). Strategy 1: point-based parallelization (shares elements of the density matrix but not the function values and its derivatives). Strategy 2: function-based parallelization (shares the function values but not the density matrix elements).

In the second approach (see Strategy 2 in Figure 5), each thread computes two terms $\chi_i(\text{point})\Sigma_{j\geq i}^{m} p_{ij}\chi_j(\text{point})$, corresponding to the eq 10; these threads are grouped in an optimal parameter named DENSITY_BLOCK_SIZE (DBS, for the current GPU generation the optimal value is 32). A two-dimensional block grid was used, where the x-dimension gives the point in the grid and in the y-dimension the number of groups of size DBS needed to compute all significative functions. Finally, a different kernel performs the summation (i.e., the reduction) over the groups for the same point and computes the contribution to the energy and gradients for that point.

This approach has several advantages. Each value of a function, which is shared for all i for the same point, is read only once from global memory to shared memory (in a coalesced form) and is used 2 × DBS times without bank

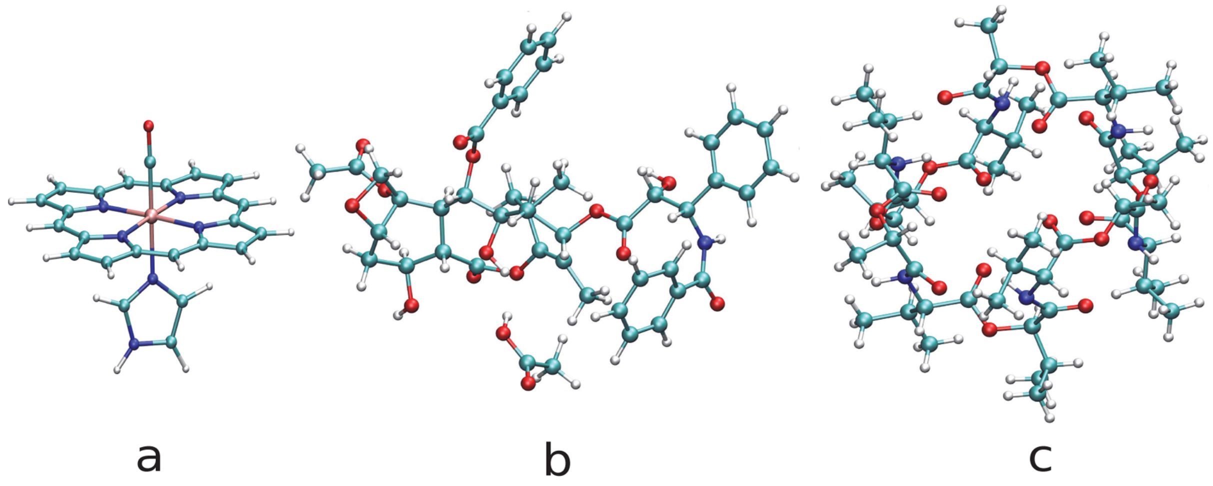


Figure 6. Schematic representation of the systems used to test the implementations: (a) heme group without any lateral chains and bound to carbon monoxide ($FeC_{23}N_6H_{16}CO$), (b) Taxol ($C_{47}H_{51}NO_{14}$) and (c) valinomycin ($C_{54}H_{90}N_6O_{18}$).

conflicts. The density matrix elements, which depend on both *i* and *j*, are accessed using texture memory (the texture memory is a specialized cache that allows efficient access to global memory following a tile pattern). This allows fast retrieval of the density matrix elements for close enough *i* and *j* (depending on the size of available texture memory).

The Kohn−Sham matrix contribution is computed using a different kernel, which is necessary during convergence of the main SCF algorithm (step k in Figure 1). In this kernel, a bidimensional grid of threads is used, where each one is mapped to a single matrix element. Because the matrix is symmetric, half of these threads would remain idle. However, because the computation is actually subdivided in blocks of threads, blocks completely contained in the lower-left triangle of the matrix are discarded without computational cost. For the blocks laying over the matrix diagonal, dummy computation is performed for the unnecessary threads to minimize branch divergence. In this step, precision is of key importance (mainly for the GGA based calculations), double-precision variables are internally used in this kernel. The rest of the GPU kernels can use single-precision variables without significative impact on the final result. As a result a mixed precision code are used.

2.4. Coulomb Integrals. Another highly CPU demanding part of the electronic-structure calculations corresponds to the Coulomb integrals, which account for the repulsive interaction between electrons (ERIs, third term in eq 1 and step f in Figure 1). Previous works have proposed the use of GPU for computing these integrals.[1,3] In medium size systems, storing ERIs in RAM memory drastically reduces the calculation time of this term compared with the exchange and correlation. Therefore, in our code, this contribution is computed in CPU and stored in main memory.

In principle, this integral depends on four centers. This would imply a complexity of $O(m^4)$ (see eq 8). However, this complexity can be easily reduced.

In particular, we use two approaches to reduce the complexity. The first one consists in rewriting the density as a linear combination of Gaussian functions (instead of a product of Gaussians) by employing auxiliary basis functions. This reduces the complexity by one, while also reducing the pre-exponential factor (in general, the number of necessary functions is much lower than the number of products associated with the original basis). The second approach for reducing the complexity without compromising the numerical quality consists in employing the Gaussian Product Theorem[23] to discard negligible terms. By using these techniques, a great reduction in the number of terms to compute is achieved. The obtained complexity is $O(m \times m')$, where m' is the number auxiliary basis functions. Then, other optimization are applied. Using the same criteria based on Gaussian product theorem, the coulomb integrals can be classified in two groups depending of its absolute value, the group with the smaller values can be stored using single precision with a negligible loss of accuracy. The second group (corresponding to the larger absolute values) is stored using double precision. This optimization reduces the size of the storage needed for this terms.

All these optimizations allows to precompute and store these terms in memory (for example, the valinomycin ($C_{54}H_{90}N_6O_{18}$) with the DZVP basis uses only 5 GB of memory). As a consequence, the time consumed by this portion of the calculation is drastically reduced resulting considerable less than the time involved in the exchange-correlation integral computation.

2.5. QM/MM Implementation. The DFT GPU based code developed in our group was coupled with the AMBER 12[20] (and AMBER 11) molecular dynamics package. The periodic boundary conditions were treated with a simple cutoff scheme. Minor changes on the AMBER code were performed, in particular, the possibility of uncoupling the temperature control for the QM and MM subsystems. The coupling was made as external program adapting the routines written by Andrea Götz and Ross Walker.[24] This implementation was already used in the determination of the free energy profile of the oxidation of cisteine by peroxinitrite.[25]

3. RESULTS

Two main aspects are analyzed in the present work: performance and numerical quality. In terms of performance, the scalability of the exchange-correlation computation, the overall DFT iteration and grid generation times are considered. Comparisons are performed using GPU and CPU implementations.

The CPU and GPU versions are compared over three different systems of moderate size: Taxol ($C_{47}H_{51}NO_{14}$), valinomycin ($C_{54}H_{90}N_6O_{18}$); and a heme group without any lateral chains and bound to carbon monoxide

($FeC_{23}N_6H_{16}CO$), see Figure 6. The first two molecules are commonly used in performance measurements,[9,26] and the third is a typical example of a system that is studied using hybrid techniques.[10,13] For all cases, we employs the DZVP basis set with a DGA1 auxiliary basis set. Computations are performed using a high-density grid (194 angular points and 30 to 35 shells). The PBE[27] functional was adopted for all the calculations.

3.1. Technical Details. The graphical processor used for the tests is NVIDIA GeForce GTX 780, implementing the Kepler architecture and having 3GB of DDR5 graphical memory. This GPU was installed on a desktop PC with an Intel Core i5-3330 processor (3.0 GHz clock speed and 6MB cache size) and 8GB of memory. The CPU based implementation is tested in the same desktop PC.

The GPU implementation uses CUDA (Compute Unified Device Architecture), which is a parallel computing platform and programming model created by NVIDIA. The GPU Computing SDK used is the CUDA Toolkit Version 5.5, which is bundled with the driver version 319.60, which includes the necessary run time libraries and compiler.

Both CPU and GPU implementations are specifically optimized for the corresponding architecture. In the case of the CPU version, it consisted in a sequential version using Floating-point vector library (fvec, developed by Intel) producing SSE2 type instructions.

The parallelized version of Math Kernel Library (MKL, developed by Intel) is used for certain steps that are solved in both implementations in CPU (for example matrix diagonalization).

3.2. Numerical Quality. Two aspects mainly affect the numerical quality: the parameter δ and the use of mixed precision in the exchange correlation terms. Table 1 shows the final energy and differences for two set of parameters on the systems presented in Figure 6. In the first one, the computation is done using a CPU version without discarding any functions and exclusively using double precision. The second one is based on a GPU version with $\delta = 10$ and mixed precision. These results show, in all cases, a difference less than 1 kcal/mol ("chemical accuracy"), concluding that they are equivalent.

Table 1. Final Energy (in kcal/mol) and Differences Showing That Using a Full Double Precision or a Mixed Precision and Discarding Negligible Functions Produces Equivalent Results

	heme	Taxol	valinomycin
CPU[a]	−1625197.143	−1835014.862	−2378314.553
GPU[b]	−1625197.125	−1835014.764	−2378314.264
diff	0.018	0.098	0.288

[a]With full double precision and without discarding any function. [b]With $\delta = 10$ and using mixed precision.

After determining δ, the sizes of the cubes and the shells included in the spheres should be optimized. The optimal parameters found for maximizing performance on GPU and CPU are different. Both cases share the same percentage of shells in the spherical groups (60%), but the size of the cubes are significantly different (CPU: 1a.u., GPU: 8a.u.). The cube size is strongly related with the total computational cost. The selection of functions to be dropped is more effective with smaller cubes, reducing the number of functions and points in each one. For CPU, having large quantities of groups with small computational cost is a desirable scenario, being only limited by the available memory. In GPU, where a large amount of processing cores is available (e.g., the Nvidia GTX 780 has 2304 CUDA cores), this situation results in less efficient computation. For this reason, is normally preferred having larger cubes in despite of increasing the computational cost.

3.3. Scalability Analysis. One important aspect to verify is the linear scalability of this implementation with respect to the number of atoms (Figure 7). Execution times for the exchange-correlation calculation were measured for the GPU implementation. This was done on systems consisting of a series of water clusters of growing size (1−24 molecules) with a DZVP basis set and an auxiliary DGA1 basis set. The quadrature grids of tested have the following:[22,28,29] 194 angular points and 30 to 35 shells (depending on the element). The DFT-GGA functional PBE was used.

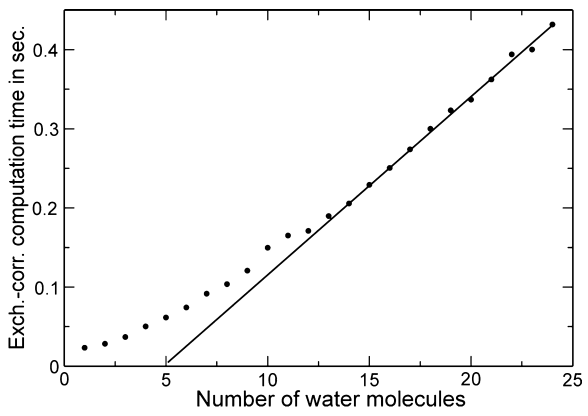


Figure 7. Scalability analysis. Best times obtained for high density grids for the water clusters systems.

3.4. Performance. *3.4.1. Group Partitioning.* The use of a hybrid group partitioning scheme based on spheres surrounding the atoms and cubes for the rest of the points have a positive impact on performance. In Table 2, we show the exchange and correlation execution time with the best parameter with and without the spheres groups, for the heme and valinomycin (both with DZVP basis set).

Table 2. Execution Times (in s) for Exchange-Correlation Terms for Different Partition Schemes in GPU[a]

	heme	Taxol	valinomycin
hybrid partition	0.55	1.40	2.55
cubes only	0.82	2.21	3.90
speedup	33%	37%	35%

[a]The use of hybrid partition presents a speedup of near 35% in the considered systems.

In all cases, the impact is similar, and the execution time is decreased by 33 to 37% without loss of accuracy in GPU (in CPU have similar results) implementation. The best results are obtained using a value between 50 and 70% of the grid points for the spheres.

In the hybrid partition scheme and using the presented parameters for GPU, the number of significative functions per group varies from only a few to more than 800 (using valinomycine as an example). However the total time is strongly dominated by the groups with more than 256

Table 3. GPU and CPU Performance Comparison for Selected Systems for One SCF Iteration[a]

		GPU	CPU_1[b]	CPU_2[c]	CPU_1/GPU	CPU_2/GPU
heme	grid gen	0.304	6.782	8.438	22	28
	SCF	1.021	25.822	16.551	25	16
	exc-corr	0.549	25.474	16.094	46	29
Taxol	grid gen	0.868	30.775	25.503	35	29
	SCF	4.246	72.562	40.028	17	9
	exc-corr	1.395	69.077	36.433	50	26
valinomycin	grid gen	1.637	72.247	43.042	44	26
	SCF	10.566	133.99	62.605	13	6
	exc-corr	2.317	125.350	53.019	54	23

[a]Times are in seconds. [b]With the same parameter as GPU. [c]With the best parameters for CPU.

significative functions. Those groups consume more than 90% of the total computing time.

3.4.2. Performance Considerations. The available GPU cards present an impressive theoretical computing power, for example the GeForce GTX 780 can process 4 TFLOPS (peak performance) in single precision. However, real applications have several limitations to achieving this outstanding performance. Examples of limitations include the need of synchronization and acceding data from memory.

One of the paradigmatic applications in scientific computing is matrix computation. The NVIDIA CUDA Basic Linear Algebra Subroutines (cuBLAS) library is a GPU-accelerated version of the complete standard BLAS library.[30]

The different matrix operations show a strong variation in terms of obtained performance. For example, SGEMM obtains 2779 GFLOPS whereas the symmetric version SSYMM only performs 876 GFLOPS (data taken from Nvidia Performance Report[31]) using K20X board with cuBLAS 5.0. This reveals the variations even in testbed scenarios focused on operations with large matrices.

In our application, the most demanding kernel corresponds to computing the density (gradient and Hessian) on each point. The number of terms to compute the eq 8 are given by the following:

$$\#_{\text{terms}} = \frac{1}{2}\sum_{\text{groups}} \#_{\text{points}}\#_{\text{fcts}} \times (\#_{\text{fcts}} + 1) \tag{11}$$

Where $\#_{\text{points}}$ is the number of points in the group and $\#_{\text{fcts}}$ is the number of significative functions for this group. In the larger case (valinomycin with DZVP basis set), the number of such terms is 4.40×10^{11}. The number of floating points operation for computing each term has a minimum of two (one multiplication plus one accumulation), discarding the reading from local, texture and shared memory, dummy computation, etc. The time needed to compute these terms in a GeForce GTX 780 is 1.37 s, leading to a result of 640 GFLOPS. This performance is similar to that achieved by the CUBLAS SSYMM in a Tesla K20x GPU,[31] a powerful board.

3.4.3. GPU vs CPU. Table 3 includes a comparison of execution times for an iteration of the SCF calculation for heme, Taxol and valinomycin with a DZVP basis set and DGA1 auxiliary basis set.

The overall performance is similar to the well-known ORCA package (version 2.9.1). Using this package, the time for an iteration using exactly the same basis, auxiliary set and a similar number of grid points is 23.53 s for heme, 65.30 s for Taxol and 122.39 s for valinomycin. Despite that the computation is not exactly the same (for example, ORCA does not store ERIs in memory) and ORCA uses more grid points (40% more in Heme, 30% more in Taxol and 20% more in valinomycin), this comparison reveals that the used CPU implementation is comparable with the state on the art in the quantum chemistry field.

For the CPU case, two sets of parameters are used: CPU_1 uses the same parameters as GPU, and CPU_2 is computed with optimized parameters for the CPU. On both cases, only one core is used for the execution. The application is compiled using the available specific compiler optimizations for the current processor architecture.

Using the best parameters for CPU (CPU_2 case) lowers the number of significative functions and thus the number of floating point operations performed. In spite of measuring different workloads, this procedure results in a fair comparison of the two computing platforms, each one with its best available scenario.

On the other hand, the use of the same set of parameters (CPU_1 case) allows us to compare the two computing platforms performing exactly the same operations.

In Table 3, the first row of each considered system corresponds to the grid generation stage. For the CPU_1 case, the speedup ranges between 22 and 44 times, showing an increment with the system size. For the other case, the obtained speedup is near 28 times in all the considered systems.

For all considered systems, the exchange-correlation part of SCF iteration represents a large part of its total computing time in both CPU cases. For example, Taxol (in the CPU_2 case) represents more than 90% of the SCF iteration.

When the same amount of operations are performed in each platform (CPU_1 case), the obtained speedup for the GPU implementation (column CPU_1/GPU in Table 3) for the exchange-correlation part ranges between 46 and 54 times. For the CPU_2 case, this speedup (column CPU_2/GPU) is 23 times for valinomycin and 29 times for heme.

These results show that use of the GPU implementation produces a drastic reduction in the computing time for the exchange-correlation part of the SCF iteration.

3.5. QM/MM Simulations. In this section, the impact of the new implementation on a QM/MM simulation is evaluated. The system used is the soluble domain of CopA ATPase from *Archaeoglobus fulgidus*. This protein couples the energy of ATP hydrolysis to Cu^+ translocation across cellular membranes.[32] In the reaction mechanism, a phosphorylated metastable intermediate is generated. This particular autophosphorylation reaction results in a interesting example of a system that can be studied using QM/MM techniques.

In the QM/MM simulation, the QM subsystem is defined by the atoms in the reactive region. This includes the phosphates of the ATP, one Mg^{2+}, the aspartic 424 (which is phosphorylated) and some of the atoms of lysine 600, totaling 34 atoms (see Figure 8). The MM subsystem is formed by the rest of the protein, the rest of ATP, explicit water molecules and Na^+ counterions. The initial coordinates are taken from the 3A1C structure of the Protein Data Bank (PDB)[32] and relaxed using classical simulation for 2 ns.

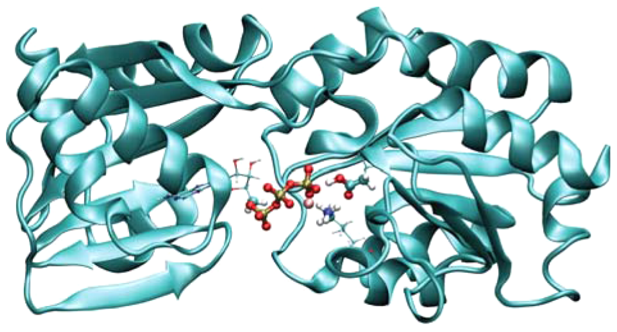


Figure 8. QM/MM system. The water molecules are not show, the QM region are shown in CPK representation.

Figure 9 shows the relative times for the one step of molecular dynamics, taking 8 SCF iterations to converge. Figure 9a includes the results for CPU based implementation using the best available parameters whereas Figure 9b is centered on the GPU version.

As was previously mentioned and can be immediately seen in Figure 9a, the exchange-correlation calculus in the SCF cycle is one of the most demanding parts in terms of computational cost. On the other hand, Figure 9b shows that this stage is notoriously diminished in the GPU version, reaching a speedup of 20×. As a consequence, the relative weight of the exchange-correlation calculus in SCF cycle is bellow QM/MM initialization, QM/MM forces and Coulomb forces. The grid generation stage and the exchange-correlation forces also present a remarkable speedup. Figure 9b shows that the relative impact of these two terms in the GPU version is minor.

4. CONCLUSIONS

The hybrid simulation tools (QM/MM) evolved into a fundamental methodology for studying chemical reactivity in complex environments. The aim of this work is to devise an electronic-structure simulation software based on DFT as efficiently as possible to be applied to hybrid molecular dynamic simulations. In this sense, two particularities of these type of calculations were considered due to their importance: the size of the systems (which usually consist of a moderate number of atoms, on the order of 100) and the nearness to convergence of the wave function at the first SCF iteration (which minimizes the number of required iterations).

In the SCF iteration part, the use of an hybrid partitioning scheme reduces the computational cost by 35% and the use of GPU contributes in an additional 20 to 30× factor (compared with the best implementation in CPU using a single core).

Although, the reduction in the time consumed by the exchange-correlation calculus reveals other parts that acquire relevance in the GPU version. As is shown in Figure 9b, some of the steps that should be considered to further optimization are QM/MM force, Coulomb force, and, to a lesser degree, other terms (basis changes, matrix diagonalization, etc).

The obtained results show a successful implementation of the exchange-correlation part of the DFT calculation that can measure up to the state of the art of the field. This new version also presents optimal characteristics to be included as part of a molecular dynamics simulation software.

AUTHOR INFORMATION

Corresponding Author

*M. C. González Lebrero. E-mail: mcgl@qb.ffyb.uba.ar.

Notes

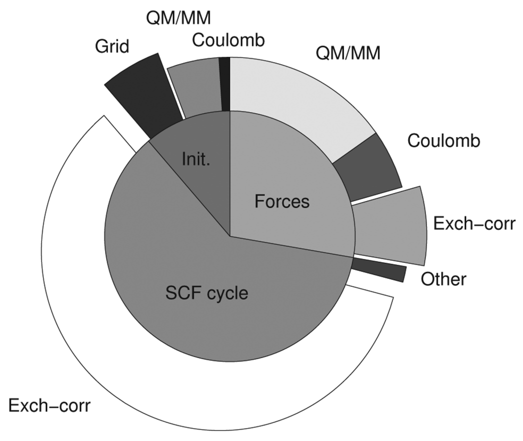
The authors declare no competing financial interest.

ACKNOWLEDGMENTS

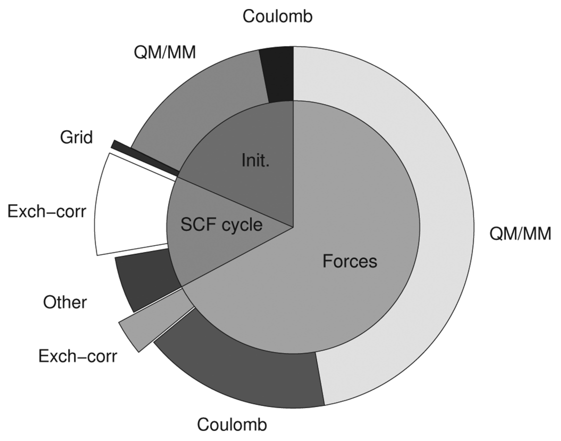
This work was partially supported by the University of Buenos Aires and the CONICET. We thank A. Roitberg for a generous allocation of computer time, Darío A. Estrin for useful discussions and David González Márquez for his help with the figures.

REFERENCES

(1) Yasuda, K. *J. Comput. Chem.* **2008**, *29*, 334−342.
(2) Vogt, L.; Olivares-Amaya, R.; Kermes, S.; Shao, Y.; Amador-Bedolla, C.; Aspuru-Guzik, A. *J. Phys. Chem. A* **2008**, *112*, 2049−2057.
(3) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2008**, *4*, 222−231.



(a) CPU version, total time 87.52s



(b) GPU version, total time 26.56s

Figure 9. Relative times for one step of molecular dynamics simulation. The emphasized portions correspond to the steps ported to GPU.

(4) Genovese, L.; Ospici, M.; Deutsch, T.; Méhaut, J.; Neelov, A.; Goedecker, S. *J. Chem. Phys.* **2009**, *131*, 034103.
(5) Stone, J. E.; Phillips, J. C.; Freddolino, P. L.; Hardy, D. J.; Trabuco, L. G.; Schulten, K. *J. Comput. Chem.* **2007**, *28*, 2618−2640.
(6) Anderson, A. G.; W. A., G., III; Schröder, P. *Comput. Phys. Commun.* **2007**, *177*, 298−306.
(7) Wilkinson, K.; Skylaris, C.-K. *J. Comput. Chem.* **2013**, *34*, 2446−2459.
(8) *TeraChem* v1.0. http://www.petachem.com (accessed February 17, 2014).
(9) Yasuda, K. *J. Chem. Theory Comput.* **2008**, *4*, 1230−1236.
(10) Capece, L.; Marti, M.; Crespo, A.; Doctorovich, F.; Estrin, D. *J. Am. Chem. Soc.* **2006**, *128*, 12455−12461.
(11) Crespo, A.; Scherlis, D.; Martí, M.; Ordejón, P.; Roitberg, A.; Estrin, D. *J. Phys. Chem. B* **2003**, *107*, 13728−13736.
(12) Crespo, A.; Martí, M.; Estrin, D.; Roitberg, A. *J. Am. Chem. Soc.* **2005**, *127*, 6940−6941.
(13) Crespo, A.; Marti, M.; Kalko, S.; Morreale, A.; Orozco, M.; Gelpi, J.; Luque, F.; Estrin, D. *J. Am. Chem. Soc.* **2005**, *127*, 4433−4444.
(14) Friesner, R.; Guallar, V. *Annu. Rev. Phys. Chem.* **2005**, *56*, 389−427.
(15) Marti, M.; Crespo, A.; Bari, S.; Doctorovich, F.; Estrin, D. *J. Phys. Chem. B* **2004**, *108*, 18073−18080.
(16) Ridder, L.; Harvey, J.; Rietjens, I.; Vervoort, J.; Mulholland, A. *J. Phys. Chem. B* **2003**, *107*, 2118−2126.
(17) Sproviero, E.; Gascón, J.; McEvoy, J.; Brudvig, G.; Batista, V. *J. Chem. Theory Comput.* **2006**, *2*, 1119−1134.
(18) Estrin, D.; Corongiu, G.; Clementi, E. In *Methods and Techniques in Computational Chemistry*; Clementi, E., Ed.; STEF: Cagliari, Italy, 1993; Chapter 12.
(19) Yasuda, K. *J Chem. Theory Comput.* **2008**, *4*, 1230−1236.
(20) Case, D. et al. *AMBER 12*; University of California, San Francisco: San Francisco, CA, 2012.
(21) Kohn, W.; Sham, L. J. *Phys. Rev.* **1965**, *140*, A1133−A1138.
(22) Becke, A. D. *J. Chem. Phys.* **1988**, *88*, 2547−2553.
(23) Stratmann, R.; Scuseria, G. E.; Frisch, M. J. *Chem. Phys. Lett.* **1996**, *257*, 213−223.
(24) Götz, A. W.; Clark, M. A.; Walker, R. C. *J. Comput. Chem.* **2014**, *35*, 95−108.
(25) Zeida, A.; Lebrero, M. C. G.; Radi, R.; Trujillo, M.; Estrin, D. A. *Arch. Biochem. Biophys.* **2013**, *539*, 81−86.
(26) *TeraChem Performance Analysis*. http://www.petachem.com/performance.html (accessed February 17, 2014).
(27) Perdew, J. P.; Burke, K.; Ernzerhof, M. *Phys. Rev. Lett.* **1996**, *77*, 3865−3868.
(28) Lebedev, V. I. *USSR Comput. Math. Math. Phys.* **1975**, *15*, 44−51.
(29) Lebedev, V. I. *USSR Comput. Math. Math. Phys.* **1976**, *16*, 10−24.
(30) The NVIDIA CUDA Basic Linear Algebra Subroutines. https://developer.nvidia.com/cuBLAS (accessed February 17, 2014).
(31) *CUDA Toolkit 5.0 Performance Report*. https://developer.nvidia.com/sites/default/files/akamai/cuda/files/CUDACUDA_5.0 Math_Libraries_ Performance.pdf (accessed February 17, 2014).
(32) Tsuda, T.; Toyoshima, C. *EMBO J.* **2009**, 1782−1791.