JCTC Journal of Chemical Theory and Computation

Article

pubs.acs.org/JCTC

GPU Linear Algebra Libraries and GPGPU Programming for Accelerating MOPAC Semiempirical Quantum Chemistry Calculations†

Julio Daniel Carvalho Maia,‡ Gabriel Aires Urquiza Carvalho,§ Carlos Peixoto Mangueira, Jr.,‡ Sidney Ramos Santana,§ Lucidio Anjos Formiga Cabral,‡ and Gerd B. Rocha*,§

‡Centro de Informática, Universidade Federal da Paraíba, CEP: 58051-900, João Pessoa/PB, Brazil

§Departamento de Química, CCEN, Universidade Federal da Paraíba, Caixa Postal: 5093, CEP: 58051-970, João Pessoa/PB, Brazil

Supporting Information

ABSTRACT: In this study, we present some modifications in the semiempirical quantum chemistry MOPAC2009 code that accelerate single-point energy calculations (1SCF) of medium-size (up to 2500 atoms) molecular systems using GPU coprocessors and multithreaded shared-memory CPUs. Our modifications consisted of using a combination of highly optimized linear algebra libraries for both CPU (LAPACK and BLAS from Intel MKL) and GPU (MAGMA and CUBLAS) to hasten time-consuming parts of MOPAC such as the pseudodiagonalization, full diagonalization, and density matrix assembling. We have shown that it is possible to obtain large speedups just by using CPU serial linear algebra libraries in the MOPAC code. As a special case, we show a speedup of up to 14 times for a methanol simulation box containing 2400 atoms and 4800 basis functions, with even greater gains in performance when using multithreaded CPUs (2.1 times in relation to the single-threaded CPU code using linear algebra libraries) and GPUs (3.8 times). This degree of acceleration opens new perspectives for modeling larger structures which appear in inorganic chemistry (such as zeolites and MOFs), biochemistry (such as polysaccharides, small proteins, and DNA fragments), and materials science (such as nanotubes and fullerenes). In addition, we believe that this parallel (GPU-GPU) MOPAC code will make it feasible to use semiempirical methods in lengthy molecular simulations using both hybrid QM/MM and QM/QM potentials.

1. INTRODUCTION

The increase in computational power available nowadays has substantially expanded the boundaries of computational chemistry, allowing the theoretical treatment of complicated molecular systems with many degrees of freedom.[1]

However, this increase in speed is not due solely to better hardware. To a significant degree, it is the result of improvements in software that can make efficient use of the computational resources available.

When computers first started to be used in chemistry, most available codes were designed to run sequentially. This scenario, however, started to change in 1981 with Rolf Seeger's publication[2] in which he showed how to reprogram time-consuming parts of the code to allow its execution in parallel. At that time, only modest speedups, two or three times faster, were obtained with this reprogramming of codes running on workstations.[2]

Nowadays, computational chemists find a different scenario in which molecular simulations are carried out using high-performance computers such as CPU clusters, multicore desktops and workstations, computers in grid, etc.

Recently, a new way of performing CPU intensive calculations has arisen, through a combined use of graphical processing units (GPUs) and multithreaded processors forming high-performance, heterogeneous many-core systems.

A consequence of this new way of performing high-demand calculations is a growing interest in their application for general purposes, such as medical diagnosis imaging, cryptography, oil prospecting, cosmology, fluid dynamics, stocks and commodities financial analysis, etc.[3]

Chemistry is one of the fields which has greatly benefited from these advancements.[4] Recently, many computational chemistry codes have been ported to explore the high computational features of GPU coprocessors, including molecular dynamics,[5−7] electron repulsion integrals calculations needed by Hartree−Fock and DFT methods,[8−10] direct SCF,[9,11] analytical energy gradient calculations,[12] MP2,[13] coupled cluster,[14,15] and others.[16−21] GPU computation has become a central tool in high performance chemistry, as documented in a recent review.[22]

These many-threaded hybrid computer architectures (GPU +GPU) allow users the chance to carry out massive computations in parallel using a computational paradigm of fine-grain granularity.

In high performance computing (HPC), one generally reprograms or fine-tunes a code to take advantage of a computer's architecture, more specifically, to take advantage of massive parallelism of GPU coprocessors.[23]

Many computational chemistry programs contain hundreds of thousands of lines of code. Modifying them in order to take advantage of parallelism of modern architectures is a task that requires substantial effort. As such, many quantum chemistry programs still run sequentially.[23]

Received: March 29, 2012

   dx.doi.org/10.1021/ct3004645 | J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

A commonly adopted strategy to develop a faster code is to find out which parts or subroutines of a program are the most time-consuming and to investigate whether they would run faster if they were modified to take advantage of a computer's architecture.

Obviously, over the years, more attention has been paid to the codes that demand a lot of CPU time, such as *ab initio*, DFT, and molecular dynamics. In part, this attention has been motivated by the possibility of simulating and modeling macromolecular systems. Conversely, codes that performed semiempirical calculations have not been improved at the same rate; indeed, the most largely distributed and widely used version of MOPAC[24] still remains fully serial.

The first effort to optimize and parallelize modern semiempirical calculation was made with MNDO code in early 1990s by Thiel and Bakowies, which enabled a complete shared-memory parallelization.[25] Benchmarks on fullerenes revealed a speedup of 7.7 on a Cray Y-MP8 (eight processors) for MNDO geometry optimization of C_{540}, which ran at 83% of the hardware limit.[25]

Moreover, a distributed-memory parallelization of MNDO code was also performed using message-passing, both at the coarse-grained and fine-grained level.[26−28]

In addition, we can find in articles and books references to parallel versions of semiempirical codes.[29−34] However, none of them have been so widely distributed or acquired a reputation for robustness and reliability as that achieved by MOPAC.

Motivated by this, we have initiated a study aiming to accelerate NDDO semiempirical quantum chemical calculations in GPUs.[35] Our modifications consisted of using a combination of highly optimized linear algebra libraries for both CPU (LAPACK and BLAS from Intel MKL) and GPU (MAGMA and CUBLAS), with new GPU kernels to hasten time-consuming parts of MOPAC like pseudodiagonalization, full diagonalization, and density matrix assembling. This study allowed us to assess the acceleration obtained for single point energy calculations with the combined use of highly optimized parallel libraries available both for CPU and GPU.

In a parallel study, Wu and co-workers[36] used similar strategies to accelerate the single point energy calculation of a semiempirical MNDO program. In addition to the use of linear algebra library routines (as in our work), they developed a GPU kernel for carrying out a sequence of noniterative Jacobi transformations during pseudodiagonalization. The authors reported that the overall computation times for single-point energy calculations and geometry optimizations of large molecules were reduced by 1 order of magnitude on a hybrid GPU-CPU platform compared to runs on a single CPU core, both for MNDO-type methods (MNDO, AM1, and PM3) and for orthogonalization-corrected methods (OM1, OM2, and OM3).[36]

The main reason to parallelize semiempirical codes is due to a renewal of interest in these methods. Nowadays, there are very well-defined problems for which semiempirical methods are more appropriate, for example, (i) the computation of a very large number of small molecules, as in the studies of the substituent effects for drug design; (ii) many and repetitive calculations from the same system, as in either molecular dynamics or Monte Carlo simulations; and (iii) the modeling of large molecular systems.

Clearly, the usage of semiempirical methods for the study of medium-size ground-state molecules of about 50 atoms, considering modern computer power, is not justified anymore; in this situation, the use of DFT methods is more appropriate, since they are, in general, more accurate than semiempirical methods, and even competitive with correlated *ab initio* methods.

Moreover, full semiempirical quantum chemical treatment of large molecular systems can be made possible by linear scaling algorithms.[37] These techniques replace the most time-consuming steps of a semiempirical calculation, such as the diagonalization of the Fock matrix (**F**) and the computation of the atomic integrals, by more efficient methodologies, thus raising the possibility of their usage for computing molecular systems that contain a great number of atoms.

Although, nowadays, it is possible to perform high level calculations, *ab initio* and DFT, on large molecular systems, because semiempirical methods are theoretically very simple, they can be used to treat larger systems at shorter times using only modest hardware. Good reviews regarding the use of high-level calculation in large molecular systems have been written.[37,38]

An outstanding example of the application of semiempirical methods to modeling proteins is the recent work by Fukushima et al.,[39] in which these authors performed a calculation on 112 solvated nonhomologous and nonredundant enzymes from the protein database PDB, using AM1.[40] These calculations were made possible by the MOZYME[41] technique. Their results indicated that enzymes could be categorized into three groups according to the amplitude of their frontier molecular orbitals, FMOs: (i) those which have extremely localized FMOs, (ii) those which have highly delocalized FMOs, and (3) the intermediates. Using these findings, the authors concluded that an enzyme's FMO degree of localization correlates well with its degree of specificity—results only achievable when electronic structure methods are applied to the problem.

On the basis of this discussion, we present in this article a set of simple modifications in the MOPAC2009 code, with the purpose of allowing its parallel execution on multicore architectures and GPUs.

The main objective of this work was to make it available for the current distributed version of the MOPAC semiempirical software to carry out calculations which benefited from both multithreaded CPUs as well as GPUs. More specifically, we present parallelization strategies for single point energy calculations both in multithreaded CPUs as well as in GPUs by using linear algebra libraries and by rewriting parts of MOPAC code. The modified code would be applied to a meaningful problem involving biomolecules.

Our results point out that these modifications produce a significant reduction in the computational time required for calculations involving molecules containing up to 2500 atoms.

This paper is organized as follows: In section 2, we present relevant computational details regarding the SCF algorithm in the newest MOPAC code. In section 3 we advance our parallel strategies to accelerate the parts of MOPAC code which carry out such procedures. In section 4, we describe all results obtained for medium-sized molecules. Section 5 presents a possible application of our findings to the study of small proteins. Finally, in section 6, we state our conclusions and final remarks.

2. COMPUTATIONAL ASPECTS IN MOPAC2009 CODE

The SCF procedure in MOPAC is driven by the ITER subroutine which is called exclusively by the COMPFG subroutine. A detailed flowchart of all of the electronic

structure calculations that can be performed by MOPAC is documented in ref 24.

The most time-consuming steps in conventional semi-empirical calculations are the Fock matrix diagonalization and density matrix assembly; these are N^3 complexity algorithms—N being the number of basis functions. The rest of the procedures carried out in the SCF have N^2 complexity, for example the assembly of Fock matrix. Because of this, most efforts are focused in reducing the time spent to call such computationally demanding procedures.

In MOPAC, subroutine DENSIT performs the assembly of the density matrix from atomic orbitals coefficients. The subroutine RSP (based on modifications of the EISPACK 3 routines TRED3, TQLRAT, TINVIT, and TRBAK3) finds all eigenvalues and some or all eigenvectors of a symmetrical packed matrix.

Indeed, the current distribution of MOPAC still uses the RSP subroutine to perform matrix diagonalizations in general. It is well-known that there are many other linear algebra libraries that are more efficient than EISPACK, such as LAPACK.[42]

MOPAC uses a strategy that was developed by Stewart and co-workers,[43] who pointed out that it is not necessary to carry out a complete diagonalization of the Fock matrix in every SCF iteration to converge the density matrix; it is sufficient to have annihilated all Fock matrix elements connecting the occupied and virtual molecular orbitals to achieve such convergence.

The pseudodiagonalization in MOPAC is carried out in subroutine DIAG, and its algorithm starts by forming the occupied-virtual block of the Fock matrix, F_{o-v}, in the molecular orbital basis of the previous iteration, eq 1.

$$F_{(o-v)} = C_o^T F C_v \tag{1}$$

where C_o is the occupied orbitals sub-matrix and C_v is the same for the virtual ones. Two matrix multiplications are necessary for the assembly of F_{o-v} in this first part.

Once F_{o-v} is built, the second part of DIAG performs a series of 2 × 2 vector rotations (unitary transformations) which approximately eliminates the elements of F_{o-v}.

Let i be indexes for the occupied orbitals and α be indexes for the virtual ones. The expressions to calculate the new eigenvectors (C_i^{new} and C_α^{new}) from the old ones (C_i and C_α) by applying successive rotations of 2 × 2 vectors are given in eqs 2 and 3.

$$C_i^{new} = cC_i - sC_\alpha \text{ and } C_\alpha^{new} = sC_i + cC_\alpha \tag{2}$$

$$s = \frac{\mathcal{F}_{i\alpha}}{\varepsilon_i - \varepsilon_\alpha} \text{ and } c = \sqrt{1 - s^2} \tag{3}$$

In these equations, ε_i and ε_α are the molecular orbital energies for the last conventional SCF iteration (using a full diagonalization procedure).

This approach significantly reduces the computational demand of the SCF procedure, making it faster.

In order to measure the percentage of time spent in different parts of the MOPAC2009 code for a single point energy calculation and to determine how many times each of the subroutines were called, a profiling analysis was performed while running the MOPAC2009 program entirely in a single CPU. A similar profiling analysis was also carried out by Wu and co-workers[36] using their MNDO code.

Test cases consisted of a single point energy calculation of C_{540} fullerene, a $(Glu\text{-}Ala)_{64}$ helix system (where Glu-Ala represents the amino acid residues of glutamic acid and alanine), and an approximately spherical water cluster, of 16.0 Å radius (named W_{16}). The geometries for the fullerene were retrieved from Hyperchem version 8.0,[44] and the geometries for both $(Glu\text{-}Ala)_{64}$ and W_{16} were retrieved from the ErgoSCF Web site (see: http://ergoscf.org/xyz.php). These analyses are presented in Table 1. A more detailed version of such profiling is presented in the Supporting Information (Tables ST1 − ST3). The MOPAC keywords used in these calculations were RM1,[45] 1SCF, XYZ, TIMES, and T=1D. In addition, the profiling inspection was run with optimization flags turned on.

Table 1. Profiling Results for Inspection of MOPAC2009 Single Point Energy Calculations Runs (In Time Percentage) for C_{540} Fullerene, $(Glu\text{-}Ala)_{64}$, and W_{16}

subroutine	description	C_{540}	$(Glu\text{-}Ala)_{64}$	W_{16}
DENSIT	calculation of one-electron density matrix	60.3	61.0	55.9
DIAG	pseudodiagonalization procedure	31.9	20.7	24.5
DDOT	BLAS dot product of two vectors	3.0	9.2	9.8
CHARMO	part of the calculation of the irreducible representation of an eigenvector	1.1		
DAXPY	BLAS constant times a vector plus a vector	1.0	3.8	4.1
ELAU	part of reduction of a real symmetric matrix to a symmetric tridiagonal matrix	0.9	2.4	2.7
COSCL1	Cholesky decomposition for the determination of the COSMO molecular area and volume	0.7	0.3	0.2
FREDA	part of reduction of a real symmetric matrix to a symmetric tridiagonal matrix	0.7	2.2	2.3
Sum		99.6	99.6	99.5

All calculations were carried out using an Intel Core I7-990X processor (six-core and hyper-threaded) clocked at 3.47 GHz with 16 GB of DDR3 main memory attached to one NVIDIA GeForce GTX 590 video card containing 3.0 GB of DDR5 memory. The FORTRAN 90 compiler used to compile MOPAC was the one included in the Intel Composer XE 2011 package. We used NVIDIA CUDA toolkit 4.1 to compile C++ codes for the Linux 64bits operational system.

Table 1 shows that subroutines DDOT and DAXPY from the MOPAC internal generic BLAS subroutines set are computationally intensive, and from Tables ST1 − ST3 (see Supporting Information) it is apparent that they are called a large number of times. These subroutines, together with subroutines ELAU and FREDA, are exclusively called by RSP (MOPAC2009 full diagonalization) in single point energy calculations and require a considerable amount of CPU time. Together, these subroutines take 5.6% of the CPU time for C_{540}, 17.6% for $(Glu\text{-}Ala)_{64}$, and 18.9% for W_{16}.

We can see that a lot of time was also spent evaluating both DIAG and DENSIT (about 92.2% for C_{540}, 81.7% for $(Glu\text{-}Ala)_{64}$, and 80.4% for W_{16}). So they were considered by us as MOPAC2009 optimization hot spots.

Subroutine COSCL1 performs a Cholesky decomposition for the determination of the COSMO molecular area and volume. This subroutine is called only once near the end of the program when all electronic calculations were done (see Tables ST1−ST3). Because such a calculation is not mandatory in

dx.doi.org/10.1021/ct3004645 | J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

MOPAC, we did not consider COSCL1 as a true hot spot; therefore, we decided to avoid calling this subroutine in all calculations reported in this study.

Subroutine CHARMO is part of the calculation of the irreducible representation of an eigenvector. This subroutine takes about 1% of the running time in the C_{540} single point energy calculation and does not have any significant impact for either $(Glu\text{-}Ala)_{64}$ or W_{16}.

In short, as a conclusion of our profiling inspection, DIAG and DENSIT are the first target subroutines to be parallelized for GPUs and shared-memory CPUs. After that, we will also consider the replacement of the RSP subroutine by more efficient full diagonalization strategies, as LAPACK.

3. STRATEGIES TO ACCELERATE MOPAC CODE FOR MEDIUM-SIZE MOLECULES

MOPAC stores symmetrical matrices, as FORTRAN vectors, saving only the upper triangle of the matrix of $(n \times (n + 1)/2)$ elements, where n is the number of basis functions, ordered column by column in column-major order.

Using this format, the program avoids allocating N^2 elements (dense and bidimensional format) for many internal matrices. MOPAC only stores in bidimensional format the atomic orbital coefficient matrix.

We should take notice that, for larger systems, both formats are inappropriate as matrix storage formats, although allocating only the upper triangle is preferable for medium-sized systems.

In the MOPAC code, a serial algorithm for multiplying symmetrical matrices is used, partially for the pseudodiagonalization and completely for the density matrix calculation.

General and symmetrical matrix multiplication (GEMMs and SYMMs) are typical linear algebra operations which can benefit from using parallel computational architectures. In some published studies, the performance of GEMMs using many linear algebra libraries has been evaluated.[3] The impact in computational chemistry codes has also been addressed.[3,9,10,13,46,47]

As can be seen from eq 1, we can use the DGEMM BLAS subroutine to perform such operations in the MOPAC DIAG subroutine. The same way, we can use either DSYRK (a subroutine that performs SYMM operations) or DGEMM in the MOPAC DENSIT subroutine to assemble the one-electron density matrix from occupied LCAO coefficients, as in eq 4.

$$P = 2C_oC_o^T \tag{4}$$

First, we have designed a new subroutine for GPU density calculation that was called DENSIT_for_GPU. In DENSIT_for_GPU, the density matrix assembling can be performed using a direct bidimensional matrix−matrix multiplication in two different forms: using cublasDSYRK (for GPU) and DSYRK from BLAS (for CPU). So, in that subroutine, we implemented two methods which will be evaluated in the results section.

Now, we shall describe the GPU and memory-shared CPU parallelization strategy that was adopted for the DIAG subroutine.

Before we proceed, it should be noted that, according to our profiling analysis, the DIAG subroutine is responsible for about 32% of the CPU time in a serial molecular energy calculation of the C_{540} molecule. So, if this subroutine could be parallelized efficiently enough, we might guarantee good speedups and lower execution times for the 1SCF calculation.

As we have detailed before, the pseudodiagonalization procedure can be divided into two parts: (i) F_{o-v} matrix building, whose expression is in eq 1, and (ii) calculation of the new eigenvectors through repetitive applications of eqs 2 and 3 for the elements in the resulting F_{o-v} that exceed a predefined threshold.

The algorithm implemented nowadays in MOPAC still has a lot of data dependency, mainly caused by the in-order filling of the vector associated with the F_{o-v} matrix. These prevent the application of any parallel model to the first step of the subroutine.

In Lin's thesis,[33] it was initially recommended that the Fock matrix in the atomic orbitals basis (which is in column-major format) be converted to bidimension format, even though it would increase the allocated memory, as working with a bidimensional Fock matrix would avoid a lot of data dependency in the calculation of the F_{o-v} matrix. The pseudocode for these algorithms as well as a detailed discussion are present in Lin's thesis.

In contrast to the strategy adopted in Lin's thesis, however, and according to the approach adopted in the MNDO program,[25−28,36] we have performed the two multiplications shown in eq 1, by taking advantage of the improved performances of linear algebra libraries developed for GPUs and equivalent libraries for multithreaded CPUs. So, we have decided to use matrix multiplication subroutines for both architectures (cublasDGEMM and DGEMM) in this step (see eq 1).

Likewise, as in the first step of DIAG, the bidimensional conversion from an upper-triangle column-major vector allows us to eliminate some data dependency for the second step of DIAG (2 × 2 Jacobi rotations). After a closer analysis of the code in that part of DIAG, we have found out that eq 2 can be calculated using the DROT subroutine from BLAS linear algebra library. Indeed, DROT applies an in-place plane rotation to vectors, updating them.

In our first attempt, we tried using the GPU version of DROT, cublasDROT. However, preliminary tests revealed that calculating eq 2 in parallel might lead to a bottleneck, because of the high rate of data transfer between the CPU and GPU memories, since for each iteration into the blocks of occupied and virtual molecular orbitals, many iterations (typically, hundreds of thousands of 2 × 2 rotations are performed for molecular systems studied here) are necessary. Then, we decided to use a single-threaded version of DROT from the Intel MKL library. In fact, we have noticed that even a multithreaded version of DROT suffers from such a delay, because both creation and deletion of threads in CPU become expensive when performed many times.

So, we have designed a new subroutine for pseudodiagonalization of Fock matrices in hybrid CPU−GPU computer architectures that is named DIAG_for_GPU. In summary, in the new subroutine, the first part of DIAG can be carried out using both cublasDGEMM and DGEMM from MKL. For the second part, we had to use only the single-threaded version of the MKL DROT subroutine.

Since the pseudodiagonalization procedure needs eigenvectors and eigenvalues as input, the SCF process in MOPAC previously performed some iterations involving full diagonalization of the Fock matrix using RSP (as we have discussed before), something which is decided internally by the SCF algorithm implemented in MOPAC and which the user does not have any control over.

dx.doi.org/10.1021/ct3004645 | J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

Table 2. Descriptions for Single Point Energy Calculations Applied to the Chosen Molecular Systems

protocol	label	platform	description
1	**STD**	CPU-only	MOPAC standard 1SCF calculation
2	**1T**	CPU-only	uses single-threaded BLAS/LAPACK from MKL for density matrix calculations (DSYRK), first part of pseudodiagonalization (DGEMM), second part of pseudodiagonalization (DROT), and full diagonalizations (DSYEVD)
3	**6T**	CPU-only	calculations are performed in six cores; uses multithreaded BLAS/LAPACK from MKL for density matrix calculations (DSYRK), first part of pseudodiagonalization (DGEMM), and full diagonalizations (DSYEVD); for the second part of pseudodiagonalization, DROT is single-threaded called
4	**GPU**	CPU-GPU	calculations performed using six CPU cores and one GPU uses cublasDGEMM for first part of pseudodiagonalization, cublasDSYRK for density matrix calculations, and single-threaded BLAS/MKL DROT for second part of pseudodiagonalization; also calls MAGMA DSYEVD for full diagonalizations

Table 3. Runtimes (In Seconds) of All Protocols Calculations and Speedups for the Model $(Glu\text{-}Ala)_n$, n = 4, 8, 16, 32, and 64

			runtimes (s)				speedups		
model	atoms	orbitals	STD	1T	6T	GPU	$S_{STD,1T}$	$S_{1T,6T}$	$S_{6T,GPU}$
$(Glu\text{-}Ala)_4$	106	274	0.40	0.28	0.18	2.25	1.43	1.56	0.12
$(Glu\text{-}Ala)_8$	210	546	3.0	1.39	0.89	3.20	2.16	1.56	0.43
$(Glu\text{-}Ala)_{16}$	418	1090	71.95	8.97	4.05	7.42	8.02	2.21	1.21
$(Glu\text{-}Ala)_{32}$	836	2178	682.91	60.80	24.93	24.55	11.23	2.44	2.48
$(Glu\text{-}Ala)_{64}$	1666	4354	7019.76	466.49	206.19	131.06	15.05	2.26	3.56

The number of full diagonalizations performed in single point energy calculations for the molecules we have used in the profiling analysis can be seen in Tables ST1−ST3, from the difference of DENSIT and DIAG calls. For fullerene C_{540}, five full diagonalizations were performed, six for $(Glu\text{-}Ala)_{64}$, and seven for W_{16} water box.

So, we have also tried to replace RSP with a full diagonalization using LAPACK diagonalization subroutine DSYEVD, implemented in the Intel MKL library, as well as with the diagonalization subroutine DSYEVD, implemented in the hybrid CPU-GPU MAGMA library.[48] Our compilation of MAGMA was carried out using the Intel MKL multithreading version and NVIDIA CUDA toolkit 4.1.

4. RESULTS AND DISCUSSIONS

In order to measure the performances of our implementations, we have explored four different single-point energy protocols. The objective of these different approaches is to establish the impact of exploring the hotspots of MOPAC with our own strategies and algorithms, as detailed in section 3, combined all together. Table 2 (and Table ST4 of the Supporting Information) describes all of these calculations.

From Table 2, we notice that the **STD** calculations run sequentially on a single CPU the entire time and are not modified whatsoever. In other words, **STD** executes MOPAC2009 as it can be downloaded from the distribution site.[24]

Calculations denoted by **1T** and **6T** had only CPU modifications. Such protocols perform, respectively, single-threaded and multithreaded calculations on the CPU. **6T** represents the best possible CPU-only calculation performed on six cores by using our modifications for 1SCF. Finally, protocol 4 includes GPU-powered modifications, so we have labeled it **GPU** (hybrid CPU-GPU code).

To run these calculations, we have recompiled MOPAC2009, applying the compiler full optimization flags for both NVIDIA CUDA C++ and Intel FORTRAN 90 compilers.

We have chosen two test sets to evaluate our modifications. The first was composed exclusively of amino acid polymers (peptides) of many sizes. These structures were labeled as $(Glu\text{-}Ala)_n$, with n assuming the values 4, 8, 16, 32, and 64. The descriptions of such molecular systems can be found in Table ST5 of the Supporting Information. These geometries were obtained from the ErgoSCF Web site (http://ergoscf.org/xyz/gluala.php).

We have compared the performance of the four protocols in running this test set, highlighting the impact of our changes in the original code of MOPAC2009.

Table 3 shows running times of all of the protocols for the test set composed of peptides. The impact of simply replacing matrix multiplication operations of DIAG and DENSIT subroutines as well as full diagonalizations performed by RSP by the equivalent procedures using MKL on the current version of MOPAC can be observed in the comparison between running times of protocols 1 (**STD**) and 2 (**1T**). We can see that it is possible to obtain speedups of up to 15 times ($S_{STD,1T}$) just by using CPU serial optimized linear algebra libraries. This was expected, since MOPAC is a legacy code and it has many nonoptimized parts. Because of these results, we have defined **1T** as the reference calculation against which all speedups (columns 6 and 7 of Table 3) will be compared.

The impact of using multithreaded linear algebra libraries can be seen, in Table 3, by the comparison between running times of **1T** and **6T** and its respective speedup $S_{1T,6T}$. We can see that this speedup increases up to the system with 836 atoms.

Finally, comparisons between protocols **1T** and **GPU** reveal the impact of using a Fermi GPU in the acceleration of the single point energy calculation for that system. The data in Table 3 show that speedups above 1 were obtained for systems with more than 1000 orbitals in its basis set. Even so, it is only advantageous to use GPU acceleration for systems with over 2000 orbitals, as can be seen by comparing running times between protocols **6T** and **GPU**. Analyzing further the speedup obtained from the comparison between protocols **1T** and **GPU**, we notice that the largest speedup (3.45) was for the molecule $(Glu\text{-}Ala)_{64}$, which has 4354 orbitals.

With the second test set, we have investigated the performance of our modifications to handle other molecular systems. Most structures are simulation boxes of molecules that are commonly used as solvents. Besides those, we also ran tests on crambin (a small protein, PDB: 1EJG), C_{540} fullerene, and nanotubes (labeled ACN, acronym for armchair nanotube).

dx.doi.org/10.1021/ct3004645 | J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

For some simulation boxes, we had to complete the empty atomic valences with hydrogen atoms. This was performed using Hyperchem 8.0. In addition, we did not carry out any further geometry optimization on those molecules. Detailed descriptions of these molecular systems are depicted in Table 4. For this test set, we have considered only systems with over 1500 orbitals.

Table 4. Descriptions of Molecular Systems in the Second Test Set

model	description	atoms	orbitals
W_{13}	simulation spherical water boxes with 13 Å of radius	903	1806
W_{14}	simulation spherical water boxes with 14 Å of radius	1152	2304
W_{15}	simulation spherical water boxes with 15 Å of radius	1413	2826
W_{16}	simulation spherical water boxes with 16 Å of radius	1719	3438
crambin	crambin is a small plant protein (PDB code: 1EJG)	642	1623
meohbox	methanol (400 AMOEBA molecules in 30.0 Å Box)	2400	4800
nmabox	N-methylacetamide (125 NMA in 25.725 Å cubic box)	1500	3375
C_{540}	fullerene	540	2160
ACN25	ACN replicated 25 times in the *c* axis from the unit cell	624	2424
ACN30	ACN replicated 30 times in the *c* axis from the unit cell	744	2904
ACN35	ACN replicated 35 times in the *c* axis from the unit cell	864	3384
ACN40	ACN replicated 40 times in the *c* axis from the unit cell	984	3864
ACN45	ACN replicated 45 times in the *c* axis from the unit cell	1104	4344
ACN50	ACN replicated 50 times in the *c* axis from the unit cell	1224	4824

In Figure 1, we present runtimes and many speedups for all protocols applied to W_n molecular systems. The chart in Figure 1 was put in a logarithmic scale of time in milliseconds for a better presentation of data. Nevertheless, the respective times in seconds are placed into each bar.

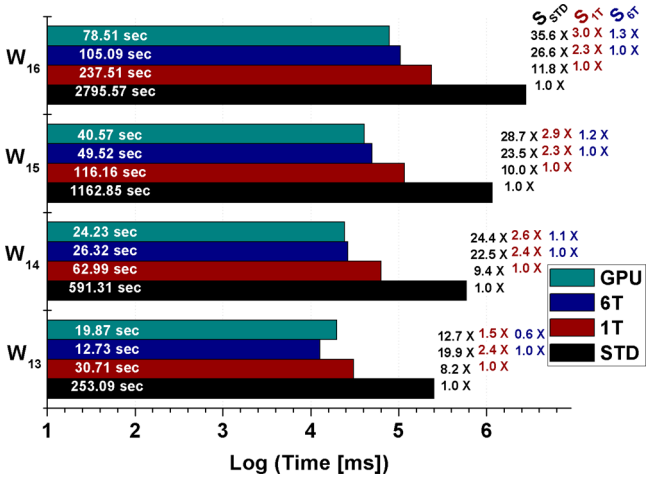

Figure 1. Runtimes (in log scale) and speedups of all calculations for the model W_n, n = 13, ..., 16. Label S_{STD} means speedup calculated in relation to protocol **STD**; S_{1T}, speedup in relation to protocol **1T**; and S_{6T}, speedup in relation to protocol **6T**.

We also present the speedups of all protocols relative to one another. S_{STD} represents the speedup of each protocol relative to the **STD**. S_{1T} represents the speedup of each protocol relative to **1T**, and S_{6T} represents the speedup relative to **6T**.

We can see that standard MOPAC single-point energy calculations are significantly accelerated by the use of the MKL linear algebra library for water clusters, both single and multithreaded. Speedups larger than 12 times can be achieved with such modifications.

Single point energy calculations using **GPU** become advantageous (in comparison with protocol **6T**) for water clusters with more than 380 molecules.

In the same way, in Figures 2 and 3, we present runtimes and many speedups for all inputs applied to the rest of the molecular systems contained in Table 4.

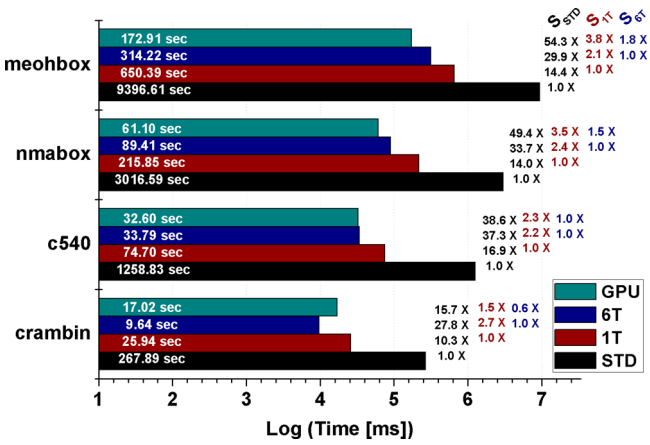

Figure 2. Runtimes (in log scale) and speedups of all calculations for other simulation boxes, fullerene, and crambin. Label S_{STD} means speedup calculated in relation to protocol **STD**; S_{1T}, speedup in relation to protocol **1T**; and S_{6T}, speedup in relation to protocol **6T**.

The data in Figure 2 show that GPU-accelerated calculation (**GPU**) for the methanol box system (4,800 orbitals) is 3.8 and 1.8 times faster than those performed using **1T** and **6T** (single and multithreaded versions of MOPAC), respectively. For the

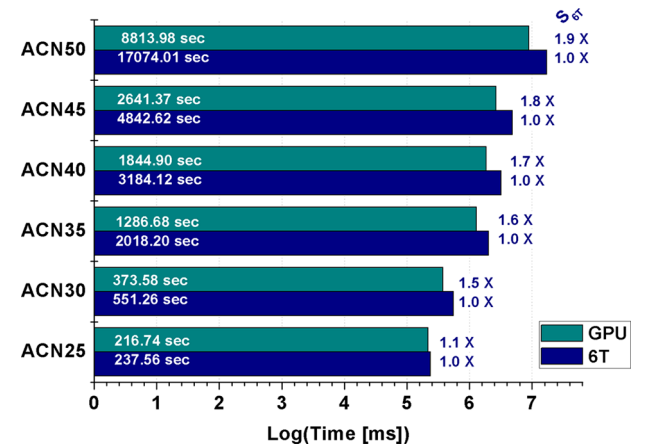

Figure 3. Runtimes (in log scale) and speedups of all calculations for armchair nanotubes (ACN). Label S_{6T} represents speedup in relation to protocol **6T**.

dx.doi.org/10.1021/ct3004645 | J. Chem. Theory Comput. XXXX, XXX, XXX–XXX

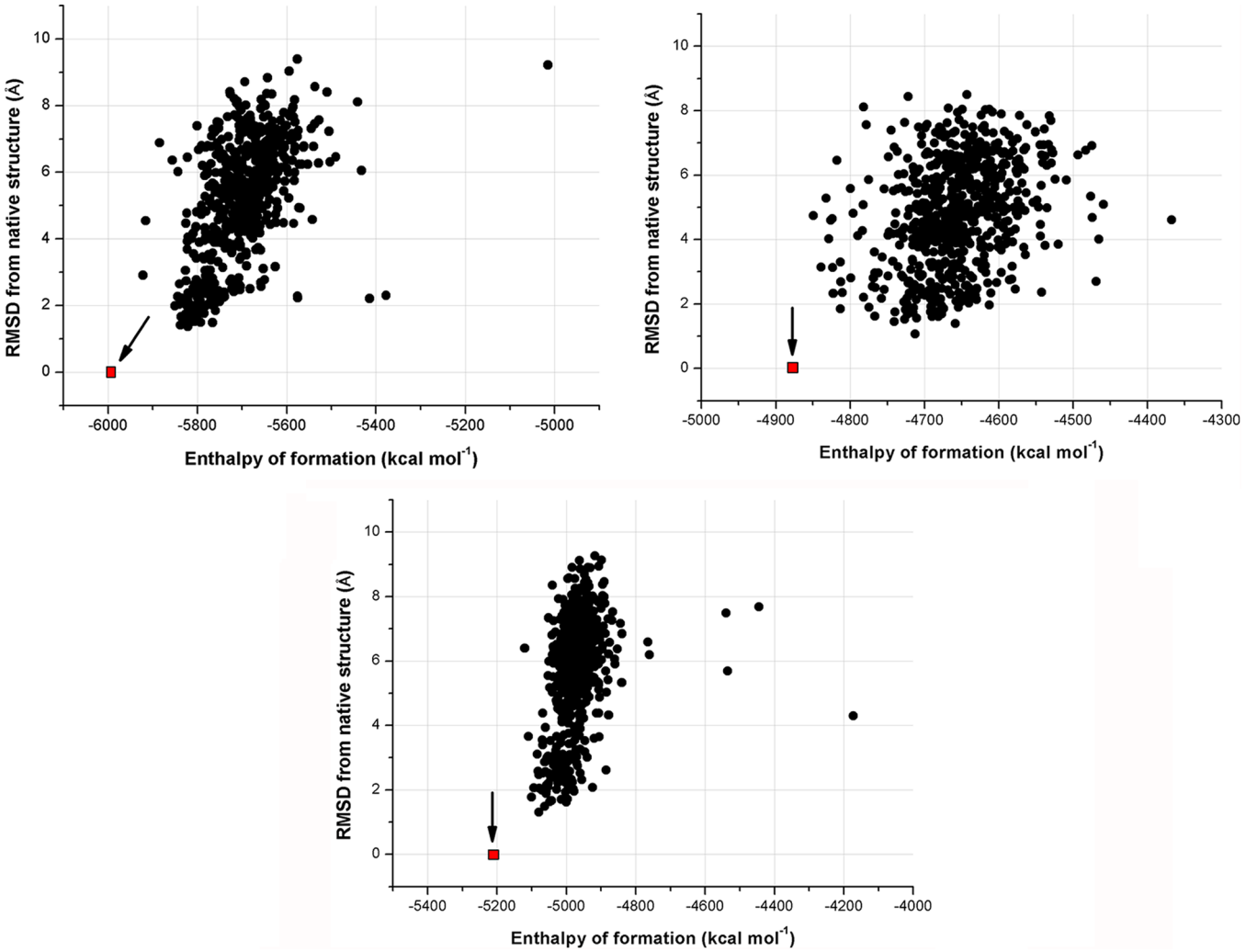

Figure 4. PM6-DH+ enthalpy of formation versus RMSD from native structure scatter plots for each decoy. Graphs are plotted for each decoy set used. Top-left, 1CTF; top-right, 1R69; bottom, 1SN3.

simulation box containing N-methylacetamide, the **GPU** was faster than **6T** by a factor of 1.5.

For the C_{540} fullerene system, the **GPU** protocol was only slightly more efficient than the best of our CPUs protocols (about 1 s). Finally, from among the molecular system contained in the Figure 2 graph, the only one in which GPU acceleration (**GPU**) could not perform faster than the best possible CPU calculation (**6T**) was the small protein crambin.

Speedups in relation to **STD** runtime for these systems show that MOPAC can have a highly enhanced performance when used with optimized linear algebra libraries for CPU as well as for GPU.

Figure 3 shows the gain in performance in the armchair nanotube series. Comparisons are only made between multi-threaded CPU and GPU implementations because single-threaded CPU versions were severely noncompetitive and took a significantly larger amount of time to complete.

The relative acceleration scaling ratio between **6T** and **GPU** increased by 0.1 for each 500-orbital increase in the system, following the trend observed in the water spheres systems.

Evidently, all speedups presented depend on the hardware used, more specifically from the relative performance between CPU and GPU. For instance, the GTX-590 device delivers 230 Gflops in double precision, while the Tesla C2070 device delivers 515 Gflops. The precise point where GPU acceleration starts to be advantageous depends on the ratio between the processing power of the CPU (in Gflops) and that of the GPU, so it is probable that this point will come increasingly closer to smaller systems as the GPU increasingly outperforms the CPU.

5. APPLICATIONS OF PARALLEL MOPAC IMPLEMENTATIONS

In order to assess how the speed increment in the semiempirical code extends the possibilities of MOPAC, we have applied the GPU-powered MOPAC in the area of biomolecule study, more specifically, to address a feature of the protein folding problem. The protein folding problem is the issue of, given the amino acid sequence of a protein, predicting the tridimensional geometry of the protein in its functional form (called native form). It is a very difficult problem for which no general solutions are currently available.

In its most fundamental form, the folding problem can be separated into two distinct parts. The first is to find an efficient and accurate way of sampling the very large conformational space of a protein. The second is to find an energy function that can discriminate accurately between the native and non-native forms of the protein geometry.[49]

dx.doi.org/10.1021/ct3004645 | J. Chem. Theory Comput. XXXX, XXX, XXX−XXX

In the effort to design accurate energy functions, sets of computationally misfolded protein conformations are being used as common grounds for an unbiased comparison of the energy functions. The accuracy of these functions is determined by their ability to correctly guess the native function from among the many other misfolded conformations. According to the thermodynamics hypothesis of the protein foldings, the native structure should be the one with the most negative free energy of folding,[50] so the energy functions need to accurately describe the folding free energy surface.

Merz and Wollacott have shown that semiempirical methods can be used to produce energy functions capable of successfully discerning between decoys and the native structure with a useful degree of accuracy.[51] In that paper, the free energy of folding for a given protein conformation was estimated as the sum of the heat of formation, free energy of solvation, and a Lennard-Jones dispersion correction from AMBER force field. The enthalpic terms were calculated using either PM3 or AM1 Hamiltonians, and the solvation term was calculated solving the finite differences Poisson−Boltzmann equation.[52]

Here, we have applied a different methodology. We used the dispersion-corrected PM6 Hamiltonian, PM6-DH+,[53] to calculate the enthalpies of formation in water, using the COSMO implicit solvent field.[54]

We have used the same four-state-reduced decoy set from Levitt, considered to be high-quality,[55] that Merz and collaborators used. From the proteins included in the set, we have chosen the PDBs 1CTF, 1R69, and 1SN3 and their decoys as our test systems.

Following the methodology of Merz and Wollacott,[51] we have minimized the structure of each decoy in the set, including the native structure, in a two-step process. First, we ran 2000 steps of restrained minimization. The atoms were restrained in their positions by means of a harmonic potential with a force constant of 2.0 kcal mol^{-1} $Å^{-2}$. Then, the output of this preliminary minimization was put through an unrestrained minimization of 5000 steps. The minimizations were carried out using the AMBER molecular dynamics package, version 11.[56] We have applied the generalized Born implicit solvation model and used the energy function of the FF99SB force field.[57] Molecular charges were determined using MOPAC.

After the minimization sequence, the GPU-powered MOPAC code was used to calculate the enthalpies of formation in water using COSMO for all of the decoys in all three sets. To assess the overall ability of the method to discriminate between native and decoy conformations, we have taken the RMSD of each decoy from its native structure and crossed these data with the PM6-DH+ enthalpies of formation. The plots containing these crossed data are shown in Figure 4. The native structure is indicated in red and by an arrow pointing at it.

It is clear from the results presented in Figure 5 that PM6 with dispersion corrections (PM6-DH+) successfully identifies the native structure from each decoy set based solely on the enthalpy of formation using COSMO's implicit solvent field. The same results could not be achieved for decoy sets 1CTF and 1R69 by Merz and Wollacott[51] using solely the enthalpies of formation calculated with AM1 and PM3 Hamiltonians. This confirms the need for corrections to improve upon the AM1 and PM3 results.

There is no clear relation between a low enthalpy of formation and geometric proximity to the native structure; however, the methodology employing solely PM6-DH+ enthalpies of formation is, indeed, able to find the native structure from among a large number of decoy conformations, provided that the native structure is included in the set.

6. CONCLUSIONS

We have developed and implemented new features in the MOPAC2009 program in order to allow it to run faster on GPUs and shared-memory CPUs. We modified it by replacing time-consuming parts of the semiempirical single point energy calculations with accelerated procedures using a combination of highly optimized linear algebra libraries for both CPU (LAPACK and BLAS from Intel MKL) and GPU (MAGMA and CUBLAS).

In short, we proposed both hybrid CPU−GPU and multithreaded CPU accelerated versions of the pseudodiagonalization, full diagonalization, and density matrix assembling methods in the MOPAC2009 code. These time-consuming parts of the SCF calculation were detected using a profiling tool to inspect a standard run of MOPAC2009.

We designed four different protocols in order to assess the real impact of our modifications on MOPAC legacy code to accelerate semiempirical single point energy calculations. For that, we have carried out calculations on different molecular systems comprised of small proteins, various solvent boxes, a fullerene, and nanotubes.

As a general conclusion, we can highlight the fact that just about every modification used brought some sort of gain to MOPAC's performance, indicating that the use of optimized linear algebra libraries (LAPACK and BLAS) provides a very powerful way to boost it, either by itself or coupled with multithreaded CPU environments (Intel MKL) as well as their GPU counterparts (CUBLAS and MAGMA).

A possible application was shown in the single-point calculation of protein decoy sets in terms of canonical molecular orbitals. Around 2000 calculations of that type were performed at a rate of around 30 s per structure. Furthermore, PM6-DH+ was shown to be capable of identifying the native structure of a protein from a large set of decoy conformations by using just the enthalpies of formation in water solution.

Speedups of that magnitude for quantum chemical calculations open new perspectives of studying larger structures which appear in inorganic chemistry (as zeolites and MOFs), biochemistry (as polysaccharides, small proteins, and DNA fragments), and material science (as nanotubes and fullerenes). In addition, we believe that this parallel (CPU−GPU) MOPAC code will make possible the use of semiempirical methods to produce long molecular simulations using both hybrid QM/MM and QM/QM potentials.

In future works, we plan to accelerate others parts of the MOPAC code, specifically the gradient computation and frequency calculations.

■ ASSOCIATED CONTENT

Ⓢ Supporting Information

More detailed profiling data, speedup graphs, descriptions, and geometries for some molecular systems. This information is available free of charge via the Internet at http://pubs.acs.org/.

■ AUTHOR INFORMATION

Corresponding Author

*Fax/Phone: +55-83-3216-7437. E-mail: gbr@quimica.ufpb.br.

Notes

The authors declare no competing financial interest.

†Part of the material in this article was presented at the XVI Brazilian Symposium on Theoretical Chemistry, XVI SBQT, Poster P377, Nov. 20−24, 2011, Brazil, http://www.sbqt.net/.

■ ACKNOWLEDGMENTS

We appreciate the financial support from CNPq (477906/2009-5, 506964/2010-8, 506027/2010-4, and 305251/2009-0), CAPES, and INCT-INAMI (Brazilian agencies). We also thank Prof. Alfredo Mayall Simas for helpful discussions.

■ REFERENCES

(1) De Jong, W. A.; Bylaska, E.; Govind, N.; Janssen, C. L.; Kowalski, K.; Müller, T.; Nielsen, I. M. B.; Dam, H. J. J.; Van; Veryazov, V.; Lindh, R. *Phys. Chem. Chem. Phys.* **2010**, *12*, 6896−6920.

(2) Seeger, R. *J. Comput. Chem.* **1981**, *2*, 168−176.

(3) *GPU Computing Gems Emerald*, 1st ed.; Hwu, W. W., Ed.; Elsevier Inc.: Burlington, MA, 2011.

(4) Stone, J. E.; Hardy, D. J.; Ufimtsev, I. S.; Schulten, K. *J. Mol. Graphics Modell.* **2010**, *29*, 116−125.

(5) Xu, D.; Williamson, M. J.; Walker, R. C. *Annu. Rep. Comput. Chem.* **2010**, *6*, 1−19.

(6) Anderson, J.; Lorenz, C.; Travesset, A. *J. Comput. Phys.* **2008**, *227*, 5342−5359.

(7) Friedrichs, M. S.; Eastman, P.; Vaidyanathan, V.; Houston, M.; Legrand, S.; Beberg, A. L.; Ensign, D. L.; Bruns, C. M.; Pande, V. S. *J. Comput. Chem.* **2009**, *30*, 864−872.

(8) Brown, P.; Woods, C. J.; McIntosh-Smith, S.; Manby, F. R. *J. Comput. Chem.* **2010**, *31*, 2008−2013.

(9) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2009**, *5*, 3138−3138.

(10) Luehr, N.; Ufimtsev, I. S.; Martínez, T. J. *J. Chem. Theory Comput.* **2011**, *7*, 949−954.

(11) Kindratenko, V.; Wilhelmson, R.; Brunner, R.; Martinez, T. J.; Hwu, W. M. *Comput. Sci. Eng.* **2010**, *12*, 12−16.

(12) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2009**, *5*, 2619−2628.

(13) Vogt, L.; Olivares-Amaya, R.; Kermes, S.; Shao, Y.; Amador-Bedolla, C.; Aspuru-Guzik, A. *J. Phys. Chem. A* **2008**, *112*, 2049−2057.

(14) DePrince, A. E.; Hammond, J. R. *J. Chem. Theory Comput.* **2011**, *7*, 1287−1295.

(15) Ma, W.; Krishnamoorthy, S.; Villa, O.; Kowalski, K. *J. Chem. Theory Comput.* **2011**, *7*, 1316−1327.

(16) Ye, D.; Titov, A.; Kindratenko, V.; Ufimtsev, I.; Martinez, T. Porting Optimized GPU Kernels to a Multi-core CPU: Computational Quantum Chemistry Application Example. In *Proceedings of the 2011 Symposium on Application Accelerators in High-Performance Computing*, Knoxville, TN, July 19−21, 2011; IEEE Press: New York, 2011; pp 72−75.

(17) Levine, B. G.; Stone, J. E.; Kohlmeyer, A. *J. Comput. Phys.* **2011**, *230*, 3556−3569.

(18) Uejima, Y.; Terashima, T.; Maezono, R. Y. O. *J. Comput. Chem.* **2011**, *32*, 2264−2272.

(19) Wilkinson, K. A.; Sherwood, P.; Guest, M. F.; Naidoo, K. J. *J. Comput. Chem.* **2011**, *32*, 2313−2318.

(20) Watson, M.; Olivares-Amaya, R.; Edgar, R. G.; Aspuru-Guzik, A. *Comput. Sci. Eng.* **2010**, *12*, 40−51.

(21) Gotz, A. W.; Wolfle, T.; Walker, R. C. *Annu. Rep. Comput. Chem.* **2010**, *6*, 21−35.

(22) Harvey, M. J.; De Fabritiis, G. *WIREs Comput Mol Sci.* **2012**, DOI: http://dx.doi.org/10.1002/wcms.1101.

(23) Cruz, F. A.; Layton, S. K.; Barba, L. A. *Comput. Phys. Commun.* **2011**, *182*, 2084−2098.

(24) Stewart, J. J. P. *MOPAC*; Stewart Computational Chemistry: Colorado Springs, CO, 2012. http://www.openmopac.net (accessed August 2012).

(25) Bakowies, D.; Thiel, W. *J. Am. Chem. Soc.* **1991**, *113*, 3704−3714.

(26) Green, D.; Boston, I.; Thiel, W. Parallelisation in quantum chemistry: the MNDO code. In *Lect. Notes Comput. Sc.*; Hertzberger, B., Serazzi, G., Eds.; Springer-Verlag: Berlin, Germany, 1995; Vol. *919*, pp 880−885.

(27) Thiel, W.; Green, D. G. In *Methods and Techniques in Computational Chemistry: METECC-95*; Clementi, E., Corongiu, G., Eds.; STEF: Cagliari, Italy, 1995; pp 141−168.

(28) Thiel, W. In *Modern Methods and Algorithms of Quantum Chemistry*; Grotendorst, J., Eds.; John von Neumann Institute for Computing, NIC Series: Jülich, Germany, 2000; Vol 3, pp 261−283.

(29) Baldridge, K. K. *J. Math. Chem.* **1996**, *19*, 87−109.

(30) Früchtl, H. A.; Nobes, R. H.; Bliznyuk, A. *THEOCHEM* **2000**, *506*, 87−97.

(31) Rendell, A. P.; Bliznyuk, A.; Huber, T.; Nobes, R. H.; Akhmatskaya, E. V.; Früchtl, H. A.; Kung, P. W.C.; Milman, V.; Lung, H. *Parallel Comput.* **2000**, *26*, 887−911.

(32) Berzigiyarov, P. K.; Zayets, V. A.; Ginzburg, I. Y.; Razumov, V. F.; Sheka, E. F. *Int. J. Quantum Chem.* **2002**, *88*, 449−462.

(33) Lin, T. H. *Parallelizing Legacy Applications Using Message Passing Programming Model and The Example of MOPAC*; Syracuse University: Syracuse, NY, 2000.

(34) Lin, T. H.; Tomasz, T. L.; Geo, H.; Fox, C. *Parallelizing MOPAC on Distributed Computing Systems within AVS Framework*; Syracuse University: Syracuse, NY, 2000.

(35) Mangueira, C. P., Jr.; Maia, J. D. C.; Cabral, L. A. F.; Rocha, G. B. Accelerating semiempirical quantum chemistry calculations using sparse linear algebra GPU libraries for large systems. Presented at XVI Simpósio Brasileiro de Química Teórica; Ouro Preto, Minas Gerais, Brazil, November 20−24, 2011. http://www.sbqt.net/resumos/412.pdf (accessed July 25, 2012).

(36) Wu, X.; Koslowski, A.; Thiel, W. *J. Chem. Theory Comput.* **2012**, *8*, 2272−2281.

(37) Zalesny, R.; Mezey, P. G.; Papadopoulos, M. G.; Leszczynski, J. *Linear-Scaling Techniques in Computational Chemistry and Physics*; Springer: Dordrecht, The Netherlands, 2011.

(38) Fedorov, D.; Kitaura, K. *The Fragment Molecular Orbital Method - Practical Applications to Large Molecular Systems*; CRC Press: Boca Raton, FL, 2009.

(39) Fukushima, K.; Wada, M.; Sakurai, M. *Proteins* **2008**, *71*, 1940−1954.

(40) Dewar, M. J. S.; Zoebisch, E. G.; Healy, E. F.; Stewart, J. J. P. *J. Am. Chem. Soc.* **1985**, *107*, 3902−3909.

(41) Stewart, J. J. P. *Int. J. Quantum Chem.* **1996**, *58*, 133−146.

(42) Anderson, E.; Bai, Z.; Bischof, C.; Blackford, L. S.; Demmel, J.; Dongarra, J.; Croz, J. Du; Greenbaum, A.; Hammarling, S.; McKenney, A.; Sorensen, D. *LAPACK Users' Guide*, 3rd ed.; SIAM: Philadelphia, PA, 1999.

(43) Stewart, J. J. P.; Császár, P.; Pulay, P. *J. Comput. Chem.* **1982**, *3*, 227−228.

(44) *HyperChem Professional*, version 8.0; Hypercube, Inc.: Gainesville, FL, 2012.

(45) Rocha, G. B.; Freire, R. O.; Simas, A. M.; Stewart, J. J. P. *J. Comput. Chem.* **2006**, *27*, 1101−1111.

(46) Allada, V.; Benjegerdes, T.; Bode, B. In *Proceedings of 2009 IEEE International Conference on Cluster Computing and Workshops*, New Orleans, Louisiana, USA, Aug 31−Sep. 4, 2009; IEEE: New York, 2009; pp 1−9.

(47) Olivares-Amaya, R.; Watson, M. A.; Edgar, R. G.; Vogt, L.; Shao, Y.; Aspuru-Guzik, A. *J. Chem. Theory Comput.* **2010**, *6*, 135−144.

(48) Tomov, S.; Nath, R.; Du, P.; Dongarra, J. *Matrix Algebra on GPU and Multicore Architectures (MAGMA), Version 1.0.0, Users' Guide*. Technical Report; Innovative Computing Laboratory, University of Tennessee, Knoxville: Knoxville, TN, 2011.

(49) Park, B.; Levitt, M. *J. Mol. Biol.* **1996**, *258*, 367−92.

(50) Lazaridis, T. *Curr. Opin. Struc. Biol.* **2000**, *10*, 139−145.

(51) Wollacott, A. M.; Merz, K. M. *J. Chem. Theory Comput.* **2007**, *3*, 1609−1619.

(52) Gogonea, V.; Merz, K. M. *J. Phys. Chem. A* **1999**, *103*, 5171−5188.
(53) Korth, M. *J. Chem. Theory Comput.* **2010**, *6*, 3808−3816.
(54) Klamt, A.; Schüürmann, G. *J. Chem. Soc., Perkin Trans. 2* **1993**, *5*, 799−805.
(55) McConkey, B. J.; Sobolev, V.; Edelman, M. *Proc. Natl. Acad. Sci. U.S.A.* **2003**, *100*, 3215−3220.
(56) Case, D. A.; Darden, T. A.; Cheatham, T. E.; Simmerling, C. L.; Wang, J.; Duke, R. E.; Luo, R.; Walker, R. C.; Zhang, W.; Merz, K. M.; Roberts, B.; Hayik, S.; Roitberg, A.; Seabra, G.; Swails, J.; Goetz, A. W.; Kolossvai, I.; Wong, K. F.; Paesani, F.; Vanicek, J.; Wolf, R. M.; Liu, J.; Wu, X.; Brozell, S. R.; Steinbrecher, T.; Gohlke, H.; Cai, Q.; Ye, X.; Wang, J.; Hsieh, M.-J.; Cui, G.; Roe, D. R.; Mathews, D. H.; Seetin, M. G.; Salomon-Ferrer, R.; Sagui, C.; Babin, V.; Luchko, T.; Gusarov, S.; Kovalenko, A.; Kollman, P. A. *Amber 11*; University of California, San Francisco: San Francisco, CA, 2010.
(57) Hornak, V.; Abel, R.; Okur, A.; Strockbine, B.; Roitberg, A.; Simmerling, C. *Proteins* **2006**, *65*, 712−725.