JCTC Journal of Chemical Theory and Computation

Article

pubs.acs.org/JCTC

Semiempirical Quantum Chemical Calculations Accelerated on a Hybrid Multicore CPU−GPU Computing Platform

Xin Wu, Axel Koslowski, and Walter Thiel*

Max-Planck-Institut für Kohlenforschung, Kaiser-Wilheim-Platz 1, 45470 Mülheim an der Ruhr, Germany

Supporting Information

ABSTRACT: In this work, we demonstrate that semiempirical quantum chemical calculations can be accelerated significantly by leveraging the graphics processing unit (GPU) as a coprocessor on a hybrid multicore CPU−GPU computing platform. Semiempirical calculations using the MNDO, AM1, PM3, OM1, OM2, and OM3 model Hamiltonians were systematically profiled for three types of test systems (fullerenes, water clusters, and solvated crambin) to identify the most time-consuming sections of the code. The corresponding routines were ported to the GPU and optimized employing both existing library functions and a GPU kernel that carries out a sequence of noniterative Jacobi transformations during pseudodiagonalization. The overall computation times for single-point energy calculations and geometry optimizations of large molecules were reduced by one order of magnitude for all methods, as compared to runs on a single CPU core.

1. INTRODUCTION

Semiempirical molecular orbital (MO) methods are widely used in quantum chemical studies of large molecules.[1−8] They provide qualitative insights into chemical problems at low cost, because they are much faster than the more accurate ab initio and density functional methods, typically by at least 3 orders of magnitude. This efficiency makes them especially useful for initial explorations before employing more expensive calculations, for correlating large sets of experimental and theoretical data to establish trends, and for studying the properties and dynamical behavior of complex systems.[1,2]

Semiempirical and ab initio MO approaches share the same conceptual framework, but a number of drastic approximations are introduced at the semiempirical level to speed up the calculations. The most important approximation is that many of the smaller one- and two-electron integrals are neglected, and the remaining ones are either determined directly from experiment or calculated from suitable parametric functions.[4,9] As a consequence, integral evaluation in semiempirical calculations formally scales as $O(N^2)$ for N basis functions, in contrast to $O(N^4)$ in generic ab initio MO calculations. The dominant computational effort is therefore shifted from integral evaluation to linear algebra operations, for example, matrix multiplication and matrix diagonalization, which formally scale as $O(N^3)$.

To further extend the scope of the current quantum chemical work, it is essential to take advantage of new powerful computer architectures. The traditional hardware is based on the general-purpose central processing unit (CPU), which is capable of executing multiple threads simultaneously due to various multicore designs. On the other hand, the graphics processing unit (GPU) is a many-core architecture originally designed for the rapid processing of images. To the present day, GPUs have evolved to being able to perform on the order of 10^{12} floating-point operations per second (FLOP/s) using thousands of threads and high memory bandwidth.[10] These advances have made the GPU become particularly suitable for highly parallel arithmetic-intensive computations, in particular also in the field of molecular modeling (for reviews, see refs 11−13). Recently, many authors have implemented GPU-oriented algorithms in their quantum chemistry codes and reported considerable speedups, up to about 100 times as compared to CPU-only implementations.[14−20] At the semiempirical level, we are only aware of some unpublished work in this area.[21]

In this article, we describe a comprehensive optimization of the self-consistent-field (SCF) code in our semiempirical quantum chemistry program MNDO[22] on a hybrid multicore CPU−GPU computing platform. Six semiempirical methods, MNDO (Modified Neglect of Differential Overlap),[23] AM1 (Austin Model 1),[24] PM3 (Parameterized Model 3),[25] and OMx (Orthogonalization-corrected Model x, x = 1, 2, and 3),[26] are considered. We first report profiles of our original code on a single CPU core using three representative types of test systems, fullerenes,[27,28] water clusters,[29] and crambin[15] solvated in water spheres of increasing size. The most time-consuming routines identified in this manner are ported to the GPU by utilizing both vendor-optimized linear algebra library functions[30,31] as well as a manually tuned GPU kernel. Finally, the performance of the resulting code is checked through single-point energy evaluations and geometry optimizations for all test cases to establish the speedups that can be achieved for semiempirical calculations on a hybrid multicore CPU−GPU platform.

2. COMPUTATIONAL DETAILS

This work covers the semiempirical MNDO,[23] AM1,[24] PM3,[25] and OMx (x = 1, 2, and 3)[26] methods, which are all based on the NDDO (Neglect of Diatomic Differential Overlap) integral approximation.[32] The standard MNDO-type methods (MNDO, AM1, and PM3) have served for decades as widely

Received: March 2, 2012
Published: May 25, 2012

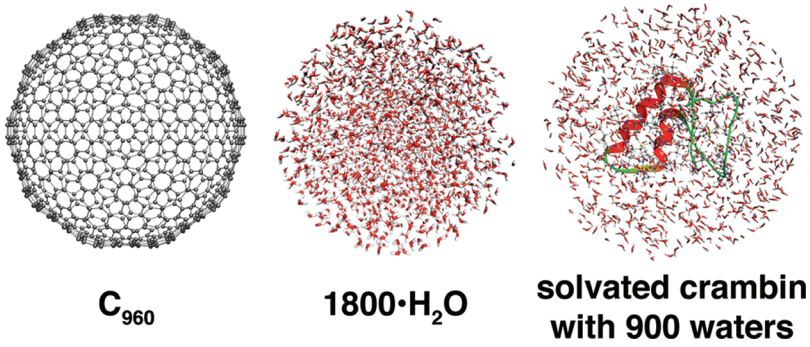


Figure 1. The largest system in each test set.

used computational tools. The more recently developed OM*x* methods[26] add orthogonalization corrections to the MNDO model and offer significant improvements in systematic benchmarks for ground-state molecules[33,34] and especially for excited-state properties.[35] The major computational burdens during SCF calculations are expected to be similar for all of these methods. We employ three sets of test molecules, fullerenes (C*i*), water clusters (W*j*), and the protein crambin solvated in water balls (P*k*).[36] These systems represent typical application areas of semiempirical methods, ranging from regularly shaped highly symmetric molecules to molecular assemblies with many degrees of freedom and complex biochemical systems.

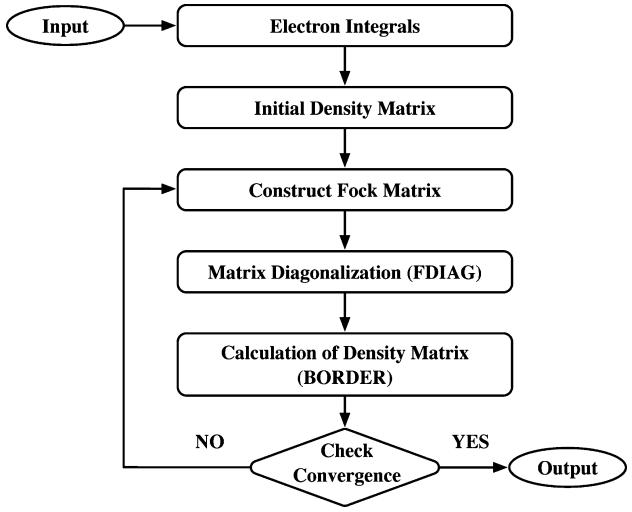
All present calculations have been carried out on a workstation featuring one Intel Xeon X5670 processor (6 cores @ 2.93 GHz),[37] 12 GB of main memory, and one NVidia Tesla C2070 GPU (448 cores @ 1.15 GHz) with 6 GB of global memory. Figure 1 shows the largest test systems in each set: C_{960}, $1800{\cdot}H_2O$, and crambin$\cdot 900H_2O$. The latter two are close to the size that can be handled in the current setup with the memory available in the workstation.[38] All floating point arithmetic operations were performed in double precision to avoid numerical inaccuracies in the computation of the large molecules considered (>1000 heavy atoms). The SCF convergence criteria were 1.0×10^{-6} eV for the electronic energy and 1.0×10^{-6} for the maximum change in the density matrix. When using the DIIS (direct inversion of iterative subspace)[39] procedure in the SCF iterations, the maximum error matrix element was required to be less than 1.0×10^{-6} eV. The norm of the gradient vector ($\|g\| < 1.0$ kcal/(mol·Å)) served as the convergence criterion in geometry optimizations.

The FORTRAN compiler from the development suite "Intel Composer XE 12.0" and the NVidia CUDA (Compute Unified Device Architecture) Toolkit 3.2 were used during the development of the CPU and GPU code in a CVS version of MNDO99.[22] In the CPU-only implementation, the Intel Math Kernel Library (MKL) was dynamically linked to provide the required BLAS (Basic Linear Algebra Subprograms) and LAPACK (Linear Algebra Package) functions. Alternatively, on the hybrid multicore CPU−GPU platform, the corresponding GPU-accelerated routines were taken from the libraries CUBLAS[30] and MAGMA.[31] On our workstation, the CPU−GPU communication is limited by a 16-lane PCIe (Peripheral Component Interconnect Express) connection with a theoretical bandwidth of 8.0 GB/s. The actually available bandwidth was observed to be greater than 4.0 GB/s in our tests. Having sufficient bandwidth is essential because all data to be processed by the GPU and the results obtained must be transferred over this connection. In our implementation, the communication overhead was carefully minimized such that the data transfer only needs a very small portion of the total computation time, for example, ∼0.5% for the MNDO calculation of $1800{\cdot}H_2O$ on the hybrid CPU−GPU platform.

3. SEMIEMPIRICAL CALCULATIONS ON THE CPU-ONLY PLATFORM

A simplified workflow of a semiempirical single-point SCF calculation is sketched in Chart 1. The tasks in this workflow

Chart 1. Workflow of a Single-Point Semiempirical SCF Calculation



are generic and will appear in virtually any semiempirical SCF program. The profiling results reported below for these tasks are thus of general relevance for semiempirical code development.

Because of the NDDO approximation, the evaluation of the two-electron integrals formally requires only $O(N^2)$ operations. Therefore, the diagonalization of the Fock matrix (FDIAG) and the computation of the density matrix (BORDER) are the $O(N^3)$ steps that dominate the calculations in practice.[40] Two additional optional procedures, DIIS[39] and the pseudodiagonalization (PDIAG),[41] may also consume a significant part of the computation time and hence deserve special attention. The DIIS scheme makes use of the results from previous SCF iterations to construct a new Fock matrix such that SCF convergence is accelerated; this requires the computation of an error matrix that scales as $O(N^3)$. The full diagonalization (FDIAG) can often be replaced by a less expensive

dx.doi.org/10.1021/ct3001798 | J. Chem. Theory Comput. 2012, 8, 2272−2281

pseudodiagonalization (PDIAG), which involves a triple matrix product (MMM) and noniterative Jacobi transformations (JACOBI); normally, PDIAG can be applied throughout the SCF process except for the first and the last cycles.

We have used the existing and already sufficiently optimized CPU code to profile the semiempirical calculations for all test molecules on a single CPU. Two typical computational protocols were chosen: (A) pseudodiagonalization without DIIS, and (B) full diagonalization with DIIS. In the case of the MNDO-type methods, the fullerenes and the water clusters were run using (A), and the crambin set was computed using (B). All MNDO-type methods behave similarly in this analysis, so we show only the profiles for AM1 in Figure 2a. For the

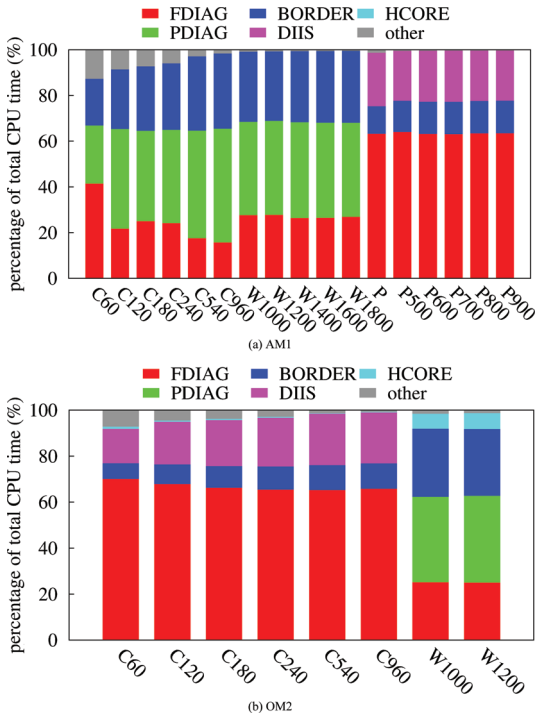


Figure 2. Profiles of the AM1 and OM2 calculations of the test molecules on a single CPU core.

OM*x* methods, we only examined the fullerenes (B) and the water clusters W1000 and W1200 (A); crambin could not be handled because of the missing sulfur parameters, while for the larger water clusters the memory capacity was not sufficient to process the orthogonalization corrections with the existing setup. We present only the OM2 profiles in Figure 2b because all three OM*x* methods show similar behavior (except for the integral routines because there are no orthogonalization corrections to the two-center terms in OM1). The profiling results for the other methods (MNDO, PM3, OM1, and OM3) are documented in the Supporting Information.

Table 1 reports the computation times as well as the number of SCF iterations and full diagonalizations during the SCF procedure for all six semiempirical methods, taking the water cluster W1000 ($1000{\cdot}H_2O$) as a typical example.[42] It is obvious that the number of SCF iterations (~20) and the computation times per SCF cycle are roughly in the same range for all methods; none of them is much more expensive or much cheaper than the others. AM1 and PM3 are slightly more costly than the original MNDO method, presumably at least partly due to the more complex core-repulsion function.[24,25] Despite the orthogonalization corrections to the one-center terms, OM1 turns out to be the fastest of all six methods. OM2 and OM3 are somewhat slower than OM1 because of the additional three-center terms arising from the orthogonalization corrections to the two-center resonance integrals.[26]

The MNDO-type calculations (Figure 2a) are completely dominated by the four routines FDIAG, PDIAG, BORDER, and DIIS (see above). The percentage of the CPU time consumed by other tasks (i.e., mainly integral evaluation and Fock matrix construction) decreases dramatically with increasing system size, for example, from 12.1% to 1.6% in the MNDO calculations on C_{60} and C_{960}, respectively. In small molecules, the other tasks may require ~10% of the time, but the total computation time is negligible in these cases; for example, the MNDO calculations on C_{60} and C_{120} just take 0.1 and 1.0 s, respectively. In larger molecules with several hundred atoms, the four dominant routines together usually account for >98% of the total time of an MNDO-type calculation. The situation is slightly different for the OM*x* methods because of the additional orthogonalization corrections to the one-electron core Hamiltonian (HCORE). The share of HCORE becomes more pronounced especially in OM2 and OM3 due to the inclusion of three-center terms in the orthogonalization corrections. For the water cluster W1000, for instance, the fraction of the computer time for HCORE amounts to 6.5% in OM2 and 6.2% in OM3. Nevertheless, FDIAG, PDIAG, BORDER, and DIIS remain the four most CPU-intensive routines also in these cases and together consume more than 90% of the total computation time.

By default, PDIAG rather than FDIAG is commonly applied whenever possible. According to the profiling, PDIAG will then normally contribute more than one-third to the total CPU time, which calls for a more detailed analysis of this routine. The breakdown of the computational effort in PDIAG is shown in Figure 3 for MNDO calculations on fullerenes and water clusters that use the default threshold for deciding whether the Jacobi transformation is done for a given pair of eigenvectors.[41] The triple matrix product MMM takes 50−60% of the CPU time within PDIAG in the case of the water clusters, and

Table 1. Computation Times (in seconds) of MNDO, AM1, PM3, OM1, OM2, and OM3 Single-Point Energy Calculations of $1000{\cdot}H_2O$ on a Single CPU Core[a]

	MNDO-based methods			OM*x* methods		
	MNDO	AM1	PM3	OM1	OM2	OM3
N_{SCF}	18	19	23	18	20	22
N_{diag}	4	4	4	4	4	4
time	1433.17	1595.66	2045.83	1316.51	1740.69	1818.91
time/N_{SCF}	79.62	83.98	88.95	73.14	87.03	82.68

[a]The number of SCF iterations and full diagonalizations is denoted by N_{SCF} and N_{diag}, respectively.

dx.doi.org/10.1021/ct3001798 | J. Chem. Theory Comput. 2012, 8, 2272−2281

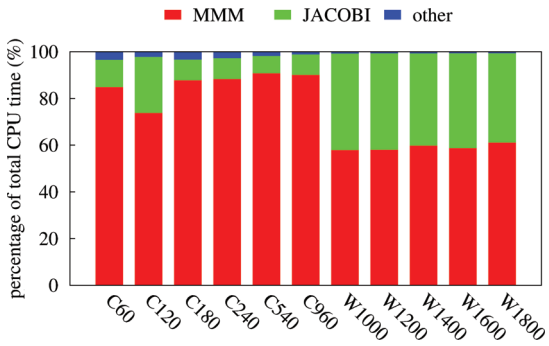


Figure 3. Profiles of PDIAG in MNDO calculations of fullerenes and water clusters on a single CPU core.

typically more than 80% in the case of the fullerenes, while most of the rest is consumed by the noniterative Jacobi transformations. The difference between the fullerenes and the water clusters arises from the thresholding: in the latter case, more Jacobi rotations are found to be needed, which leads to a significant increase in the computational effort (see Figure 3).

To summarize, the profiling has identified four bottlenecks that together consume more than 90% of the CPU time in semiempirical calculations on large molecules. The GPU adaptation of the code will focus exclusively on the corresponding four routines FDIAG, PDIAG, BORDER, and DIIS.

4. PROFILE-GUIDED OPTIMIZATION ON THE HYBRID PLATFORM

4.1. Survey of Computational Tasks. To establish notation, we briefly summarize the main computational tasks. FDIAG solves the eigenvalue equation:

$$\mathbf{FC} = \mathbf{CE} \tag{1}$$

where $\mathbf{F}$ denotes the Fock matrix, $\mathbf{C}$ is the matrix of MO coefficients, and $\mathbf{E}$ is a diagonal matrix containing the MO energies. The triple matrix multiplication in PDIAG is

$$\mathbf{F}_{\mathrm{MO}} = \mathbf{C}_{\mathrm{o}}^{\mathrm{T}}\mathbf{F}\mathbf{C}_{\mathrm{v}} \tag{2}$$

where $\mathbf{F}_{\mathrm{MO}}$ is the occupied-virtual block of the Fock matrix. $\mathbf{C}_{\mathrm{o}}$ and $\mathbf{C}_{\mathrm{v}}$ are the occupied and virtual vectors, respectively. BORDER performs one dominant matrix multiplication (>99% CPU time, eq 3) to determine the density matrix $\mathbf{P}$.

$$\mathbf{P} = 2\mathbf{C}_{\mathrm{o}}\mathbf{C}_{\mathrm{o}}^{\mathrm{T}} \tag{3}$$

DIIS evaluates the error matrix, which requires matrix multiplications (eq 4) that are computationally much more demanding (~98% CPU time) than the subsequent matrix−vector multiplications (eqs 5 and 6).

$$\mathbf{e}_k = \mathbf{FP} - \mathbf{PF} \tag{4}$$

$$\mathbf{b}_k = \tilde{\mathbf{E}}^{\mathrm{T}}\mathbf{e}_k \tag{5}$$

$$\mathbf{F}' = \tilde{\mathbf{F}}\mathbf{x} \tag{6}$$

$\mathbf{e}_k$ is the error matrix of the kth DIIS step, which is stored in linearly packed form as the kth column of the composite matrix $\tilde{\mathbf{E}}$. $\mathbf{b}_k$ collects the scalar products of error matrices evaluated in the kth DIIS step. The new Fock matrix $\mathbf{F}'$ is obtained as a linear combination of previous Fock matrices stored in composite form ($\tilde{\mathbf{F}}$), using the coefficient vector $\mathbf{x}$ determined in the DIIS procedure.[39]

4.2. GPU-Accelerated Library Functions. As documented above, matrix diagonalization, matrix−matrix multiplication, and matrix−vector multiplication are the most important linear algebra operations in semiempirical SCF calculations. To identify the routines in the MKL, CUBLAS, and MAGMA libraries best suited for our purposes, we have performed test calculations. In the following, we briefly summarize the results of these tests (for detailed data, see the Supporting Information).

Matrix Diagonalization. We have applied four different LAPACK routines (SYEV, SYEVX, SYEVD, and SYEVR)[43] for Fock matrix diagonalization (eq 1). The MKL library contains an implementation of all four routines, and multithreaded versions are available for use with symmetric multiprocessing systems. MAGMA offers SYEVD only, taking advantage of the hybrid CPU−GPU architecture: one of the steps in the algorithm is executed on the CPU and the other two on the GPU.[44]

The tests on one single CPU core indicate that SYEV is always slowest while SYEVR is a little faster than SYEVD. All CPU-only calculations are accelerated when run in parallel on 6 CPU cores, but the speedups are moderate (at most by a factor of 3 for large matrices) because these routines are bandwidth-bound; that is, the performance of the algorithm is limited by the memory bandwidth the hardware can provide. On the hybrid CPU−GPU platform, there are larger gains in performance when all six CPU cores are used in combination with the GPU: the hybrid SYEVD (H6CG) routine is always the fastest one for any matrix size, with a speedup over SYEVD (1C) of about 5-fold. Because SYEVR is not available in MAGMA, SYEVD was used for Fock matrix diagonalization in the following semiempirical calculations to allow for a fair comparison between CPU-only and hybrid CPU−GPU computing platforms.

Matrix−Matrix Multiplication. Depending on the type of the matrices involved, multiplications can be carried out with the BLAS routines GEMM, SYMM, or SYRK. GEMM performs the generic matrix−matrix multiplication, whereas SYMM is more specific because at least one of the matrices is required to be symmetric. SYRK can be applied if the product matrix is known to be symmetric so that only the upper or lower triangular entries need to be computed, which can lead to a speedup of almost 2 by skipping almost one-half of the floating-point operations as compared to the general case. SYRK will thus be an attractive choice for computing the density matrix, which is symmetric by definition (eq 3).

To assess the performance of GEMM, SYMM, and SYRK for our purposes, we evaluated the product $\mathbf{P} = \mathbf{A}\mathbf{A}^{\mathrm{T}}$, where $\mathbf{A}$ is a random real symmetric matrix. On the CPU-only platform, the parallel runs with 6 cores show a speedup of almost 6 over those on one core (6C vs 1C). On the hybrid CPU−GPU platform, GEMM and SYRK are much faster still, with a peak performance close to 300 GFLOP/s. Hence, the SYRK routine was selected for computing the density matrix in BORDER (eq 3), while GEMM was chosen for carrying out the other matrix−matrix multiplications in PDIAG (eq 2) and in DIIS (eq 4). The use of these GPU-accelerated routines on the hybrid platform offers a ~25× boost in performance as compared to a single CPU core.

Matrix−Vector Multiplication. Matrix−vector multiplications are required in DIIS to compute $\mathbf{b}_k$ (eq 5, operation $\mathbf{p} = \mathbf{M}^{\mathrm{T}}\mathbf{v}$) and $\mathbf{F}'$ (eq 6, operation $\mathbf{p} = \mathbf{M}\mathbf{v}$). These are rather special cases because the matrices in these products have a very large

dx.doi.org/10.1021/ct3001798 | J. Chem. Theory Comput. 2012, 8, 2272−2281

number of rows (of the order of 10^7 for the biggest molecules in this work) and only a very small number of columns (of the order of 10). Three different library routines are available: GEMV is the generic routine for matrix−vector multiplications and can be applied in both situations; DOTL and AXPYL are alternatives. DOTL loops over the dot product (DOT) of each row in $\mathbf{M}$ and $\mathbf{v}$ to produce $\mathbf{p} = \mathbf{M}^T\mathbf{v}$, whereas AXPYL sequentially sweeps each column in $\mathbf{M}$ with $\mathbf{v}$ and assembles the results for $\mathbf{Mv}$ in $\mathbf{p}$. In DIIS, the use of DOTL and AXPYL ensures coalesced memory access in the computation of $\mathbf{b}_k$ and $\mathbf{F}'$, respectively, because the two-dimensional arrays $\tilde{\mathbf{E}}$ and $\tilde{\mathbf{F}}$ are stored in column major order. The row dimension of these arrays is given by the number of iterations in the DIIS procedure and will thus be much smaller than their column dimension that is equal to the number of elements in the linearly packed Fock matrix.

Because of the inherently low ratio of compute-to-memory accesses, the routines GEMV, DOTL, and AXPYL are all bandwidth-bound, and using more CPU cores thus hardly helps to increase the speed. The DOTL function from the GPU-based CUBLAS library is best suited for $\mathbf{p} = \mathbf{M}^T\mathbf{v}$, being up to about 5 times faster than the CPU-based counterpart. GEMV is always superior to AXPYL for $\mathbf{p} = \mathbf{Mv}$ on the same platform: in this case, the GPU-based GEMV routine can be roughly 10 times faster than the CPU-based version. In view of these comparisons, DOTL and GEMV were chosen for the computation of $\mathbf{b}_k$ and $\mathbf{F}'$ on the hybrid computing platform, respectively.

4.3. Jacobi Transformation on GPU. There is no ready-to-use library function available for the noniterative Jacobi transformation in PDIAG. The underlying algorithm[41] involves several approximations, including the neglect of changes in the matrix elements of the secular determinant upon rotation. This substantially simplifies parallelization of the algorithm. For further background information, the interested reader is referred to other recent work related to implementing Jacobi rotations on GPUs[45] and to earlier work concerning parallel Jacobi rotations on CPUs.[46−48] In this section, we first describe the basic algorithms considered in our implementation and then discuss the results.

The original CPU algorithm[41] in PDIAG performs sweeps of independent 2 × 2 rotations of one occupied and one virtual molecular orbital trial vector from the last SCF iteration, C_i and C_a:

$$C_i' = cC_i - sC_a;\ C_a' = sC_i + cC_a \tag{7}$$

where c and s denote the rotation coefficients, and the new MO vectors are indicated by primes. The rotation coefficients are calculated from the approximate MO energies ε_i and ε_a of the previous SCF iteration and from the matrix element $\mathcal{F}_{ia}$ of the Fock matrix in the MO basis connecting both orbitals.

$$c = 1 - \frac{u}{2};\ s = \pm\sqrt{u - \frac{u^2}{4}};\ u = \left(\frac{\mathcal{F}_{ia}}{\varepsilon_a - \varepsilon_i}\right)^2 \tag{8}$$

The sign of s is opposite of that of the Fock matrix element.

During one sweep, all rotations on a set of independent occupied-virtual orbital pairs are performed that satisfy a predefined threshold criterion.[41] If, respectively, n_o and n_v are the number of occupied and virtual molecular orbitals, and n_{small} and n_{large} are the smaller and the larger number of n_o and n_v, there are n_{large} sweeps and n_{small} orbital pairs examined per sweep.

The rotation of one pair of vectors in eq 7 can be carried out by the BLAS level 1 routine ROT. In semiempirical calculations, the number of basis functions N is usually less than 10^4. In this range, the ROT (G) routine from the GPU-based CUBLAS library is not yet efficient, due to the low ratio (<3) of compute-to-memory accesses, and is outperformed by the single-core CPU variant ROT (1C); this situation is reversed only for huge values of N (>10^6) that are not reached in practice. Semiempirical calculations will thus not benefit from the naive use of ROT (G) for the Jacobi transformation in PDIAG. This calls for the development of a dedicated version of PDIAG that exploits the potential of the GPU for speeding up this transformation, which may represent a significant part of the overall computational effort (see above).

Four algorithms were devised for the Jacobi transformation on the GPU whose architecture is designed for executing large numbers of parallel threads. The algorithms differ in the way how the work is distributed among the threads, how these are organized in blocks, how the threads are synchronized, and, most importantly, where the decision is made as to which orbital pairs to rotate. A detailed description of all algorithms is given in the Supporting Information.

Algorithms I−III have in common that the decision whether to rotate an orbital pair is made in the threads just before the rotation would be performed. These algorithms come in three variants depending on the treatment of the rotation coefficients in eq 7 that may be recomputed as needed (first variant) or precomputed by a separate routine and stored in global memory. For efficient access to these quantities in global memory, the kernels may rely on the caching mechanisms of the GPU (second variant) or may explicitly buffer them using shared memory, which is much faster than global memory (third variant). These combinations give rise to nine GPU kernels altogether.

For these algorithms, there is the possibility that the threads that skip a rotation just remain idle waiting for other threads to complete a rotation. This will reduce the workload on the GPU, which will ultimately lead to lower performance. Therefore, another algorithm IV was designed in which the decision on which orbital pairs to rotate is made in a separate initial step: the rotation coefficients are precomputed, and only those are stored that correspond to rotations actually to be performed later. This complicates the logistics of precomputing the rotation coefficients in the parallel environment of the GPU. The implementation of the actual rotations in algorithm IV was based on our experience gained with the other algorithms.

For all algorithms and their variants, we tested a large number of different execution configurations, that is, the dimensions of the thread blocks, a parameter specific to GPU programming. The detailed results for the fastest algorithm (algorithm IV) are documented in the Supporting Information. In the following, we only summarize the essential findings.

We find in our extensive search that algorithm III with cached coefficients is the fastest among the first three algorithms. For C_{960}, it is about twice as fast as the second-ranked one (algorithm I with shared memory). Therefore, the rotations needed in algorithm IV were carried out in analogy to algorithm III. Systematic tests show that the time for precomputing and storing the rotation coefficients in algorithm IV is negligible and that statistically the configuration 64 × 16 performs best on average (see the Supporting Information). We have therefore adopted this configuration as our default choice. With this setup, algorithm IV consistently outperforms the

other three algorithms by a large margin. For example, in the case of C_{960}, where the benefits from skipping rotations are largest, algorithm IV is more than 5 times faster than any of the three competitors.

Table 2 shows the execution times and speedups of the Jacobi transformation for selected test molecules, on a single CPU (1C) and on the GPU (algorithm IV). The acceleration of the Jacobi transformation on the GPU is expected to be influenced by the size and symmetry of the molecule. Because the architecture of the GPU is designed for massively parallel computations, the benefits for larger molecules should be higher. This is borne out by the data for $1000{\cdot}H_2O$ and crambin: in these two unsymmetrical molecules, the percentage (R_{rot}) of required Jacobi rotations is similar (13% vs 11%), and hence the speedup is higher in the case of the larger system (6.6 vs 4.6). In the case of a highly symmetric molecule, many Fock matrix elements between occupied and virtual MOs will be zero by symmetry, which will lower R_{rot}, for example, to 2% in the icosahedral C_{960} molecule; algorithm IV is designed to exploit this reduced workload and thus leads to a maximum speedup of 7.1 relative to the time on a single CPU.

Table 2. Computation Times (in seconds) for the Jacobi Transformation in MNDO Calculations on a Single CPU (1C) and on the GPU[a]

	C_{960} (N = 3840)		$1000{\cdot}H_2O$ (N = 6000)		crambin (N = 1623)	
	time	speedup	time	speedup	time	speedup
1C	26.96		241.34		12.39	
GPU	3.82	7.1	36.52	6.6	2.71	4.6

[a] N denotes the number of basis functions.

5. SEMIEMPIRICAL CALCULATIONS ON THE CPU–GPU PLATFORM

5.1. Performance. The workstation used in this study provides several kinds of computing environments: one single CPU core (1C), multiple CPU cores (2C, 6C), one single CPU core with one GPU (H1CG), and all CPU cores plus one GPU (H6CG). Our code also allows for OpenMP parallel execution of the most demanding tasks not yet ported to the GPU, in particular, integral evaluation and Fock matrix formation. To reach the best-effort performance on the currently used hybrid CPU–GPU platform, OpenMP parallelization was turned on using all 6 CPU cores in the calculations denoted by H6CG*. In this section, we report computation times and speedups obtained in semiempirical calculations on the test molecules in all of these environments, both for MNDO-type and for OMx methods.

The full list of performance data is given in the Supporting Information. As a representative example, the results for $1000{\cdot}H_2O$ are summarized in Table 3. For a given environment, the computation times are quite similar for the six semiempirical methods considered presently, especially when taking the slight differences in the number of SCF iterations into account. For instance, the computation times per SCF cycle range between 73.14 and 88.95 s in the slowest environment (1C) and between 7.43 and 9.97 s in the fastest one (H6CG*). The minimum and maximum times for the full SCF calculation differ at most by factors of 1.55 (1C) and 1.72

Table 3. Computation Times (in seconds) of the MNDO, AM1, PM3, OM1, OM2, and OM3 Single-Point Energy Evaluations for $1000{\cdot}H_2O$ on CPU-Only (nC) and Hybrid CPU–GPU Platforms (HnCG)[a]

	1C	2C	6C	H1CG	H6CG	H6CG*
MNDO: N_{SCF} = 18, N_{diag} = 4						
time	1433.17	858.18	491.57	217.84	160.60	154.59
time/N_{SCF}	79.62	47.68	27.31	12.10	8.92	8.59
speedup		1.7	2.9	6.6	8.9	9.3
AM1: N_{SCF} = 19, N_{diag} = 4						
time	1595.66	993.67	612.29	241.02	182.97	176.74
time/N_{SCF}	83.98	52.30	32.23	12.69	9.63	9.30
speedup		1.6	2.6	6.6	8.7	9.0
PM3: N_{SCF} = 23, N_{diag} = 4						
time	2045.83	1345.98	902.66	291.88	236.51	229.36
time/N_{SCF}	88.95	58.52	39.25	12.69	10.28	9.97
speedup		1.5	2.3	7.0	8.6	8.9
OM1: N_{SCF} = 18, N_{diag} = 4						
time	1316.51	745.44	379.78	208.55	151.75	133.68
time/N_{SCF}	73.14	41.41	21.10	11.59	8.43	7.43
speedup		1.8	3.5	6.3	8.7	9.8
OM2: N_{SCF} = 20, N_{diag} = 4						
time	1740.69	1115.49	720.15	341.31	284.65	190.67
time/N_{SCF}	87.03	55.77	36.01	17.07	14.23	9.53
speedup		1.6	2.4	5.1	6.1	9.1
OM3: N_{SCF} = 22, N_{diag} = 4						
time	1818.91	1148.49	720.07	342.92	287.18	190.59
time/N_{SCF}	82.68	52.20	32.73	15.59	13.05	8.66
speedup		1.6	2.5	5.3	6.3	9.5

[a] n is the number of CPU cores in use. The number of SCF iterations and full diagonalizations is denoted by N_{SCF} and N_{diag}, respectively. OpenMP parallelization was turned on using 6 CPU cores in H6CG* (our best effort).

(H6CG*), indicating again a similar computational burden for the different semiempirical methods in these environments.

In the CPU-only case, the speedups of the multicore calculations do not increase in proportion to the number of cores, reaching factors of about 1.5 on two cores (2C) and only around 3 on six cores (6C) regardless of the type of test molecule (see Table 3 and the Supporting Information). This is because the computation is bandwidth-bound on the present hardware for two of the major bottlenecks, FDIAG and the JACOBI part of PDIAG; adding more processing units hardly helps under these circumstances. It should be noted in this context that this limitation arises from the chosen hardware and not from the software: early tests of the MNDO program on a Cray Y-MP8 computer with eight CPU cores and comparatively fast shared memory gave speedups up to 7.7 in the MNDO geometry optimization of the fullerene C_{540}.[27] Repeating this calculation with identical input options on the current hardware (and with OpenMP parallelization being turned on) yields a smaller speedup of 4.0 on six CPU cores, because the higher clock frequency of the current processor (3 GHz vs 80 MHz on the Cray Y-MP) is not matched by a corresponding increase in memory bandwidth. The MNDO code will thus offer higher speedups in multicore CPU-only calculations on large molecules when the hardware performance is compute-bound (and not bandwidth-bound as in the present case).

For the chosen representative example of 1000·H_2O, the hybrid CPU–GPU platform with only one CPU core (H1CG) offers speedups of 5.1–7.0 relative to the CPU-only case (1C) due to the higher FLOP rate and bandwidth of the GPU (Table 3). H1CG outperforms the best CPU-only platform (6C) by a wide margin (speedup factors of 2.3–3.5). Adding more CPU cores to H1CG further accelerates the SYEVD routine (see Figure 3 in the Supporting Information) and leads to speedups of 6.1–8.9 for H6CG relative to 1C. Turning on OpenMP parallelization (H6CG*) for the other parts of the code leads to further (rather small) gains in the case of the MNDO-type calculations where integral evaluation is fast anyway, but to substantial accelerations especially for OM2 and OM3 where the computation of the three-center orthogonalization corrections strongly benefits from this kind of parallelization. As a consequence, HC6G* provides an excellent overall performance for all semiempirical methods considered presently, with speedup factors for 1000·H_2O of 8.9–9.8 (Table 3).

Average speedups (H6CG and H6CG* vs 1C) for large molecules (N > 1000) are summarized in Table 4. Those for the MNDO-type methods are quite uniform because of the similar underlying theoretical framework, both for H6CG (8.1–8.3) and for H6CG* (8.4–8.7). In the case of the OMx methods, the average speedups for OM2 and OM3 are again relatively low (5.6–5.7) because the orthogonalization corrections are computed using single-threaded CPU code, which can consume up to 7% of the computation time in the CPU-only case (Figure 2). After turning on OpenMP parallelization (H6CG*), the average speedups for the OMx methods increase to a rather uniform level (7.3–7.7) but remain slightly below those for the MNDO-type methods.

Figures 4–6 show the speedups obtained in the MNDO, AM1, PM3, and OMx calculations for all test molecules on the H6CG and H6CG* platforms as compared to the single CPU case (1C); the corresponding numerical results are available in the Supporting Information.

Table 4. Average Speedups of Semiempirical Single-Point Energy Evaluations on the Hybrid Multicore CPU–GPU Platform (H6CG and H6CG*) Relative to the Single-CPU Case (1C), for Large Molecules with More than 1000 Basis Functions[a]

	MNDO-type methods			OMx methods		
	MNDO	AM1	PM3	OM1	OM2	OM3
H6CG	8.3	8.2	8.1	7.1	5.6	5.7
H6CG*	8.7	8.6	8.4	7.7	7.3	7.5

[a]OpenMP parallelization was turned on using 6 CPU cores in H6CG* (our best effort).

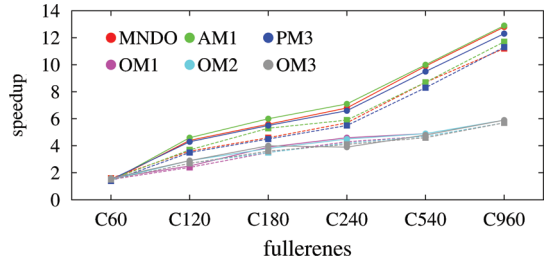


Figure 4. Speedup of the semiempirical single-point energy calculations of fullerenes on the hybrid multicore CPU–GPU platform (H6CG, dashed lines with squares; and H6CG*, solid lines with circles) with respect to one single CPU core (1C). The MNDO-type and OMx calculations employ pseudodiagonalization (without DIIS) and full diagonalization (with DIIS), respectively.

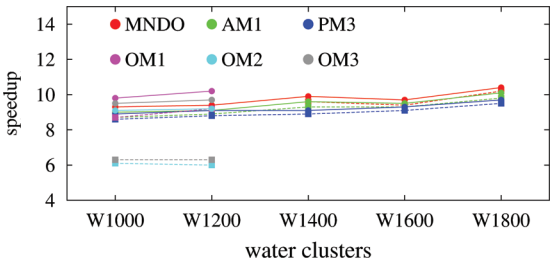


Figure 5. Speedup of the semiempirical single-point energy calculations of water clusters on the hybrid multicore CPU–GPU platform (H6CG, dashed lines with squares; and H6CG*, solid lines with circles) with respect to one single CPU core (1C). The calculations make use of pseudodiagonalization (without DIIS).

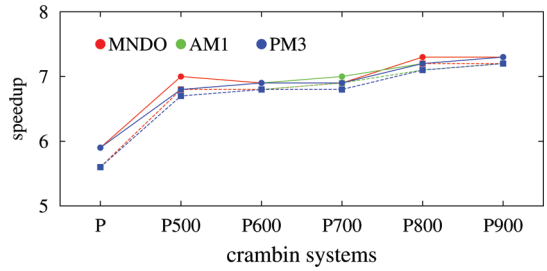


Figure 6. Speedup of the semiempirical single-point energy calculations of solvated crambin systems on the hybrid multicore CPU–GPU platform (H6CG, dashed lines with squares; and H6CG*, solid lines with circles) with respect to one single CPU core (1C). The calculations employ full diagonalization and DIIS extrapolation (no use of pseudodiagonalization).

dx.doi.org/10.1021/ct3001798 | J. Chem. Theory Comput. 2012, 8, 2272–2281

The overall speedups generally increase with the size of the test molecules and the number of basis functions (N) ranging from 240 to 10 800. Small molecules like C_{60} benefit only little from the GPU. For example, relative to 1C, MNDO-type and OMx calculations on C_{60} are merely ~1.5 times faster on H6CG and H6CG*. Several factors restrict the performance for smaller molecules. First, routines other than those ported to the GPU take a higher percentage of the computation time, for example, 12.1% for C_{60} versus 1.6% for C_{960} (see Figure 2a). Second, the CPU-only computations for small molecules are really fast, for example, 0.1 s for C_{60}, so that the overhead for CPU−GPU communication on the hybrid platform is no longer negligible. Hence, there is not much advantage of using the GPU for semiempirical calculations on small molecules (say, less than 100 atoms).

In the fullerene and water cluster series, the MNDO-type calculations for the largest systems reach maximum speedups on the hybrid multicore CPU−GPU platform of around 13 (C_{960}) and 10 ($1800{\cdot}H_2O$). Because of the memory limitations on our workstation, OMx calculations are possible for systems up to the size of $1200{\cdot}H_2O$, for which speedups of 9−10 can be achieved (Figure 5).

In the MNDO-type calculations on the solvated crambin series, the observed speedups are generally somewhat lower (around 7, see Figure 6). This is mostly due to the deliberate decision to turn off pseudodiagonalizations (PDIAG) and use only full diagonalizations (FDIAG) in these calculations. FDIAG, which dominates the computation time in this case (>60%, see Figure 2), is accelerated typically only 5-fold on the GPU (see Figure 3 in the Supporting Information), much less so than PDIAG.

Finally, it should be pointed out that the number of required SCF iterations may occasionally differ on different platforms, despite using double-precision floating point arithmetic throughout (see the Supporting Information). These differences arise from roundoff errors and are unavoidable. In such cases, the calculated properties of the molecules, for example, heats of formation, ionization energies, etc., are the same on the different platforms (within the limits imposed by the SCF convergence criteria).

5.2. Profiles on the Hybrid Multicore CPU−GPU Platform. In this section, we examine the hotspots in semiempirical calculations on the hybrid CPU−GPU platform (H6CG). For the reasons outlined in section 3, we only show the H6CG profiles of the AM1 and OM2 calculations for all test systems in Figure 7. The data for the other methods are documented in the Supporting Information.

It is obvious that Fock matrix diagonalization (FDIAG, in red) and pseudodiagonalization (PDIAG, in green) demand a larger share of the computation time than on the CPU-only platform (Figure 2). For example, the percentage of time consumed by FDIAG grows from 64% to 82% in the MNDO calculation of unsolvated crambin. The reason for this larger share is that both SYEVD in FDIAG and the Jacobi transformation in PDIAG are bandwidth-bound and are thus not accelerated too much on the GPU, that is, typically by a factor of ~5; see Figure 3 in the Supporting Information. By contrast, the major computations in BORDER and DIIS, that is, the matrix multiplications in SYRK and GEMM, are dramatically accelerated on the GPU (by a factor of about 25, see Figure 4 in the Supporting Information), and they are thus no longer bottlenecks in semiempirical calculations on H6CG. The other routines, for example, integral evaluation and

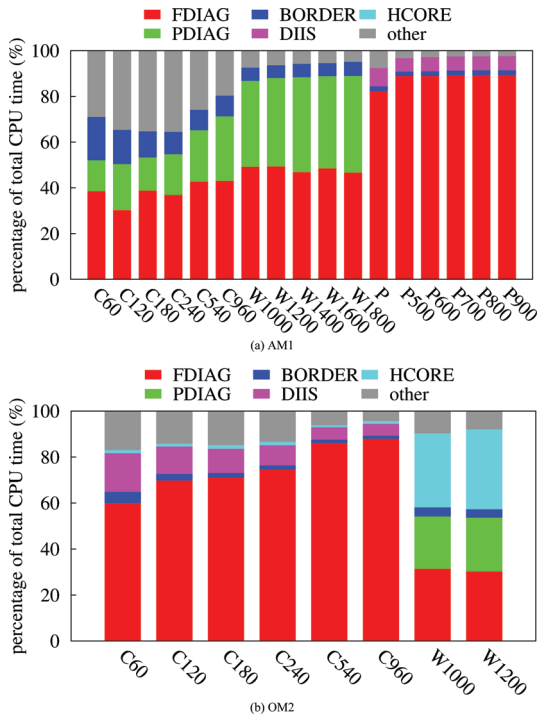


Figure 7. Profiles of the semiempirical AM1 and OM2 calculations of the test molecules on the hybrid multicore CPU−GPU computing platform.

construction of the Fock matrix, have not yet been ported to the GPU in our present work. Their share of the computation time starts to rise on H6CG especially for small molecules, but remains rather low for large molecules in MNDO-type calculations.

Figure 8 presents the profiles of PDIAG in the MNDO calculations on H6CG. Unlike the situation on a single CPU

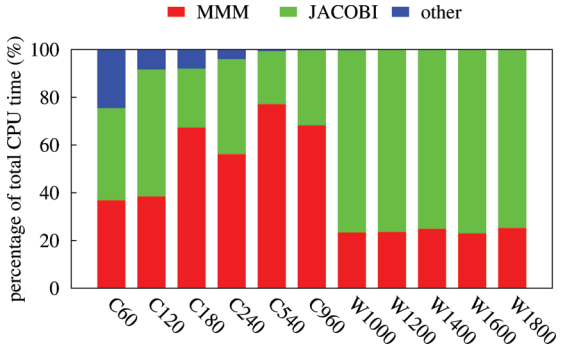


Figure 8. The profiles of PDIAG in the MNDO calculations of fullerenes and water clusters on the hybrid CPU−GPU platform.

core (Figure 3) where the triple matrix product (MMM) consumes more time than the Jacobi transformation (JACOBI), the latter takes most of the computation time on the GPU (~80%). This is because the matrix−matrix multiplication is far more accelerated on the GPU than the Jacobi transformation, with speedups of about 25 (see Figure 4 in the Supporting Information) versus 4−7 (Table 2).

The H6CG profiles of the OMx calculations on water clusters look quite different from the other profiles, particularly in the case of OM2 and OM3; see Figure 7b. The computations in HCORE become quite demanding, taking about 32% and

dx.doi.org/10.1021/ct3001798 | J. Chem. Theory Comput. 2012, 8, 2272−2281

33% of the time for OM2 and OM3, respectively. This latter increase is caused by the three-center orthogonalization corrections to the resonance integrals,[26] which are presently still computed in HCORE by a single-threaded CPU code. These corrections are crucial for the quality of the OM*x* results.[33−35] One way to accelerate their computation is to turn on OpenMP parallelization (H6CG*, see above), but it would also seem worthwhile to develop a dedicated GPU kernel and find out whether this offers an even more efficient solution on hybrid CPU−GPU platforms.

5.3. Geometry Optimizations. As a further test, geometry optimizations of the fullerenes were carried out at the MNDO level using the 1C, H6CG, and H6CG* platforms. In the MNDO program, the required gradients are computed by default using a simple finite-difference procedure with recalculated integrals and a constant density matrix. This procedure scales as $O(N^2)$ and thus does not constitute a bottleneck in the present approach, but will benefit from the existing OpenMP parallelization (in full analogy to integral evaluation). Therefore, the geometry optimizations were performed without any additional GPU-oriented modification of the code using both H6CG and H6CG*. They were done in internal coordinates to exploit the symmetry of the fullerenes. The performance data are shown in Table 5. The speedups increase with the size of the fullerene being optimized, as expected, and they are generally somewhat higher than those obtained in the single-point calculations (see the Supporting Information). The improved performance arises from the fact that the relatively slow full diagonalizations (FDIAG) at the beginning of an SCF calculation can normally be avoided in the course of a geometry optimization because the density matrix from the preceding point usually provides a sufficiently good starting guess for the SCF procedure of the next point. For the largest fullerene considered (C_{960}), the MNDO geometry optimizations on the hybrid multicore CPU−GPU platform thus achieve record speedups of 15.5 (H6CG) and 19.2 (H6CG*).

Table 5. Computation Times (in seconds) for MNDO Geometry Optimization of Fullerenes on a Single CPU Core (1C) and on the Hybrid Multicore CPU−GPU Platform (H6CG and H6CG*)[a]

	C_{60}	C_{120}	C_{180}	C_{240}	C_{540}	C_{960}
N_{opt}	4	49[b]	14	28	34	40[c]
symmetry	I_h	T_d	I_h	I_h	I_h	I_h
N_{var}	2	17	6	7	15	26
1C	0.46	59.41	42.68	274.49	4927.51	57 651.94
H6CG	0.29	14.76	8.27	39.88	398.28	3731.21
speedup	1.5	4.0	5.2	6.9	12.4	15.5
H6CG*	0.24	10.37	5.82	28.54	309.63	3007.62
speedup*	1.9	5.7	7.3	9.6	15.9	19.2

[a] N_{opt} and N_{var} denote the number of the optimization cycles and geometric variables, respectively. OpenMP parallelization was turned on using 6 CPU cores in H6CG* (our best effort). [b] The geometry optimization of C_{120} finishes ($\|g\|$ = 0.05 kcal/(mol·Å)) in 44 cycles on H6CG. [c] An incomplete optimization for benchmark purposes with 40 cycles.

6. CONCLUSIONS

In this work, GPU-accelerated routines for semiempirical quantum chemical calculations were implemented on a hybrid multicore CPU−GPU platform to enable more efficient computations with MNDO-type methods (for MNDO, AM1, PM3) and with orthogonalization-corrected methods (OM1, OM2, OM3). Systematic calculations on fullerenes, water clusters, and solvated crambin systems were performed with all of these methods on a variety of computing environments encompassing one single CPU core, multiple CPU cores, a single CPU core with GPU, and a multicore CPU with GPU.

For all methods considered, the computational bottlenecks on one single CPU core were identified to be full diagonalization (FDIAG), pseudodiagonalization (PDIAG), and matrix multiplications during density matrix formation (BORDER) and SCF convergence acceleration (DIIS). The computational effort in other routines, for example, integral evaluation and Fock matrix construction, turned out to be small for large molecules. Vendor-optimized library functions and a manually tuned GPU kernel for Jacobi transformations (PDIAG) were employed in the profile-guided code optimization on the hybrid multicore CPU−GPU platform. The least accelerated routines on the GPU were found to be FDIAG and PDIAG, which both contain parts that are memory-bandwidth bound. Therefore, FDIAG and PDIAG govern the overall speedup that can be achieved in semiempirical calculations on the hybrid CPU−GPU platform.

The performance gains in the semiempirical calculations increase with the size of the molecule. On average, speedups close to 10 are achieved in MNDO-type and OM*x* single-point calculations for large molecules, on the multicore CPU−GPU platform relative to one single CPU core, with a maximum of 12.9 for the AM1 energy evaluation of the largest fullerene (C_{960}, 3840 basis functions). MNDO geometry optimizations for the largest fullerenes on the hybrid multicore CPU−GPU platform reach an even higher performance (with speedups up to 19.2). The OM*x* calculations are slightly less accelerated than the MNDO-type calculations because of the orthogonalization corrections to the core Hamiltonian. Their computation requires a significant share of the overall time on the hybrid platform (especially for the three-center terms in OM2 and OM3). While their evaluation benefits from the available OpenMP parallelization, it would still seem desirable to develop a carefully optimized GPU kernel for this task in future work.

■ ASSOCIATED CONTENT

Supporting Information

Further profiles of semiempirical SCF calculations on CPU and hybrid CPU-GPU platforms (MNDO, PM3, OM1, and OM3), performance tests of standard library routines, detailed description of the GPU algorithms designed for the Jacobi transformation, benchmarks of the optimized GPU kernel for this transformation, and computation times for the semiempirical calculations of the test molecules on all computing platforms. This material is available free of charge via the Internet at http://pubs.acs.org.

■ AUTHOR INFORMATION

Corresponding Author

*E-mail: thiel@mpi-muelheim.mpg.de.

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

We would like to thank Wolfgang Angenendt for implementing parts of the OpenMP parallelization.

■ REFERENCES

(1) Clark, T.; Stewart, J. J. P. In *Computational Methods for Large Systems*; Reimers, J. R., Ed.; John Wiley & Sons, Inc.: Hoboken, NJ, 2011; pp 259−286.
(2) Thiel, W. In *Theory and Applications of Computational Chemistry*; Dykstra, C. E., Frenking, G., Kim, K. S., Scuseria, G. E., Eds.; Elsevier: Amsterdam, 2005; pp 559−580.
(3) Bredow, T.; Jug, K. *Theor. Chem. Acc.* **2005**, *113*, 1−14.
(4) Thiel, W. In *Modern Methods and Algorithms of Quantum Chemistry*; Grotendorst, J., Ed.; John von Neumann Institute for Computing: Jülich, 2000; pp 261−283.
(5) Clark, T. *J. Mol. Struct. (THEOCHEM)* **2000**, *530*, 1−10.
(6) Thiel, W. *Advances in Chemical Physics*; John Wiley & Sons, Inc.: New York, 1996; pp 703−757.
(7) Stewart, J. J. P. *Reviews in Computational Chemistry*; John Wiley & Sons, Inc.: New York, 1990; pp 45−81.
(8) Thiel, W. *Tetrahedron* **1988**, *44*, 7393−7408.
(9) Dewar, M. J. S.; Thiel, W. *Theor. Chim. Acta* **1977**, *46*, 89−104.
(10) *NVIDIA CUDA C Programming Guide*; NVIDIA Corp.: Santa Clara, CA, 2010.
(11) Stone, J. E.; Hardy, D. J.; Ufimtsev, I. S.; Schulten, K. *J. Mol. Graphics Modell.* **2010**, *29*, 116−125.
(12) Farber, R. M. *J. Mol. Graphics Modell.* **2011**, *30*, 82−89.
(13) van der Spoel, D.; Hess, B. *WIREs Comput. Mol. Sci.* **2011**, *1*, 710−715.
(14) Anderson, A. G.; Goddard, W. A., III; Schröder, P. *Comput. Phys. Commun.* **2007**, *177*, 298−306.
(15) (a) Ufimtsev, I. S.; Martínez, T. J. *Comput. Sci. Eng.* **2008**, *10*, 26−34. (b) Ufimtsev, I. S.; Martínez, T. J. *J. Chem. Theory Comput.* **2008**, *4*, 222−231. (c) Ufimtsev, I. S.; Martínez, T. J. *J. Chem. Theory Comput.* **2009**, *5*, 1004−1015. (d) Ufimtsev, I. S.; Martínez, T. J. *J. Chem. Theory Comput.* **2009**, *5*, 2619−2628. (e) Luehr, N.; Ufimtsev, I. S.; Martínez, T. J. *J. Chem. Theory Comput.* **2011**, *7*, 949−954. (f) Isborn, C. M.; Luehr, N.; Ufimtsev, I. S.; Martínez, T. J. *J. Chem. Theory Comput.* **2011**, *7*, 1814−1823.
(16) (a) Yasuda, K. *J. Comput. Chem.* **2008**, *29*, 334−342. (b) Yasuda, K. *J. Chem. Theory Comput.* **2008**, *4*, 1230−1236.
(17) Asadchev, A.; Allada, V.; Felder, J.; Bode, B. M.; Gordon, M. S.; Windus, T. L. *J. Chem. Theory Comput.* **2010**, *6*, 696−704.
(18) (a) Vogt, L.; Olivares-Amaya, R.; Kermes, S.; Shao, Y.; Amador-Bedolla, C.; Aspuru-Guzik, A. *J. Phys. Chem. A* **2008**, *112*, 2049−2057. (b) Olivares-Amaya, R.; Watson, M. A.; Edgar, R. G.; Vogt, L.; Shao, Y.; Aspuru-Guzik, A. *J. Chem. Theory Comput.* **2010**, *6*, 135−144.
(19) (a) DePrince, A. E.; Hammond, J. R. *J. Chem. Theory Comput.* **2011**, *7*, 1287−1295. (b) Ma, W.; Krishnamoorthy, S.; Villa, O.; Kowalski, K. *J. Chem. Theory Comput.* **2011**, *7*, 1316−1327.
(20) Wilkinson, K. A.; Sherwood, P.; Guest, M. F.; Naidoo, K. J. *J. Comput. Chem.* **2011**, *32*, 2313−2318.
(21) Mangueira, C. P., Jr.; Carvalho, J. D.; Cabral, L. A. F.; Rocha, G. B. *XVI Brasilian Symposium of Theoretical Chemistry*; Ouro Preto, 21 November, 2011; Poster P377.
(22) Thiel, W. *MNDO99 CVS Development Version*; Max-Planck-Institut für Kohlenforschung: Mülheim an der Ruhr, Germany, 2012.
(23) (a) Dewar, M. J. S.; Thiel, W. *J. Am. Chem. Soc.* **1977**, *99*, 4899−4907. (b) Dewar, M. J. S.; Thiel, W. *J. Am. Chem. Soc.* **1977**, *99*, 4907−4917.
(24) Dewar, M. J. S.; Zoebisch, E. G.; Healy, E. F.; Stewart, J. J. P. *J. Am. Chem. Soc.* **1985**, *107*, 3902−3909.
(25) (a) Stewart, J. J. P. *J. Comput. Chem.* **1989**, *10*, 209−220. (b) Stewart, J. J. P. *J. Comput. Chem.* **1989**, *10*, 221−264.
(26) (a) Kolb, M.; Thiel, W. *J. Comput. Chem.* **1993**, *14*, 775−789. (b) Weber, W.; Thiel, W. *Theor. Chem. Acc.* **2000**, *103*, 495−506. (c) Scholten, M. Semiempirische Verfahren mit Orthogonalisierungskorrekturen: Die OM3 Methode. Ph.D. thesis, University of Düsseldorf, 2003.
(27) Bakowies, D.; Thiel, W. *J. Am. Chem. Soc.* **1991**, *113*, 3704−3714.
(28) Bakowies, D.; Bühl, M.; Thiel, W. *J. Am. Chem. Soc.* **1995**, *117*, 10113−10118.
(29) Martínez, L.; Andrade, R.; Birgin, E. G.; Martínez, J. M. *J. Comput. Chem.* **2009**, *30*, 2157−2164.
(30) *CUDA CUBLAS Library*; NVIDIA Corp.: Santa Clara, CA, 2010.
(31) Tomov, S.; Nath, R.; Du, P.; Dongarra, J. *MAGMA Users' Guide*; Innovative Computing Laboratory, University of Tennessee: Knoxville, TN, 2010.
(32) Pople, J. A.; Santry, D. P.; Segal, G. A. *J. Chem. Phys.* **1965**, *43*, S129−S135.
(33) Otte, N.; Scholten, M.; Thiel, W. *J. Phys. Chem. A* **2007**, *111*, 5751−5755.
(34) Korth, M.; Thiel, W. *J. Chem. Theory Comput.* **2011**, *7*, 2929−2936.
(35) Silva-Junior, M. R.; Thiel, W. *J. Chem. Theory Comput.* **2010**, *6*, 1546−1564.
(36) i (i = 60, 120, 180, 240, 540, 960) is the number of carbon atoms in the fullerenes. j (j = 1000, 1200, 1400, 1600, 1800) and k (k = 500, 600, 700, 800, 900) specify the number of water molecules in the water clusters and in the solvation spheres around crambin, respectively. An isolated crambin (642 atoms) is denoted as P.
(37) Intel Turbo Boost Technology, which increases the CPU clock depending on the workload, has been switched off to obtain consistent benchmark timings.
(38) For example, in the case of crambin·900H_2O, 6 DIIS cycles in double precision demand about 3 GB memory on the GPU, which combined with the requirements by other subroutines approaches the memory capacity of the GPU.
(39) Pulay, P. *J. Comput. Chem.* **1982**, *3*, 556−560.
(40) Thiel, W.; Green, D. G. In *Methods and Techniques in Computational Chemistry: METECC-95*; Clementi, E., Corongiu, G., Eds.; METECC (Series); STEF: Cagliari, 1995; pp 141−168.
(41) Stewart, J. J. P.; Császár, P.; Pulay, P. *J. Comput. Chem.* **1982**, *3*, 227−228.
(42) Detailed information on all tests is given in the Supporting Information.
(43) Anderson, E.; Bai, Z.; Bischof, C.; Blackford, S.; Demmel, J.; Dongarra, J.; Croz, J. D.; Greenbaum, A.; Hammarling, S.; McKenney, A.; Sorensen, D. *LAPACK Users' Guide*, 3rd ed.; Society for Industrial and Applied Mathematics: Philadelphia, PA, 1999.
(44) The three steps in the MAGMA SYEVD implementation are (i) reduce a real symmetric matrix to tridiagonal form (SYTRD on GPU), (ii) compute all eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the divide and conquer method (STEDC on CPU), and (iii) multiply a real matrix by a previously determined orthogonal matrix (DORMTR on GPU).
(45) Novakovic, V.; Singer, S., arXiv:1008.1371v2.
(46) Modi, J. J.; Parkinson, D. *Comput. Phys. Commun.* **1982**, *26*, 317−320.
(47) Berry, M.; Sameh, A. *J. Comp. Appl. Math.* **1989**, *27*, 191−213.
(48) Eberlein, P. J.; Park, H. *J. Par. Dist. Comp.* **1990**, *8*, 358−366.