11
Semiempirical Quantum Chemistry

Xin Wu, Axel Koslowski and Walter Thiel
Max-Planck-Institut für Kohlenforschung, Mülheim an der Ruhr, Germany

In this chapter, we demonstrate how graphics processing units (GPUs) can be used to accelerate large-scale semiempirical quantum-chemical calculations on hybrid CPU–GPU platforms. We examine the computational bottlenecks using a series of calculations on eight proteins with up to 3558 atoms and outline how relevant operations are parallelized and ported to GPUs, making use of multiple devices where possible. Significant speedups are achieved that enable simulations on large systems with thousands of atoms. As an example we present results for geometry optimizations of three representative proteins with α-helix, β-sheet, and random coil structures using several common semiempirical Hamiltonians.

11.1 Introduction

Semiempirical quantum chemical methods are cost-effective tools for chemists to study the structure, stability, and spectroscopy of molecules as well as chemical reactions [1] (see also Chapter 3). They are based on the Hartree–Fock method commonly used in *ab initio* molecular orbital (MO) theory [2]. The different semiempirical models simplify the Hartree–Fock procedure by introducing distinct approximations to the Hamiltonian, neglecting many integrals to speed up computations by several orders of magnitude [3]. The remaining integrals are modeled using empirical functions with adjustable parameters that are calibrated against a large number of accurate experimental or high-level theoretical reference data to make semiempirical methods as reliable and general as possible. These features make semiempirical models well suited to many research areas in chemistry, and enabled a large number of semiempirical applications already in the 1970s and 1980s. Since the 1990s, density functional theory (DFT) has become the major workhorse in computational chemistry [4]. However, considering that semiempirical methods are ~1000× faster than standard DFT approaches [5], they are still valuable computational tools nowadays, for example, for screening large numbers of drug

Electronic Structure Calculations on Graphics Processing Units:
From Quantum Chemistry to Condensed Matter Physics, First Edition.
Edited by Ross C. Walker and Andreas W. Götz.

candidates [6], for calculations on proteins [7], for long-time scale ground-state molecular dynamics simulations [8], and for nonadiabatic excited-state dynamics of large chromophores [9].

The development of computational chemistry is intimately tied to the evolution of computer technology. The first computational chemistry programs developed since the 1950s had been exclusively written for sequential execution on a single central processing unit (CPU) [10]. With the widespread advent of parallel computing in the 1990s, many quantum chemical codes were parallelized to take advantage of the new architectures, including semiempirical programs [11]. The most recent wave of hardware-driven code development was triggered by the rise of graphics processing units (GPUs). A GPU is a specially designed integrated circuit with powerful, but fixed-function pipelines for faster image rendering and video games (see also Chapter 1). Until 2006, implementing algorithms for general numeric calculations on a GPU was tediously difficult because the problem had to be cast into graphics operations by resorting to a specific (graphics) API (application programming interface). Programming purely computational tasks on a GPU was considerably simplified by the introduction of the CUDA (compute unified device architecture) and OpenCL (open computing language) frameworks (see Chapter 2). In this chapter, we will focus exclusively on the CUDA framework, which allows developers to employ the C programming language, with CUDA-specific extensions, to use a CUDA-capable GPU as coprocessor of the CPU for computations [12]. As of 2012, the raw hardware peak performance and memory bandwidth of a many-core GPU had significantly outpaced a multicore CPU (see Figure 1.5 in Chapter 1). For example, the maximum floating-point performance and theoretical memory bandwidth of an Intel Xeon E5-4650 CPU (eight cores with a base clock of 2.7 GHz and a maximum boost clock of 3.3 GHz with the Intel Turbo Boost Technology, four-channel DDR-1600) are 0.17–0.21 TFlop/s (floating-point operations per second) and 51.2 GB/s, respectively. By contrast, the flagship Tesla K20x by Nvidia (2688 CUDA cores at 732 MHz) has a peak of 1.31 TFlop/s for double-precision arithmetic and a memory bandwidth of 250 GB/s with ECC (error-correcting code) off. Hence many groups decided to develop GPU-accelerated programs [13, 14] to take advantage of this promising device for quantum Monte Carlo computations [15, 16], evaluation of two-electron integrals [17–22], DFT calculations [23–30], high-level correlated *ab initio* methods [31–38], and semiempirical quantum chemistry [39, 40]. The other chapters of this book contain an excellent overview of many of these porting efforts.

In this chapter, we begin with a brief review of semiempirical quantum chemistry, referring readers interested in the detailed formalism and the numerical results to available books [41–43] and reviews [5, 11, 44–50]. We then examine the computational bottlenecks by performing systematic calculations on a set of eight proteins with up to 3558 atoms and 8727 basis functions. Thereafter, we outline how the hotspots identified in this manner are ported to a GPU (making use of multiple devices where possible), and how the remaining code is parallelized using CPUs only using the symmetric multiprocessing (SMP) capabilities via OpenMP. Next, we analyze the overall performance of our code on the hybrid CPU–GPU platform and compare it with the CPU-only case. Finally, as an illustrative application, we use our CPU–GPU hybrid program to optimize the geometries of three small proteins, each consisting predominantly of one type of secondary structure, namely α-helix, β-strand, and random coil, employing six different semiempirical methods.

11.2 Overview of Semiempirical Methods

Nonrelativistic quantum chemistry aims at finding sufficiently accurate but approximate solutions to the Schrödinger equation. In the early days of quantum chemistry, the zero-differential-overlap (ZDO) approximation [51, 52] was introduced to deal with "the nightmare of the integrals" [10], that is, the difficulty of evaluating the large number of three- and four-center integrals in *ab initio* methods. As a consequence, the integral problem could be tackled at different levels of approximation. Currently, the most accurate semiempirical methods are based on the NDDO (neglect of diatomic differential overlap) model [3], which retains all one- and two-center two-electron repulsion integrals in the Fock matrix. The first successful and widely adopted NDDO-based parameterization was the MNDO (modified neglect of diatomic overlap) method [53–55]. The MNDO model also serves as the basis for later parameterizations that have been widely applied, including AM1 (Austin Model 1) [56], PMx (parametric methods, $x = 3, 5, 6$, and 7) [57–60], as well as PDDG/MNDO and PDDG/PM3 (MNDO and PM3 augmented with pairwise distance directed Gaussian functions) [61].

Conceptual deficiencies in the established MNDO-type methods include the lack of representation of Pauli exchange repulsion in the Fock matrix. One possible remedy is to introduce orthogonalization corrections into the Fock matrix to account for Pauli exchange repulsion. This can be done through truncated and parameterized series expansions in terms of overlap, which provide corrections to the one-electron core Hamiltonian. These corrections are applied to the one-center matrix elements in OM1 (orthogonalization model 1) [62] and to all one- and two-center matrix elements in OM2 [63] and OM3 [64]. Benchmark calculations demonstrate that the OMx methods, especially OM2 and OM3, are superior to AM1 and PM3 for both ground-state and excited-state molecular properties [65–67]. The computational cost of OMx calculations is roughly the same as that for MNDO-type calculations [39], especially when using suitable cutoffs to neglect the exponentially decreasing three-center orthogonalization corrections to matrix elements involving distant atoms.

11.3 Computational Bottlenecks

In this chapter, the OM3 method is taken as an example to illustrate the general strategy of optimizing a semiempirical quantum chemical program on a hybrid CPU–GPU platform. We have selected a set of eight proteins that are denoted as P_x (x being the number of residues) and listed in Table 11.1, for the purpose of profiling OM3 calculations in a systematic manner [68–75]. Timings for the OMx methods are also representative for MNDO-type methods, because the most time-consuming parts of the calculations are the same in both cases. Consequently, similar wall clock times are obtained: for example, one SCF (self-consistent field) iteration in MNDO, AM1, PM3, OM1, OM2, and OM3 calculations on a cluster of 1000 water molecules takes 80, 84, 89, 73, 87, and 83 seconds, respectively, on a single Intel Xeon X5670 CPU core [39]. Hence, it is sufficient to consider only OM3 in the following.

The OM3 calculations on our test proteins were performed on a server with two Intel Xeon X5690 CPUs (six cores at 3.46 GHz per chip), 48 GiB host memory (24 GiB of triple-channel DDR-1333 per chip) with a total theoretical bandwidth[1] of 64 GB/s, and two Nvidia Tesla M2090 GPUs (512 CUDA cores at 1.3 GHz per device) with 5.25 GiB ECC memory and a bandwidth of 155 GB/s per device. Intel Turbo Boost Technology (which may automatically increase the CPU frequency above the base clock in accordance with the workload in order to exhaust the allowed thermal envelope of the CPU) was intentionally turned off to ensure consistent timings. Three criteria were adopted for SCF convergence in our single-point energy calculations: (i) a variation of the electronic energy in successive

Table 11.1 *Proteins in the test set for the OM3 calculations*

Notation	P_{020}	P_{063}	P_{086}	P_{100}	P_{125}	P_{156}	P_{166}	P_{221}
PDB ID	1BTQ	1K50	2HXX	3K6F	1ACF	2A4V	4A02	3AQO
N_a	307	1097	1495	1842	2004	2969	3415	3558
N_f	754	2699	3655	4446	4920	7157	8173	8727

N_a and N_f denote the number of atoms and basis functions, respectively.

[1] If one CPU needs to access memory connected to the other CPU, the theoretical bandwidth is lower.

SCF iterations of at most 1.0×10^{-6} eV, (ii) a maximum change of the density matrix elements of 1.0×10^{-6}, and (iii) a maximum entry in the error matrix of 1.0×10^{-6} in the DIIS (direct inversion of iterative subspace) extrapolation [76]. To speed up the calculations, the full diagonalization was automatically replaced in the SCF procedure by fast pseudo-diagonalization [77] whenever possible.

The code development was conducted on a CVS version of the MNDO99 package [78]. The Intel composer XE 13.1 and Nvidia CUDA Toolkit 5.0 were used for compiling the Fortran subroutines of the CPU code and the CUDA kernels for the GPU, respectively. The final executable was dynamically linked against Intel Math Kernel Library (MKL) 11.0, CUBLAS from the Nvidia Toolkit, and MAGMA version 1.3.0 [79]. The latter includes a subset of LAPACK routines ported to the GPU; it has been modified locally to conform to the ILP64 (64-bit integers, long integers, and pointers) data model, which is needed to access arrays with 2^{32} or more elements.[2] Before the inclusion of dynamic memory allocation in the Fortran standard, the early versions of the MNDO program emulated dynamic memory by passing sections of a fixed-size array in the unnamed COMMON block as arguments to subroutines. The current version of the MNDO code uses essentially the same mechanism, but with a dynamically allocated array instead of the fixed-size array. For larger proteins, the indices of this array may exceed the 32-bit integer range—this is why 64-bit integers are needed.

The computing setup for the OM3 benchmark calculations is denoted as $C_{[x\mathrm{C}-y\mathrm{G}]}$, where the subscripts x and y are numbers of CPU cores and GPU devices in use, respectively. The wall clock time of an OM3 calculation on $C_{[1\mathrm{C}]}$ is the reference for calculations with the other compute configurations and the basis for assessing the corresponding speedups. Timings for $C_{[1\mathrm{G}]}$ and $C_{[2\mathrm{G}]}$ refer to subroutines executed exclusively on one GPU or two GPUs, respectively, including the associated and generally negligible CPU–GPU communication. All floating-point operations were done in double precision, both on the CPUs and GPUs, and therefore the numerical results produced on all hardware setups are essentially the same. Deviations in the computed heat of formation (total energy) were occasionally encountered, but remained below 1.0×10^{-5} kcal/mol. Such tiny discrepancies can be attributed to the different order in which the floating-point operations are performed on the CPU and GPU architectures. Since many operations are performed in parallel, the execution order may not even be fixed, that is, there might be small deviations between different runs of the same calculation on the same computing setup. The execution order matters because limited-precision floating-point arithmetics is not associative.

The general form of a two-electron repulsion integral (ERI) in *ab initio* and DFT methods is

$$(\mu\nu|\lambda\sigma) = \int_1\int_2 \frac{\mu(1)\ \nu(1)\ \lambda(2)\ \sigma(2)}{r_{12}}\ dV_1\ dV_2,$$

where the Greek letters represent basis functions or atomic orbitals (AOs). The complexity of the two-electron integral evaluation formally scales as $\mathcal{O}(N_f^4)$ for N_f basis functions, but the actual scaling may be more favorable due to the application of screening techniques [80]. The currently applied semiempirical methods make use of the NDDO approximation [3] for ERI evaluation:

$$(\mu_A\nu_B|\lambda_C\sigma_D) = \delta_{AB}\ \delta_{CD}\ (\mu_A\nu_B|\lambda_C\sigma_D),$$

where atomic centers are denoted by capital letters and δ_{AB} (or δ_{CD}) will vanish unless A and B (or C and D) are the same atom. This rather drastic approximation reduces the formal scaling of the ERI computation in semiempirical methods to $\mathcal{O}(N_f^2)$ and makes it possible to simulate complex systems with thousands of atoms. The solution of the secular equations

$$\sum_\nu (F_{\mu\nu} - \delta_{\mu\nu}\epsilon_i)C_{\nu i} = 0 \tag{11.1}$$

scales as $\mathcal{O}(N_f^3)$ and thus becomes the primary computational task in semiempirical methods. ϵ_i is the energy of the ith MO. Because the Fock matrix elements $F_{\mu\nu}$ depend on the elements $C_{\nu i}$ of the

[2] Starting with version 1.4, MAGMA supports both 32-bit and 64-bit integers out of the box.

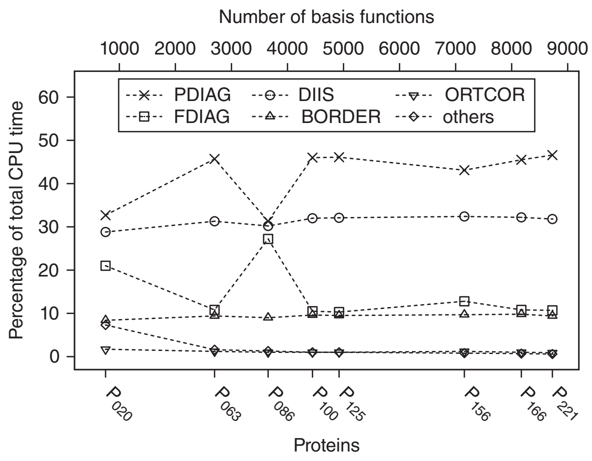

Figure 11.1 *Profiles of the OM3 calculations for the test proteins for the $C_{[1C]}$ computing setup*

eigenvectors, Eq. (11.1) has to be solved by an iterative SCF procedure that requires $\mathcal{O}(N_f^3)$ dense linear algebraic operations.

Figure 11.1 depicts the profiles of OM3 calculations for the $C_{[1C]}$ setup. The pseudo-diagonalization procedure (PDIAG) is roughly twice as fast as a full diagonalization (FDIAG), and it is thus preferable to replace FDIAG by PDIAG as often as possible. Applying the default criteria of the MNDO code for the choice between FDIAG and PDIAG, it is normally sufficient to call FDIAG in four of the SCF iterations (i.e., the first three and the last one) during single-point energy evaluation and to call PDIAG in the other SCF iterations (typically 25).[3] Hence most OM3 calculations are dominated by PDIAG with 42.1% of the wall clock time on average. FDIAG and PDIAG complement each other; they collectively contribute ~55% of the total CPU time and are thus the first two bottlenecks.

DIIS is the third hotspot that consumes ~30% of the computation time (see Figure 11.1). Although the DIIS extrapolation may be omitted for small systems (with less than 100 atoms), it is in our experience imperative to apply DIIS to reliably converge the SCF procedure for larger molecules such as proteins. We will thus also investigate the option of leveraging multiple GPUs for the DIIS treatment (see Section 11.4).

The last two bottlenecks are the calculation of the density matrix (also called the bond-order matrix, subroutine BORDER) and the orthogonalization corrections (subroutine ORTCOR in the case of OM3). We spent considerable effort on both routines to achieve optimum performance with the MNDO99 program [78], especially for ORTCOR, where we obtained a huge speedup by formulating all operations as standard matrix–matrix multiplications. After code optimization, BORDER and ORTCOR take 9.4% and 1.1% of the wall clock time on average, respectively, on the $C_{[1C]}$ setup.

Other computational tasks in an OM3 calculation include integral evaluation, formation of the Fock matrix, and initial density matrix generation, which all scale as $\mathcal{O}(N_f^2)$. Cumulatively, they require 7% of the CPU time in a serial calculation for a small protein such as P_{020} with 307 atoms and 754 orbitals, but this portion quickly diminishes with increasing system size, to ~0.5% for the largest proteins in our test set, which are the main targets of our code development. Therefore, these other tasks are not considered to be real bottlenecks, and the corresponding subroutines are thus only subjected to an OpenMP parallelization to take advantage of multiple CPU cores.

[3] An exception is P_{086} with 11 calls to FDIAG.

In summary, we have identified five subroutines (FDIAG, PDIAG, DIIS, BORDER, and ORTCOR) as computational bottlenecks by systematic analysis of OM3 calculations on a set of proteins. We describe the optimization of these hotspots on a hybrid CPU–GPU platform in the following.

11.4 Profile-Guided Optimization for the Hybrid Platform

11.4.1 Full Diagonalization, Density Matrix, and DIIS

The GPU-accelerated full diagonalization, density matrix construction, and DIIS extrapolation are jointly described here because they heavily rely on the standard routines in the BLAS (basic linear algebra subprograms) and LAPACK (linear algebra package) libraries.

Equation (11.1) is an eigenvalue problem that can be solved by diagonalizing the Fock matrix **F**, which yields the *i*th MO energy ϵ_i and the coefficient vector $\mathbf{c}_i$:

$$\mathbf{F}\mathbf{c}_i = \epsilon_i \mathbf{c}_i.$$

This task can be carried out by the LAPACK function DSYEVD, which computes all eigenvalues and eigenvectors of a real symmetric matrix using the divide-and-conquer algorithm. DSYEVD of the Intel MKL library makes use of all processor cores on a CPU-only platform, whereas the DSYEVD implementation in MAGMA is a hybrid that utilizes both multicore CPUs and GPU(s)[4] for the diagonalization [81]. In Figure 11.2, the speedups of FDIAG are plotted as obtained in the OM3 calculations on the proteins in our test set. The scalability on CPU-only setups is evidently rather poor: for instance, the best speedups are observed in the calculations on P_{063}, which are 4.3 on $C_{[6C]}$ and 5.4 on $C_{[12C]}$. Hence, the symmetric parallel processors are highly underutilized in the FDIAG subroutine, and the efficiency[5] is merely 0.72 and 0.45, respectively. This becomes even worse for larger systems: for example, the speedup of FDIAG for P_{221} on $C_{[6C]}$ is 3.3 and barely increases to 3.8

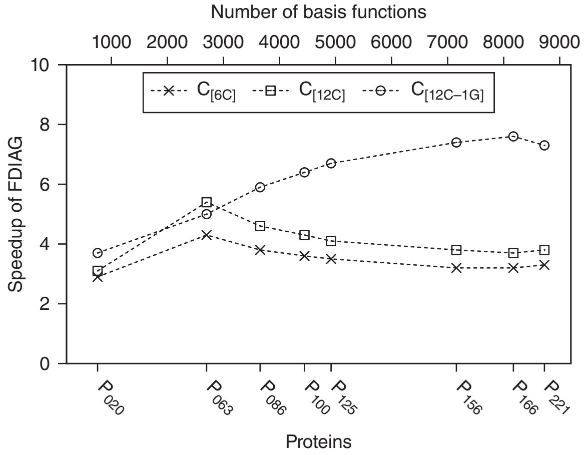

Figure 11.2 *Speedups of the FDIAG subroutine in the OM3 calculations on the multi-CPU $C_{[6C]}$, $C_{[12C]}$, and hybrid CPU–GPU $C_{[12C-1G]}$ computing setups over the serial configuration*

[4] The hybrid DSYEVD function in MAGMA version 1.3 does not support multiple GPUs. This feature is available starting with MAGMA version 1.4.

[5] Processor efficiency is defined as the speedup divided by the number of parallel processing units.

on $C_{[12C]}$, with corresponding efficiencies of 0.55 and 0.32, respectively. On the contrary, the speedup of the hybrid FDIAG subroutine is constantly rising until P_{166} (up to more than 8000 basis functions). Moreover, it is always superior to its CPU-only counterpart with the exception of P_{063} on the $C_{[12C]}$ setup. For the larger calculations, the hybrid FDIAG subroutine tends to be more than 7× faster than the serial version and at least 2× faster than the parallel CPU-only version.

The primary computational task (>99% of the CPU time) in BORDER is a matrix–matrix multiplication, $\mathbf{P} = 2\mathbf{C}_o\mathbf{C}_o^T$, where **P** is the density matrix and $\mathbf{C}_o$ is the coefficient matrix of the occupied MOs. A general DGEMM routine could be used to perform this task. Because **P** is symmetric, and only the lower triangle is stored as a linear array in the MNDO99 package, we employ a more specific function, namely DSYRK, which only calculates the lower part of a symmetric matrix and thus avoids unnecessary floating-point operations. The CPU-only DSYRK routine has no difficulty to fully load all processors, and the performance scales almost ideally with respect to the number of CPU cores (see Figure 11.3). For example, the speedups for P_{166} are 5.8 on $C_{[6C]}$ and 9.9 on $C_{[12C]}$. At present, no multi-GPU-enabled version of DSYRK is available in either CUBLAS or MAGMA. On the other hand, DSYRK on a single GPU may be more than 20× faster than a single-threaded CPU routine. Thus, we will stick to DSYRK in our development, hoping that multi-GPU support will be added by the vendors in the future.

The DIIS procedure is composed of several different kinds of algebraic operations, in which the calculation of the error matrix ($\mathbf{\Delta} = \mathbf{FP} - \mathbf{PF}$) usually consumes more than 98% of the CPU time [39]. Because the product of **F** and **P** is a general matrix, the standard DGEMM function is chosen for the DIIS subroutine. The number of floating-point operations and memory accesses in DGEMM scale as $\mathcal{O}(N^3)$ and $\mathcal{O}(N^2)$ (N being the matrix dimension), respectively. This implies that the number of compute operations per memory access is proportional to N in DGEMM. Thus DGEMM is a compute-bound routine that should be well suited to parallelization. The observed speedups on the CPU-only setups are ~5.5 on $C_{[6C]}$ and ~10.0 on $C_{[12C]}$. Moreover, a call to DIIS accelerated by a single GPU ($C_{[1G]}$) can be up to 20× faster than for the baseline setup $C_{[1C]}$. However, the speedup for $C_{[1G]}$ turns out not to be monotonous with increasing system size: it is highest for P_{125} with ~20 and then drops again for the next-larger protein P_{156} to ~18.

In order to make the best use of our dual-GPU equipped hardware, we designed a block matrix scheme for the matrix–matrix multiplication aimed at multiple GPU devices based on the standard DGEMM routine (X. Wu, A. Koslowski, W. Thiel, unpublished results). There are of course more

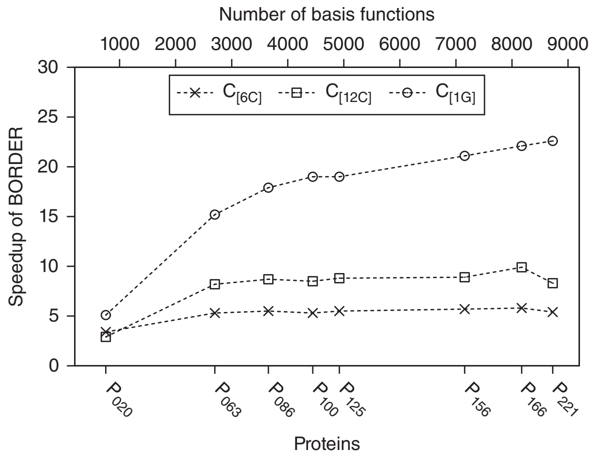

Figure 11.3 *Speedups of the BORDER subroutine in the OM3 calculations on the multi-CPU $C_{[6C]}$, $C_{[12C]}$, and GPU-only $C_{[1G]}$ computing setups over the serial configuration*

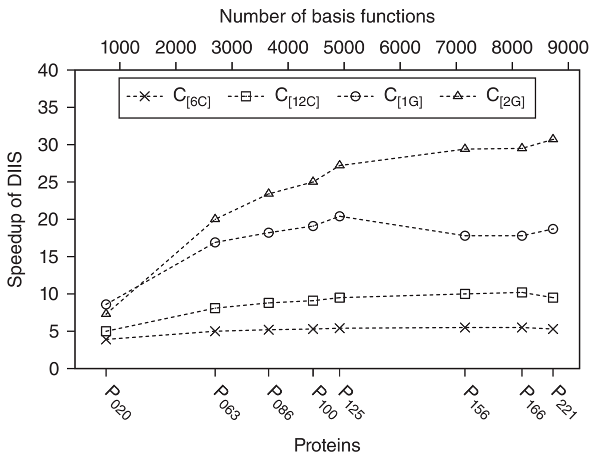

Figure 11.4 *Speedups of the DIIS subroutine in the OM3 calculations on the multi-CPU $C_{[6C]}$, $C_{[12C]}$, and GPU-only $C_{[1G]}$ and $C_{[2G]}$ computing setups over the serial configuration*

sophisticated multi-GPU DGEMM implementations reported in the literature [82, 83], but the performance of our homemade multi-GPU DGEMM is virtually doubled on two GPUs ($C_{[2G]}$ compared to $C_{[1G]}$) with a peak around 0.7 TFlop/s.

The overall speedup for the DIIS procedure with the multi-GPU DGEMM routine on the $C_{[2G]}$ setup is plotted in Figure 11.4. We find a monotonous increase in performance up to a factor of 30 compared with the $C_{[1C]}$ setup. The use of two GPU devices ($C_{[2G]}$) results in a 1.6-fold speedup over the setup with one single GPU ($C_{[1G]}$).

11.4.2 Pseudo-diagonalization

As mentioned in the previous section, pseudo-diagonalization will be ~2× faster than the conventional diagonalization in a given SCF iteration. Thus PDIAG is used instead of FDIAG whenever possible. However, an efficient implementation of PDIAG on multiple GPUs can be challenging. Here, we first analyze the computations involved in pseudo-diagonalization, and then report the individual and overall speedups that have been achieved.

The details of pseudo-diagonalization have been described in the original paper [77]. From a computational point of view, it is basically comprised of two tasks. First, the Fock matrix is transformed from the AO basis to the MO basis by a triple matrix multiplication (FMO):

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

where $\mathbf{C}_{\mathrm{o}}$ and $\mathbf{C}_{\mathrm{v}}$ denote the matrices of the occupied and virtual MO vectors, respectively. Then noniterative Jacobi-like 2×2 rotations (JACOBI) between pairs of occupied ($\mathbf{c}_{\mathrm{o}}$) and virtual ($\mathbf{c}_{\mathrm{v}}$) vectors are executed:

$$\mathbf{c}_{\mathrm{o}}' = a\mathbf{c}_{\mathrm{o}} - b\mathbf{c}_{\mathrm{v}} \qquad \text{and} \qquad \mathbf{c}_{\mathrm{v}}' = b\mathbf{c}_{\mathrm{o}} + a\mathbf{c}_{\mathrm{v}}, \tag{11.2}$$

where a and b are the elements of the rotation matrix, and the new MO vectors $\mathbf{c}_o'$ and $\mathbf{c}_v'$ are denoted by primes.

The profiles of the serial PDIAG version for the OM3 calculations on the proteins in our test set are given in Table 11.2. On average, FMO and JACOBI consume ~45% and ~55% of the CPU time, respectively. The other operations are negligible (<1%) and can be safely excluded from optimization.

Table 11.2 Percentages (%) of computation time in the PDIAG subroutine consumed by FMO, JACOBI, and other tasks in the OM3 calculations on a single CPU core

Notation	P_{020}	P_{063}	P_{086}	P_{100}	P_{125}	P_{156}	P_{166}	P_{221}
FMO	47.2	40.8	45.1	41.8	42.5	44.7	42.2	42.3
JACOBI	51.6	58.8	54.5	57.9	57.2	55.1	57.6	57.5
Others	1.3	0.5	0.4	0.3	0.2	0.2	0.2	0.2

Table 11.3 Speedups of the FMO and JACOBI steps in the PDIAG subroutine on the multi-CPU $C_{[6C]}$, $C_{[12C]}$, and GPU-only $C_{[1G]}$ and $C_{[2G]}$ computing setups over the serial setup

	FMO				JACOBI			
	$C_{[6C]}$	$C_{[12C]}$	$C_{[1G]}$	$C_{[2G]}$	$C_{[6C]}$	$C_{[12C]}$	$C_{[1G]}$	$C_{[2G]}$
P_{020}	5.2	7.8	5.9	5.2	3.4	5.1	2.6	3.6
P_{063}	5.7	10.2	16.2	18.6	1.6	1.6	4.4	7.9
P_{086}	5.8	10.6	19.6	22.4	1.4	1.4	4.6	8.6
P_{100}	5.8	10.7	20.0	23.1	1.3	1.2	4.5	8.1
P_{125}	5.8	10.8	20.3	25.4	1.2	1.2	5.0	9.4
P_{156}	5.8	11.3	21.0	30.4	1.2	1.2	4.4	8.6
P_{166}	5.8	11.3	20.6	31.9	1.2	1.2	4.4	8.6
P_{221}	5.5	10.6	20.8	32.9	1.4	1.4	5.1	9.9

The FMO step contains only the DGEMM calls for the matrix multiplications. The relevant speedups with different computing configurations are summarized in Table 11.3. Since DGEMM is compute-bound, FMO scales well with respect to the number of parallel processors in the CPU-only setups. One single GPU-accelerated FMO step can be as much as 20× faster than on one CPU core. The setup with two GPU devices may further increase the speedup to more than 30-fold, being about 1.6× faster than on $C_{[1G]}$. The best performance for a small protein like P_{020} is achieved with the CPU-only setup of 12 cores, however. This is because a GPU is designed for massively parallel tasks that a small system will not fully exploit, and some inevitable overhead such as CPU–GPU data transfer may hurt the overall performance of a smaller calculation.

The GPU-oriented optimization of the JACOBI step is demanding. The technical details can be found in our paper [39]. The resulting speedups are shown in Table 11.3. As one 2×2 rotation given in Eq. (11.2) involves six memory accesses (four reads and two writes) and six floating-point operations, the performance of JACOBI is fully determined by the memory bandwidth. In the case of P_{020}, the MO coefficient matrix is small enough (4.3 MiB) to completely fit into the CPU cache (12 MiB per chip). Modest speedups of 3.4 and 5.1 are therefore achieved on the $C_{[6C]}$ and $C_{[12C]}$ setups, respectively. On the other hand, numerous cache misses can occur for larger proteins starting from P_{063}. The performance on the CPU-only platform will then be determined entirely by the available memory bandwidth. The obtained speedup rapidly falls down to 1.2, no matter how many CPU cores are in use for parallelization. On the contrary, JACOBI on a single GPU benefits from the enhanced memory bandwidth (155 GB/s vs. 64 GB/s for two CPUs), and speedups of around 4.5-fold are consistently achieved in the benchmarks except for the smallest case, P_{020}. Addition of a second GPU doubles the total memory bandwidth, and the equal distribution of horizontal blocks of the coefficient matrix among the available devices enables the rotations to be carried out independently on each device (X. Wu, A. Koslowski, W. Thiel, unpublished results). The overall speedup on the $C_{[2G]}$ setup for P_{221} is 10, which is 1.9× higher than that on a single GPU ($C_{[1G]}$).

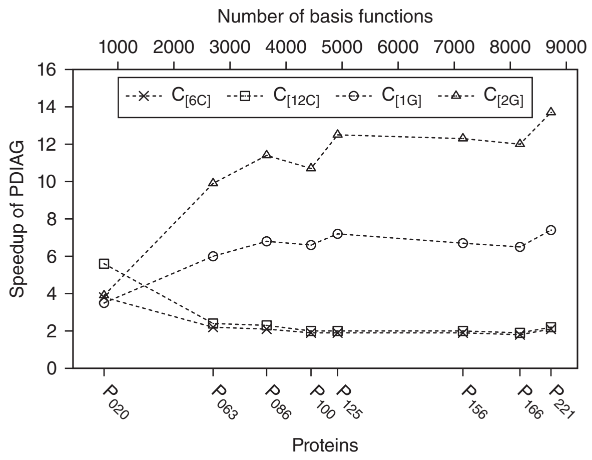

Figure 11.5 *Speedups of the PDIAG subroutine in the OM3 calculations on the multi-CPU $C_{[6C]}$, $C_{[12C]}$, and GPU-only $C_{[1G]}$ and $C_{[2G]}$ computing setups over the serial configuration*

Since the JACOBI step consumes a slightly higher fraction of the CPU time (between 55% and 60% for most proteins in our test set) than FMO in the PDIAG subroutine for the serial configuration, and since JACOBI benefits less from parallelization than PDIAG on all computing setups, the overall speedups of PDIAG shown in Figure 11.5 resemble those of JACOBI (see Table 11.3), but with some additional performance benefits from the FMO step. The highest speedup is 13.7 for P_{221} on $C_{[2G]}$, which is again 1.9× higher than that on a single GPU ($C_{[1G]}$).

11.4.3 Orthogonalization Corrections in OM3

The OM3 method [64] accounts for Pauli exchange repulsion by explicitly adding the orthogonalization corrections ($V^{\text{ORT}}_{\mu_A\nu_B}$) to the core Hamiltonian of the Fock matrix:

$$V^{\text{ORT}}_{\mu_A\nu_B} = -\frac{1}{2}G_1^{AB}\sum_{\lambda_C}(S_{\mu_A\lambda_C}\beta_{\lambda_C\nu_B} + \beta_{\mu_A\lambda_C}S_{\lambda_C\nu_B}) \qquad (\text{C} \neq \text{A and C} \neq \text{B}),$$

where S and β denote elements of the overlap and resonance matrices, respectively, and G_1^{AB} is defined in terms of parameters that can be adjusted to fit reference data. μ_A, ν_B, and λ_C are AOs at atoms A, B, and C, respectively. If A and B are the same atom, $V^{\text{ORT}}_{\mu_A\nu_B}$ is a correction to a one-center term; otherwise it refers to a two-center element. Inclusion of the latter three-center contributions leads to qualitative improvements over the MNDO-type methods for calculated molecular properties, such as rotational barriers, relative energies of isomers, hydrogen bonds, and vertical excitation energies [1, 65–67].

Even though the ORTCOR subroutine consumes only ~1% of the wall clock time for the $C_{[1C]}$ setup, we implemented a dedicated algorithm utilizing multiple GPUs in an attempt to harness all available computing power. The ORTCOR performance for various setups is depicted in Figure 11.6. The technical details will be presented elsewhere.

The speedup of the ORTCOR subroutine scales reasonably well on the symmetric multi-CPU setups. For example, 5.5- and 10.1-fold performance boosts are feasible on the $C_{[6C]}$ and $C_{[12C]}$ setups, respectively. ORTCOR is accelerated up to 28-fold for medium-sized proteins like P_{063} on a single GPU ($C_{[1G]}$ setup), but thereafter the speedup decreases again with increasing system size to ~20 for the largest proteins in our test set. The speedup on the $C_{[2G]}$ setup can reach 35-fold for a moderately

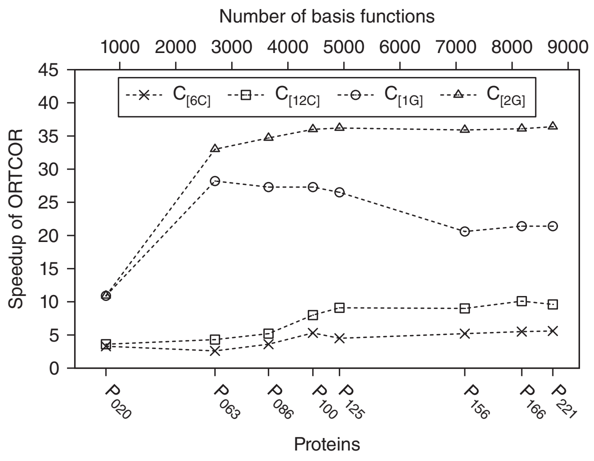

Figure 11.6 *Speedups of the ORTCOR subroutine in the OM3 calculations on the multi-CPU $C_{[6C]}$, $C_{[12C]}$, and GPU-only $C_{[1G]}$ and $C_{[2G]}$ computing setups over the serial configuration*

sized protein, and there is no performance deterioration for larger proteins. Moreover, the multi-GPU ORTCOR scales well compared to a single GPU device for sufficiently large proteins. For example, ORTCOR is 1.7× faster on $C_{[2G]}$ than on $C_{[1G]}$ for P_{221}.

11.5 Performance

Since a user will of course never run an individual subroutine by itself, the overall speedups for the OM3 calculations on proteins are more relevant in practice. They are presented in Figure 11.7.

The performance of the OM3 calculations on the CPU-only platform can hardly be improved by using more processor cores. The speedups quickly reach a saturation point and never exceed 4. The

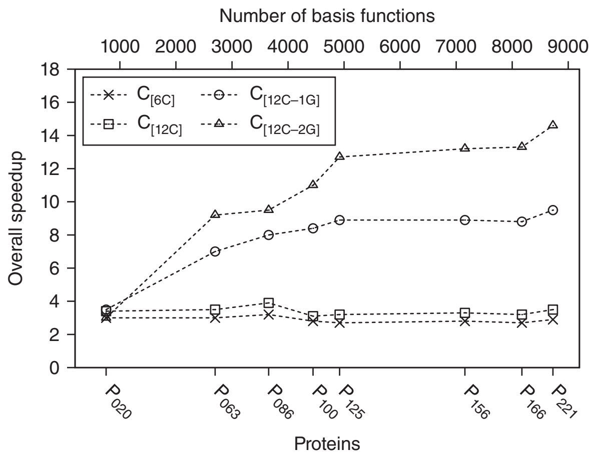

Figure 11.7 *Overall speedups of the OM3 calculations of test proteins on the multi-CPU $C_{[6C]}$, $C_{[12C]}$, and hybrid CPU–GPU $C_{[12C-1G]}$ and $C_{[12C-2G]}$ computing setups over the serial configuration*

mean values averaged over the proteins in the test set are 2.9 and 3.4 on $C_{[6C]}$ and $C_{[12C]}$, respectively. Moreover, the individual speedups seem to be almost invariant with respect to the size of the protein. Thus neither using more CPU cores nor increasing the system size yields higher speedups on the CPU-only setup. At first glance, this conclusion seems to contradict our previous result that the speedup of an MNDO calculation on fullerene C_{540} could reach 7.7 on Cray Y-MP with eight vector processors [84]. This apparent discrepancy can be resolved by considering the relevant arithmetic operations and the differences in the computer architectures. Concerning the computational bottlenecks mentioned in the preceding section, only three subroutines (BORDER, DIIS, and ORTCOR) of the five hotspots in the OM3 calculations can be well accelerated on current hardware by using additional CPU cores (see Figures 11.3, 11.4, and 11.6), whereas neither FDIAG nor PDIAG, which consume ~65% of the wall clock time, scale favorably with the number of cores (see Figures 11.2 and 11.5). This is because the former three are primarily dominated by compute-bound routines, which demand more arithmetic power than memory bandwidth. On the other hand, both diagonalization subroutines are composed of bandwidth-bound operations that would parallelize well on more CPU cores if and only if the demand for memory bandwidth could be satisfied in the first place. The theoretical floating-point peak performance of the two Xeon X5690 CPUs (a total of 166 GFlop/s) exceeds that of the Cray Y-MP (2.6 GFlop/s) by a factor of 64. The theoretical memory bandwidth of our current Xeon server (64 GB/s), however, is merely 2× greater than that of the 25-year-old Cray Y-MP (32 GB/s). Therefore, a tremendously inadequate memory bandwidth prevents the performance boost on a computer system including only parallel superscalar CPUs.

Because of the advantages of GPUs with regard to floating-point peak performance and memory bandwidth, the speedups achieved for the OM3 calculations on GPUs are monotonously growing with the size of the proteins and the number of GPUs (see Figure 11.7). Although the hybrid CPU–GPU platform provides higher speedups than the CPU-only platform for most bottlenecks, there may be exceptions in the case of calculations on small proteins like P_{020}. This may be due to the CPU–GPU communication overhead, to the unfavorable behavior of certain subroutines for small systems on a hybrid platform compared to a CPU-only setup (especially PDIAG, see Figure 11.5), or to the less optimized non-GPU routines becoming more dominant. For example, the CPU-only computation on P_{020} takes 41% of the wall clock time for the $C_{[12C\text{-}2G]}$ setup (see Figure 11.8, label "others"). Thus the overall performance of the OM3 calculations for P_{020} is rather similar on all computing setups. On the other hand, the acceleration on the hybrid CPU–GPU and CPU-only platforms is quite different for

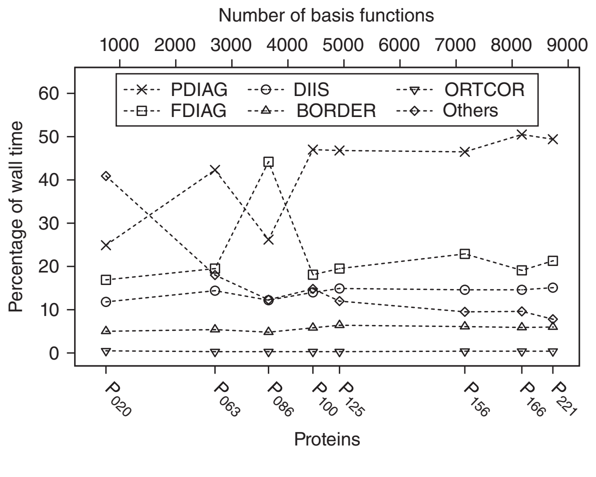

Figure 11.8 *Profiles of the OM3 calculations for the test proteins on the $C_{[12C-2G]}$ computing setup*

large calculations. The speedups of the OM3 calculations for P_{221} reach 9.5 and 14.6 on the $C_{[12C\text{-}1G]}$ and $C_{[12C\text{-}2G]}$ setups, respectively. The relative speedup of $C_{[12C\text{-}2G]}$ over $C_{[12C\text{-}1G]}$ is ~1.5 for the OM3 calculations of large proteins. Further performance increases are thus very likely when more GPU devices are employed in even larger semiempirical quantum chemical calculations.

Finally, we inspect the profiles of the OM3 calculations on the hybrid $C_{[12C\text{-}2G]}$ setup (see Figure 11.8). DIIS, BORDER, and ORTCOR are the three subroutines most accelerated on the GPU, thus their combined share of the wall clock time is just about half of that on $C_{[1C]}$. On average, the shares of DIIS, BORDER, and ORTCOR amount to 31.4%, 9.4%, 1.1% and 14.0%, 5.7%, 0.4% on the $C_{[1C]}$ and $C_{[12C\text{-}2G]}$ setups, respectively. The speedups for FDIAG and PDIAG are not as good as those for the former three routines, and hence their combined share on the $C_{[12C\text{-}2G]}$ setup is increased to 64.4% on average. The remaining subroutines (e.g., for integral evaluation and Fock matrix formation) have not yet been ported to a GPU, but are executed in parallel using multiple CPU cores (via OpenMP). They become the bottlenecks for small protein calculations with a time share of 40.9% in P_{020}, which gradually decreases with system size, down to 7.8% for a large protein like P_{221}. We may thus anticipate some further improvement of the overall performance with dedicated multi-GPU kernels for the semiempirical integral evaluation and Fock matrix construction.

11.6 Applications

Given the code developments outlined above, it has now become a routine task to carry out semiempirical quantum chemical calculations for large biomolecules, such as proteins, on a hybrid CPU–GPU computing platform. We have carried out full geometry optimizations of three proteins with α-helix, β-sheet, and random coil structures (see Figure 11.9), which were chosen from a collection of proteins used in previous work [85]. Six different semiempirical methods were applied, namely MNDO, AM1, PM3, and OMx (x = 1, 2, and 3). The optimizations were terminated when the gradient vector norm dropped below a preselected threshold value ($|\mathbf{g}| \leq 1.0\ \text{kcal} \cdot \text{mol}^{-1} \cdot \text{Å}^{-1}$). The quality of the computed structures was assessed in terms of the conformation of the main chain by using the PROCHECK package [86], in comparison with the structures determined in aqueous solution by nuclear magnetic resonance (NMR) experiments. It should be stressed that the results given here are just for demonstration, since more realistic simulations would require more elaborate approches (e.g., including explicit solvent).

The backbones of the proteins are shown in Figure 11.9. Highly regular local structures imposed by hydrogen bonds are found in P_A and P_B, whereas P_C possesses an unfolded polypeptide chain. The backbone conformation of a protein is determined by a pair of less rigid dihedral angles [ϕ, ψ] at

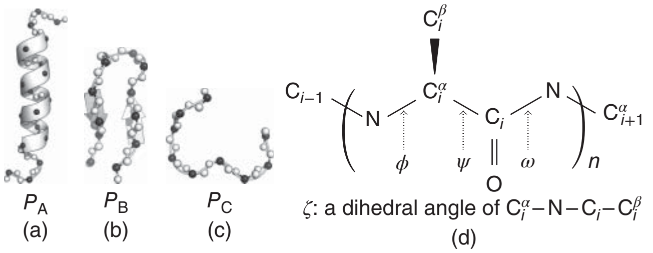

Figure 11.9 *Experimental structures of (a) P_A (PDB ID: 2AP7, 80% α-helix), (b) P_B (PDB ID: 2EVQ, 50% β-strands), and (c) P_C (PDB ID: 1LVR, 100% random coil). Only the backbone atoms are shown, with the C^α atoms represented by black balls. Four dihedral angles (ϕ, ψ, ω, and ζ) in a residue serve as stereochemical metrics, see the schematic sketch in (d)*

Table 11.4 *Statistics (%) for [ϕ, ψ] in the most favored ($P^0_{\phi,\psi}$) and additionally allowed ($P^1_{\phi,\psi}$) regions of the Ramachandran plot and standard deviations (°) of ω and ζ*

	P_A				P_B				P_C			
	$P^0_{\phi,\psi}$	$P^1_{\phi,\psi}$	σ_ω	σ_ζ	$P^0_{\phi,\psi}$	$P^1_{\phi,\psi}$	σ_ω	σ_ζ	$P^0_{\phi,\psi}$	$P^1_{\phi,\psi}$	σ_ω	σ_ζ
Expt.	100.0	0.0	0.8	0.6	87.5	12.5	2.6	0.7	42.9	42.9	2.3	1.8
MNDO	91.7	8.3	9.0	1.3	87.5	12.5	14.9	0.5	28.6	57.1	17.0	2.5
AM1	75.0	16.7	9.7	1.0	100.0	0.0	15.8	1.0	28.6	71.4	7.0	1.9
PM3	75.0	25.0	15.3	1.3	87.5	12.5	19.7	1.0	42.9	57.1	22.8	1.7
OM1	81.8	18.2	12.4	1.0	87.5	12.5	9.4	0.9	28.6	71.4	12.5	2.5
OM2	83.3	16.7	8.6	1.1	87.5	12.5	10.4	1.1	42.9	42.9	10.5	2.1
OM3	83.3	16.7	7.6	1.1	87.5	12.5	15.7	1.2	42.9	42.9	14.3	2.5

Results for the experimental structures of P_A, P_B, and P_C are compared with those calculated by semiempirical quantum chemical methods.

the C^α-atom [87] and a stiff torsion angle ω of the peptide bond. ω is usually restricted to be around 180° for an energetically more favorable trans conformer due to the partial double bond character of the amide bond, which prevents facile rotation. In addition, a virtual dihedral angle ζ is defined between C^α_i–N and noncovalently bound $C_i \cdots C^\beta_i$ as a measure of chirality at the central C^α_i atom of the amino acid [88].

PROCHECK divides a Ramachandran map into four regions: most favored, additionally allowed, generously allowed, and disallowed. The shares of the first two distributions, $P^0_{\phi,\psi}$ and $P^1_{\phi,\psi}$, for P_A, P_B, and P_C are listed in Table 11.4. In most cases, $P^0_{\phi,\psi}$ and $P^1_{\phi,\psi}$ add up to the total population. Neither experimental nor theoretically optimized protein structures are spoiled by disallowed $[\phi, \psi]$ combinations. Since more regular secondary structures exist in P_A and P_B than in the disordered P_C, significantly higher values for $P^0_{\phi,\psi}$ are obtained for the former two proteins. MNDO, AM1, PM3, and OM1 predict a higher $[\phi, \psi]$ population in the additionally allowed region for P_C, whereas OM2, OM3, and the NMR experiment give equal values for $P^0_{\phi,\psi}$ and $P^1_{\phi,\psi}$. Although the deficiencies of the original MNDO method for the description of hydrogen bonds are known from early studies [89, 90], its actual performance for the proteins in the test set seems rather satisfactory. Both the α-helix (in P_A) and β-strand (in P_B) structures are found, and reasonable $[\phi, \psi]$ distributions are retained in the optimized structures.

All semiempirical methods predict greater deviations from planarity around the peptide bond than deduced from experiment (see the σ_ω values). Such deviations from planarity in the peptide group have already been reported in earlier theoretical studies [7, 91, 92]: the sp^2-hybrid nitrogen in a peptide bond should be planar, but it tends to be pyramidalized in semiempirical calculations. The average value of ζ for L-amino acids is 33.81 ± 4.17° [88]. The σ_ζ values from experiment and from semiempirical calculations are rather small and of similar quality, indicating a good description of the local environment of the sp^3-C^α_i atoms in the main chains.

11.7 Conclusion

In this chapter, we have presented a profile-guided optimization of the semiempirical quantum chemical MNDO program on a hybrid CPU–GPU platform. OM3 calculations on a set of eight proteins were used to guide the code development and to assess the performance. The computational bottlenecks on one single CPU core were identified as the diagonalization of the Fock matrix (FDIAG), fast pseudodiagonalization (PDIAG), SCF acceleration (DIIS), density matrix formation (BORDER), and computation of the orthogonalization corrections in OM3 (ORTCOR), which cover altogether ~99% of the wall clock time in the test runs. Standard library routines and special finely tuned kernels targeting multiple GPU devices were employed to accelerate these routines, whereas the relevant remaining subroutines (~1% of the computation time) were run in parallel using multiple CPU cores (via OpenMP) to achieve optimum performance on the hybrid CPU–GPU platform.

We have identified severe restraints to parallelize the semiempirical calculations on currently available CPU-only computing architectures. No matter how many processor cores are utilized in a calculation, a ceiling of the overall acceleration is reached rapidly because of the limitations imposed by the hardware memory bandwidth. On the other hand, the speedup of the calculations on the hybrid CPU–GPU platform rises continuously with increasing system size and reaches one order of magnitude in large protein calculations. The overall performance can be further improved through the use of multiple GPUs.

As an illustrative application, geometry optimizations of three typical proteins with α-helix, β-sheet, and random coil structures were carried out by means of the MNDO, AM1, PM3, and OMx ($x = 1, 2$, and 3) methods. These calculations produced qualitatively reasonable conformations of the main chains (with regard to the usual metrics for assessing protein backbone structures) but showed some deviation from experiment by giving slightly nonplanar peptide bonds. We are confident that such quantitative deficiencies can be ameliorated in future semiempirical method development. This will enhance the impact of the current code development work on hybrid CPU–GPU platforms, which has enabled semiempirical quantum chemical calculations on large systems such as proteins with thousands of atoms.

Acknowledgement

This work was supported by an ERC Advanced Grant (OMSQC).

References

1. Thiel, W. (2014) Semiempirical quantum-chemical methods. *WIREs Comput. Mol. Sci.*, **4**, 145–157.
2. Hehre, W.J., Radom, L., Schleyer, P.v.R. and Pople, J.A. (1986) *Ab Initio Molecular Orbital Theory*, John Wiley & Sons, Inc., New York.
3. Pople, J.A., Santry, D.P. and Segal, G.A. (1965) Approximate self-consistent molecular orbital theory. I. Invariant procedures. *J. Chem. Phys.*, **43**, S129–S135.
4. Parr, R.G. and Yang, W. (1989) *Density-Functional Theory of Atoms and Molecules*, Oxford University Press, USA.
5. Thiel, W. (1996) Perspectives on semiempirical molecular orbital theory. *Adv. Chem. Phys.*, **93**, 703–757.
6. Ehresmann, B., de Groot, M.J., Alex, A. and Clark, T. (2004) New molecular descriptors based on local properties at the molecular surface and a boiling-point model derived from them. *J. Chem. Inf. Comput. Sci.*, **44**, 658–668.
7. Stewart, J.J.P. (2009) Application of the PM6 method to modeling proteins. *J. Mol. Model.*, **15**, 765–805.
8. Wu, X., Thiel, W., Pezeshki, S. and Lin, H. (2013) Specific reaction path hamiltonian for proton transfer in water: reparameterized semiempirical models. *J. Chem. Theory Comput.*, **9**, 2672–2686.
9. Fabiano, E., Lan, Z., Lu, Y. and Thiel, W. (2011) Nonadiabatic trajectory calculations with Ab initio and semiempirical methods, in *Conical Intersections: Theory, Computation and Experiment* (eds W. Domcke, D.R. Yarkony and H. Koppel), World Scientific Publishing Company, pp. 463–496.
10. Pople, J.A. (2003) Quantum chemical models, in *Nobel Lectures in Chemistry (1996–2000)* (ed. I. Grenthe), World Scientific Publishing Company, Singapore, pp. 246–260.

11. Thiel, W. and Green, D.G. (1995) The MNDO94 code: parallelization of a semiempirical qauntum-chemical program, in *Methods and Techniques in Computational Chemistry: METECC-95* (eds E. Clementi and G. Corongiu), STEF, Cagliari, pp. 141–168.
12. Kirk, D.B. and Hwu, W.M.W. (2012) *Programming Massively Parallel Processors: A Hands-on Approach*, Morgan Kaufmann Publishers.
13. Stone, J.E., Hardy, D.J., Ufimtsev, I.S. and Schulten, K. (2010) GPU-accelerated molecular modeling coming of age. *J. Mol. Graph. Model.*, **29**, 116–125.
14. Farber, R.M. (2011) Topical perspective on massive threading and parallelism. *J. Mol. Graph. Model.*, **30**, 82–89.
15. Anderson, A., Goddard, W.A. III and Schröder, P. (2007) Quantum Monte Carlo on graphical processing units. *Comput. Phys. Commun.*, **177**, 298–306.
16. Kim, J., Rodgers, J.M., Athènes, M. and Smit, B. (2011) Molecular Monte Carlo simulations using graphics processing units: to waste recycle or not? *J. Chem. Theory Comput.*, **7**, 3208–3222.
17. Yasuda, K. (2008) Two-electron integral evaluation on the graphics processor unit. *J. Comput. Chem.*, **29**, 334–342.
18. Ufimtsev, I.S. and Martínez, T.J. (2008) Quantum chemistry on graphical processing units. 1. Strategies for two-electron integral evaluation. *J. Chem. Theory Comput.*, **4**, 222–231.
19. Asadchev, A., Allada, V., Felder, J., Bode, B.M., Gordon, M.S. and Windus, T.L. (2010) Uncontracted Rys quadrature implementation of up to G functions on graphical processing units. *J. Chem. Theory Comput.*, **6**, 696–704.
20. Wilkinson, K.A., Sherwood, P., Guest, M.F. and Naidoo, K.J. (2011) Acceleration of the GAMESS-UK electronic structure package on graphical processing units. *J. Comput. Chem.*, **32**, 2313–2318.
21. Miao, Y. and Merz, K.M. (2013) Acceleration of electron repulsion integral evaluation on graphics processing units via use of recurrence relations. *J. Chem. Theory Comput.*, **9**, 965–976.
22. Titov, A.V., Ufimtsev, I.S., Luehr, N. and Martinez, T.J. (2013) Generating efficient quantum chemistry codes for novel architectures. *J. Chem. Theory Comput.*, **9**, 213–221.
23. Yasuda, K. (2008) Accelerating density functional calculations with graphics processing unit. *J. Chem. Theory Comput.*, **4**, 1230–1236.
24. Ufimtsev, I.S. and Martínez, T.J. (2009) Quantum chemistry on graphical processing units. 2. Direct self-consistent-field implementation. *J. Chem. Theory Comput.*, **5**, 1004–1015.
25. Ufimtsev, I.S. and Martínez, T.J. (2009) Quantum chemistry on graphical processing units. 3. Analytical energy gradients, geometry optimization, and first principles molecular dynamics. *J. Chem. Theory Comput.*, **5**, 2619–2628.
26. Ufimtsev, I.S. and Martínez, T.J. (2008) Graphical processing units for quantum chemistry. *Comput. Sci. Eng.*, **10**, 26–34.
27. Genovese, L., Ospici, M., Deutsch, T., Méhaut, J.-F., Neelov, A. and Goedecker, S. (2009) Density functional theory calculation on many-cores hybrid central processing unit-graphic processing unit architectures. *J. Chem. Phys.*, **131**, 034103 (8 pages).
28. Luehr, N., Ufimtsev, I.S. and Martínez, T.J. (2011) Dynamic precision for electron repulsion integral evaluation on graphical processing units (GPUs). *J. Chem. Theory Comput.*, **7**, 949–954.
29. Andrade, X. and Genovese, L. (2012) Harnessing the power of graphic processing units, in *Fundamentals of Time-Dependent Density Functional Theory*, Lecture Notes in Physics, vol. **837** (eds M.A. Marques, N.T. Maitra, F.M. Nogueira, E. Gross and A. Rubio), Springer-Verlag, pp. 401–413.
30. Andrade, X. and Aspuru-Guzik, A. (2013) Real-space density functional theory on graphical processing units: computational approach and comparison to Gaussian basis set methods. *J. Chem. Theory Comput.*, **9**, 4360–4373.

31. Isborn, C.M., Luehr, N., Ufimtsev, I.S. and Martínez, T.J. (2011) Excited-state electronic structure with configuration interaction singles and Tamm-Dancoff time-dependent density functional theory on graphical processing units. *J. Chem. Theory Comput.*, **7**, 1814–1823.
32. Vogt, L., Olivares-Amaya, R., Kermes, S., Shao, Y., Amador-Bedolla, C. and Aspuru-Guzik, A. (2008) Accelerating resolution-of-the-identity second-order Møller-Plesset quantum chemistry calculations with graphical processing units. *J. Phys. Chem. A*, **112**, 2049–2057.
33. Olivares-Amaya, R., Watson, M.A., Edgar, R.G., Vogt, L., Shao, Y. and Aspuru-Guzik, A. (2010) Accelerating correlated quantum chemistry calculations using graphical processing units and a mixed precision matrix multiplication library. *J. Chem. Theory Comput.*, **6**, 135–144.
34. Watson, M., Olivares-Amaya, R., Edgar, R.G. and Aspuru-Guzik, A. (2010) Accelerating correlated quantum chemistry calculations using graphical processing units. *Comput. Sci. Eng.*, **12**, 40–51.
35. DePrince, A.E. and Hammond, J.R. (2011) Coupled cluster theory on graphics processing units I. The coupled cluster doubles method. *J. Chem. Theory Comput.*, **7**, 1287–1295.
36. Ma, W., Krishnamoorthy, S., Villa, O. and Kowalski, K. (2011) GPU-based implementations of the noniterative regularized-CCSD(T) corrections: applications to strongly correlated systems. *J. Chem. Theory Comput.*, **7**, 1316–1327.
37. Bhaskaran-Nair, K., Ma, W., Krishnamoorthy, S., Villa, O., van Dam, H.J.J., Aprà, E. and Kowalski, K. (2013) Noniterative multireference coupled cluster methods on heterogeneous CPU-GPU systems. *J. Chem. Theory Comput.*, **9**, 1949–1957.
38. Asadchev, A. and Gordon, M.S. (2013) Fast and flexible coupled cluster implementation. *J. Chem. Theory Comput.*, **9**, 3385–3392.
39. Wu, X., Koslowski, A. and Thiel, W. (2012) Semiempirical quantum chemical calculations accelerated on a hybrid multicore CPU-GPU computing platform. *J. Chem. Theory Comput.*, **8**, 2272–2281.
40. Maia, J.D.C., Urquiza Carvalho, G.A., Mangueira, C.P., Santana, S.R., Cabral, L.A.F. and Rocha, G.B. (2012) GPU linear algebra libraries and GPGPU programming for accelerating MOPAC semiempirical quantum chemistry calculations. *J. Chem. Theory Comput.*, **8**, 3072–3081.
41. Dewar, M.J.S. (1969) *The Molecular Orbital Theory of Organic Chemistry*, McGraw-Hill Series in Advanced Chemistry, McGraw-Hill.
42. Pople, J.A. and Beveridge, D.L. (1970) *Approximate Molecular Orbital Theory*, McGraw-Hill Series in Advanced Chemistry, McGraw-Hill.
43. Murrell, J.N. and Harget, A.J. (1972) *Semi-Empirical Self-Consistent-Field Molecular Orbital Theory of Molecules*, Wiley-Interscience.
44. Thiel, W. (1988) Semiempirical methods: current status and perspectives. *Tetrahedron*, **44**, 7393–7408.
45. Thiel, W. (1997) Computational methods for large molecules. *J. Mol. Struct. THEOCHEM*, **398–399**, 1–6.
46. Thiel, W. (2000) Semiempirical methods, in *Modern Methods and Algorithms of Quantum Chemistry Proceedings*, 2nd edn (ed. J. Grotendorst), John von Neumann Institute for Computing, Jülich, pp. 261–283.
47. Thiel, W. (2005) Semiempirical quantum-chemical methods in computational chemistry, in *Theory and Applications of Computational Chemistry: The First Forty Years* (eds C.E. Dykstra, G. Frenking, K.S. Kim and G.E. Scuseria), Elsevier, Amsterdam, pp. 559–580.
48. Stewart, J.J.P. (1990) Semiempirical molecular orbital methods, in *Reviews in Computational Chemistry* (eds K.B. Lipkowitz and D.B. Boyd), John Wiley & Sons, Inc., pp. 45–81.
49. Zerner, M.C. (1991) Semiempirical molecular orbital methods, in *Reviews in Computational Chemistry* (eds K.B. Lipkowitz and D.B. Boyd), John Wiley & Sons, Inc., pp. 313–365.
50. Stewart, J.J.P. (1990) MOPAC: a semiempirical molecular orbital program. *J. Comput.-Aided Mol. Des.*, **4**, 1–103.

51. Parr, R.G. (1952) A method for estimating electronic repulsion integrals over LCAO MO's in complex unsaturated molecules. *J. Chem. Phys.*, **20**, 1499.
52. Pople, J.A. (1953) Electron interaction in unsaturated hydrocarbons. *Trans. Faraday Soc.*, **49**, 1375–1385.
53. Dewar, M.J.S. and Thiel, W. (1977) A semiempirical model for the two-center repulsion integrals in the NDDO approximation. *Theor. Chim. Acta*, **46**, 89–104.
54. Dewar, M.J.S. and Thiel, W. (1977) Ground states of molecules. 38. The MNDO method. Approximations and parameters. *J. Am. Chem. Soc.*, **99**, 4899–4907.
55. Dewar, M.J.S. and Thiel, W. (1977) Ground states of molecules. 39. MNDO results for molecules containing hydrogen, carbon, nitrogen, and oxygen. *J. Am. Chem. Soc.*, **99**, 4907–4917.
56. Dewar, M.J.S., Zoebisch, E.G., Healy, E.F. and Stewart, J.J.P. (1985) Development and use of quantum mechanical molecular models. 76. AM1: a new general purpose quantum mechanical molecular model. *J. Am. Chem. Soc.*, **107**, 3902–3909.
57. Stewart, J.J.P. (1989) Optimization of parameters for semiempirical methods I. Method. *J. Comput. Chem.*, **10**, 209–220.
58. Stewart, J.J.P. (2004) Comparison of the accuracy of semiempirical and some DFT methods for predicting heats of formation. *J. Mol. Model.*, **10**, 6–12.
59. Stewart, J.J.P. (2007) Optimization of parameters for semiempirical methods V: modification of NDDO approximations and application to 70 elements. *J. Mol. Model.*, **13**, 1173–1213.
60. Stewart, J.J.P. (2013) Optimization of parameters for semiempirical methods VI: more modifications to the NDDO approximations and re-optimization of parameters. *J. Mol. Model.*, **19**, 1–32.
61. Repasky, M.P., Chandrasekhar, J. and Jorgensen, W.L. (2002) PDDG/PM3 and PDDG/MNDO: improved semiempirical methods. *J. Comput. Chem.*, **23**, 1601–1622.
62. Kolb, M. and Thiel, W. (1993) Beyond the MNDO model: methodical considerations and numerical results. *J. Comput. Chem.*, **14**, 775–789.
63. Weber, W. and Thiel, W. (2000) Orthogonalization corrections for semiempirical methods. *Theor. Chem. Acc.*, **103**, 495–506.
64. Scholten, M. (2003) Semiempirische Verfahren mit Orthogonalisierungskorrekturen: Die OM3 Methode. PhD thesis. Universität Düsseldorf, Düsseldorf.
65. Otte, N., Scholten, M. and Thiel, W. (2007) Looking at self-consistent-charge density functional tight binding from a semiempirical perspective. *J. Phys. Chem. A*, **111**, 5751–5755.
66. Korth, M. and Thiel, W. (2011) Benchmarking semiempirical methods for thermochemistry, kinetics, and noncovalent interactions: OM*x* methods are almost as accurate and robust as DFT-GGA methods for organic molecules. *J. Chem. Theory Comput.*, **7**, 2929–2936.
67. Silva-Junior, M.R. and Thiel, W. (2010) Benchmark of electronically excited states for semiempirical methods: MNDO, AM1, PM3, OM1, OM2, OM3, INDO/S, and INDO/S2. *J. Chem. Theory Comput.*, **6**, 1546–1564.
68. Gargaro, A.R., Bloomberg, G.B., Dempsey, C.E., Murray, M. and Tanner, M.J.A. (1994) The solution structures of the first and second transmembrane-spanning segments of band 3. *Eur. J. Biochem.*, **221**, 445–454, PDB ID: 1BTQ.
69. O'Neill, J.W., Kim, D.E., Johnsen, K., Baker, D. and Zhang, K.Y. (2001) Single-site mutations induce 3D domain swapping in the B1 domain of protein L from Peptostreptococcus magnus. *Structure*, **9**, 1017–1027, PDB ID: 1K50.
70. Rubini, M., Lepthien, S., Golbik, R. and Budisa, N. (2006) Aminotryptophan-containing barstar: structure-function tradeoff in protein design and engineering with an expanded genetic code. *Biochim. Biophys. Acta*, **1764**, 1147–1158, PDB ID: 2HXX.
71. Ciatto, C., Bahna, F., Zampieri, N., VanSteenhouse, H.C., Katsamba, P.S., Ahlsen, G., Harrison, O.J., Brasch, J., Jin, X., Posy, S., Vendome, J., Ranscht, B., Jessell, T.M., Honig, B. and Shapiro, L. (2010) T-cadherin structures reveal a novel adhesive binding mechanism. *Nat. Struct. Mol. Biol.*, **17**, 339–347, PDB ID: 3K6F.

72. Fedorov, A.A., Magnus, K.A., Graupe, M.H., Lattman, E.E., Pollard, T.D. and Almo, S.C. (1994) X-ray structures of isoforms of the actin-binding protein profilin that differ in their affinity for phosphatidylinositol phosphates. *Proc. Natl. Acad. Sci. U. S. A.*, **91**, 8636–8640, PDB ID: 1ACF.
73. Choi, J., Choi, S., Chon, J.K., Choi, J., Cha, M.-K., Kim, I.-H. and Shin, W. (2005) Crystal structure of the C107S/C112S mutant of yeast nuclear 2-Cys peroxiredoxin. *Proteins*, **61**, 1146–1149, PDB ID: 2A4V.
74. Vaaje-Kolstad, G., Bøhle, L.A., Gåseidnes, S., Dalhus, B., Bjørås, M., Mathiesen, G. and Eijsink, V.G. (2012) Characterization of the chitinolytic machinery of Enterococcus faecalis V583 and high-resolution structure of its oxidative CBM33 enzyme. *J. Mol. Biol.*, **416**, 239–254, PDB ID: 4A02.
75. Tsukazaki, T., Mori, H., Echizen, Y., Ishitani, R., Fukai, S., Tanaka, T., Perederina, A., Vassylyev, D.G., Kohno, T., Maturana, A.D., Ito, K. and Nureki, O. (2011) Structure and function of a membrane component SecDF that enhances protein export. *Nature*, **474**, 235–238, PDB ID: 3AQO.
76. Pulay, P. (1982) Improved SCF convergence acceleration. *J. Comput. Chem.*, **3**, 556–560.
77. Stewart, J.J.P., Császár, P. and Pulay, P. (1982) Fast semiempirical calculations. *J. Comput. Chem.*, **3**, 227–228.
78. Thiel, W. (2012) MNDO99 CVS Development Version. Tech Rep, Mülheim an der Ruhr, Germany.
79. Dongarra, J., Dong, T., Gates, M., Haidar, A., Tomov, S. and Yamazaki, I. (2012) MAGMA: A new generation of linear algebra libraries for GPU and multicore architectures.
80. Häser, M. and Ahlrichs, R. (1989) Improvements on the direct SCF method. *J. Comput. Chem.*, **10**, 104–111.
81. Haidar, A., Solcà, R., Gates, M., Tomov, S., Schulthess, T. and Dongarra, J. (2013) Leading edge hybrid multi-GPU algorithms for generalized eigenproblems in electronic structure calculations, in *Supercomputing*, Lecture Notes in Computer Science, vol. **7905** (eds J. Kunkel, T. Ludwig and H. Meuer), Springer-Verlag, Berlin, Heidelberg, pp. 67–80.
82. Rohr, D., Bach, M., Kretz, M. and Lindenstruth, V. (2011) Multi-GPU DGEMM and high performance Linpack on highly energy-efficient clusters. *IEEE Micro*, **31**, 18–27.
83. Spiga, F. and Girotto, I. (2012) phiGEMM: a CPU-GPU library for porting quantum ESPRESSO on hybrid systems. Proceeding of 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2012), pp. 368–375.
84. Bakowies, D. and Thiel, W. (1991) MNDO study of large carbon clusters. *J. Am. Chem. Soc.*, **113**, 3704–3714.
85. Kulik, H.J., Luehr, N., Ufimtsev, I.S. and Martínez, T.J. (2012) *Ab initio* quantum chemistry for protein structures. *J. Phys. Chem. B*, **116**, 12501–12509.
86. Laskowski, R.A., MacArthur, M.W., Moss, D.S. and Thornton, J.M. (1993) PROCHECK: a program to check the stereochemical quality of protein structures. *J. Appl. Crystallogr.*, **26**, 283–291.
87. Ramachandran, G.N., Ramakrishnan, C. and Sasisekharan, V. (1963) Stereochemistry of polypeptide chain configurations. *J. Mol. Biol.*, **7**, 95–99.
88. Morris, A.L., MacArthur, M.W., Hutchinson, E.G. and Thornton, J.M. (1992) Stereochemical quality of protein structure coordinates. *Proteins*, **12**, 345–364.
89. Burstein, K.Y. and Isaev, A.N. (1984) MNDO calculations on hydrogen bonds. Modified function for core-core repulsion. *Theor. Chim. Acta*, **64**, 397–401.
90. Goldblum, A. (1987) Improvement of the hydrogen bonding correction to MNDO for calculations of biochemical interest. *J. Comput. Chem.*, **8**, 835–849.
91. Möhle, K., Hofmann, H.-J. and Thiel, W. (2001) Description of peptide and protein secondary structures employing semiempirical methods. *J. Comput. Chem.*, **22**, 509–520.
92. Seabra, G.de.M., Walker, R.C. and Roitberg, A.E. (2009) Are current semiempirical methods better than force fields? A study from the thermodynamics perspective. *J. Phys. Chem. A*, **113**, 11938–11948.