JCTC Journal of Chemical Theory and Computation

Quantum Chemistry on Graphical Processing Units. 3. Analytical Energy Gradients, Geometry Optimization, and First Principles Molecular Dynamics

Ivan S. Ufimtsev and Todd J. Martinez*

Department of Chemistry, Stanford University, Stanford, California 94305

Received June 11, 2009

Abstract: We demonstrate that a video gaming machine containing two consumer graphical cards can outpace a state-of-the-art quad-core processor workstation by a factor of more than 180× in Hartree−Fock energy + gradient calculations. Such performance makes it possible to run large scale Hartree−Fock and Density Functional Theory calculations, which typically require hundreds of traditional processor cores, on a single workstation. Benchmark Born−Oppenheimer molecular dynamics simulations are performed on two molecular systems using the 3-21G basis set - a hydronium ion solvated by 30 waters (94 atoms, 405 basis functions) and an aspartic acid molecule solvated by 147 waters (457 atoms, 2014 basis functions). Our GPU implementation can perform 27 ps/day and 0.7 ps/day of *ab initio* molecular dynamics simulation on a single desktop computer for these systems.

Introduction

The idea of using graphical hardware for general purpose computing goes back over a decade.[1−3] Nevertheless, early attempts to use GPUs for scientific calculations were largely stymied by lack of programmability and low precision of the hardware. The recent introduction of the Compute Unified Device Architecture (CUDA) programming interface[4] and hardware capable of performing double precision arithmetic operations by Nvidia significantly simplified GPU programming and has triggered an increasing number of publications in different fields, such as classical molecular dynamics[5−9] and quantum chemistry.[10−15] Our initial implementation of two-electron repulsion integral evaluation algorithms[13] and the entire direct self-consistent field procedure on the GPU[14,15] demonstrated the large potential of graphical hardware for quantum chemistry calculations. In this article, we continue exploring the use of GPUs for quantum chemistry, including the calculation of analytical gradients for self-consistent field wave functions and the implementation of *ab initio* Born−Oppenheimer (BO) molecular dynamics. We compare the GPU performance results to the GAMESS[22] quantum chemistry package running on an Intel Core2 quad-core 2.66 GHz CPU workstation, a state-of-the-art desktop computing system. Our "machine to machine" rather than "GPU to CPU core" comparison provides a realistic estimate of the actual speedup that can be obtained in real calculations. Comparison of the total time required to calculate energies and gradients on these two platforms shows that the GPU workstation is 2 orders of magnitude faster than the CPU workstation. This allows us to carry out *ab initio* molecular dynamics of large systems at more than a thousand MD steps per day on a desktop computer.

Analytical Energy Gradient Implementation. The general formula for nuclear gradients follows directly from the expression for the Hartree−Fock energy including a term accounting for the basis set dependence on molecular geometry[16]

$$\nabla_A E_{HF} = \sum_{\mu\nu} D_{\mu\nu}(\nabla_A H_{\mu\nu}) - \sum_{\mu\nu} W_{\mu\nu}(\nabla_A S_{\mu\nu}) + \sum_{\mu\nu\lambda\sigma}\left(D_{\mu\nu}D_{\lambda\sigma} - \frac{1}{2}D_{\mu\lambda}D_{\nu\sigma}\right)[\nabla_A(\mu\nu|\lambda\sigma] \quad (1)$$

where A labels an atomic center, $W_{\mu\nu}$ is the energy weighted density matrix, $S_{\mu\nu}$ is the overlap matrix, $D_{\mu\nu}$ is the density matrix, and $[\mu\nu|\lambda\sigma]$ are two-electron repulsion integrals over primitive basis functions

* Corresponding author e-mail: todd.martinez@stanford.edu.

10.1021/ct9003004 CCC: $40.75
Published on Web 08/25/2009

$$[\mu\nu|\lambda\sigma] = \iint \chi_\mu(\vec{r}_1)\chi_\nu(\vec{r}_1)\frac{1}{r_{12}}\chi_\lambda(\vec{r}_2)\chi_\sigma(\vec{r}_2)d\vec{r}_1^3 d\vec{r}_2^3 \quad (2)$$

$$\chi_\mu(\vec{r}) = c_\mu(x - X_\mu)^{n_x}(y - Y_\mu)^{n_y}(z - Z_\mu)^{n_z} \times \exp(-\alpha_\mu(\vec{r} - \vec{R}_\mu)^2) \quad (3)$$

where $\vec{r} \equiv \{x,y,z\}$ is the electronic coordinate, $\vec{R}_\mu \equiv \{X_\mu,Y_\mu,Z_\mu\}$ is the position of the atomic center associated with the μth basis function, and α_μ determines the width of this basis function. The total angular momentum of χ_μ is given by the sum of the three integer parameters $l = n_x + n_y + n_z$ and is equal to 0, 1, 2, for *s*-, *p*-, *d*-type functions, etc. Our program currently supports only *s*- and *p*-functions, although we are working on implementation of higher orbital momentum functions. The first two sums in eq 1 require little work compared to the last one and are calculated on the CPU in our implementation. All calculations on the CPU are carried out in full double precision, while calculations on the GPU are carried out in single precision unless otherwise indicated. The last sum, which is evaluated on the GPU, combines Coulomb and exchange contributions and runs over only those $\mu\nu$ pairs where at least one of μ or ν is centered on atom *A*. As in our implementation of direct SCF,[14,15] we treat the Coulomb and exchange terms separately, generating all required two-electron integrals and gradients from scratch using the McMurchie-Davidson algorithm[17]

$$[\mu\nu|\lambda\sigma] = \sum_{pq} E_p^{\mu\nu} E_q^{\lambda\sigma}[\Lambda_p|\Lambda_q] \quad (4)$$

where Λ_p (Λ_q) is a Hermite Gaussian product centered on $\vec{R}^+_{\mu\nu}$ ($\vec{R}^+_{\lambda\sigma}$), and $E_p^{\mu\nu}$ ($E_q^{\lambda\sigma}$) are the expansion coefficients of the Cartesian Gaussian product $\mu\nu$ ($\lambda\sigma$) over the Hermite functions, i.e.

$$\chi_\mu(\vec{r} - \vec{R}_\mu)\chi_\nu(\vec{r} - \vec{R}_\nu) = \sum_p E_p^{\mu\nu}(\vec{R}^-_{\mu\nu})\Lambda_p(\vec{r} - \vec{R}^+_{\mu\nu}) \quad (5)$$

$$\vec{R}^+_{\mu\nu} = \frac{\alpha_\mu\vec{R}_\mu + \alpha_\nu\vec{R}_\nu}{\alpha_\mu + \alpha_\nu} \quad (6)$$

$$\vec{R}^-_{\mu\nu} = \vec{R}_\mu - \vec{R}_\nu \quad (7)$$

The nuclear gradients ∇_A and ∇_B can now be represented in the new variables $\vec{R}^+_{\mu\nu}$ and $\vec{R}^-_{\mu\nu}$

$$\nabla_A = \frac{\alpha_\mu}{\alpha_\mu + \alpha_\nu}\nabla_{R^+} + \nabla_{R^-} \quad (8)$$

$$\nabla_B = \nabla_{R^+} - \nabla_A \quad (9)$$

where *A* and *B* label the atomic centers of the functions χ_μ and χ_ν, respectively. In the following, we will use the notation $[p|q]$ as a shorthand to indicate the integral over Hermite functions in eq 4, i.e. $[\Lambda_p|\Lambda_q]$. Thus, the indices *p* and *q* are pair indices corresponding to $\mu\nu$ or $\lambda\sigma$, respectively. This notation follows that introduced earlier in a comprehensive article by Gill on two-electron integral generation.[18]

A. Coulomb Contribution. The Coulomb contribution to $\nabla_A E$ is generally given by

$$\nabla_A^{Coul} E_{HF} = \sum_{\mu\nu\lambda\sigma} D_{\mu\nu}D_{\lambda\sigma}[\nabla_A(\mu\nu)|\lambda\sigma] \quad (10)$$

Following established practice, we expand the Cartesian Gaussian primitive pair products over Hermite Gaussian basis functions and preprocess the density matrix elements accordingly[19,20]

$$\nabla_A^{Coul} E_{HF} = \nabla_A\left(\sum_p D_p \sum_q D_q[\Lambda_p|\Lambda_q]\right) \quad (11)$$

$$D_p = \sum_{\mu\nu\in p} E_p^{\mu\nu} D_{\mu\nu}, D_q = \sum_{\lambda\sigma\in q} E_q^{\lambda\sigma} D_{\lambda\sigma} \quad (12)$$

Substituting eq 8 into eq 11 leads to the final result for $\nabla_A^{Coul} E_{HF}$

$$\nabla_A^{Coul} E_{HF} = \sum_p \frac{\alpha_{\mu(p)}}{\alpha_{\mu(p)} + \alpha_{\nu(p)}} J'_p D_p + \sum_p J_p \nabla_{R^-} D_p \quad (13)$$

$$J_p = \sum_q D_q[p|q] \quad (14)$$

$$J'_p = \sum_q D_q[\nabla_{R^+} p|q] = -\sum_q D_q[\nabla_r p|q] \quad (15)$$

In the cases under consideration here (*s* and *p* basis functions and no special treatment of *sp*-superblocks), there is only one pair of indices $\mu\nu$ for a given *p*, and we indicate this with the notation $\mu(p)$ in eq 13. This dependence is suppressed for notational convenience in the following. We calculate J_p and J'_p on the GPU, while the preprocessing of eq 12 and the postprocessing of eq 13 are carried out on the CPU.

Evaluation of J_p and J'_p is performed in a way very similar to our GPU implementation of the Coulomb matrix formation algorithm.[15] Here we use the so-called 1T1PI mapping scheme where one GPU thread calculates one primitive integral (or a batch of integrals if higher than *s* angular momentum functions are involved). This has proved to be the best choice if all required quantities are calculated directly from primitive integrals.[13] The fundamental data organization used for calculating J_p and J'_p is represented in Figure 1, using the same notation as in our previous article.[15] The left and upper triangles represent the index-symmetry pruned lists of the ($[p|$, $[\nabla p|$), and $|q]$ pairwise quantities (PQs) with doubled off-diagonal terms. The PQs are first organized into three groups according to the total angular momentum of the pair products, *ss*, *sp*, or *pp*. Furthermore, the $[p|$ and $|q]$ lists within each angular momentum grouping are sorted according to their Schwartz upper bounds

$$[p|_{Schwartz} = |D_{\mu\nu}|^{\max}[p|p]^{1/2} \quad (16)$$

$$|q]_{Schwartz} = |D_{\lambda\sigma}|^{\max}[q|q]^{1/2} \quad (17)$$

where we also use the maximum Cartesian density matrix elements among all angular momentum functions in a batch.

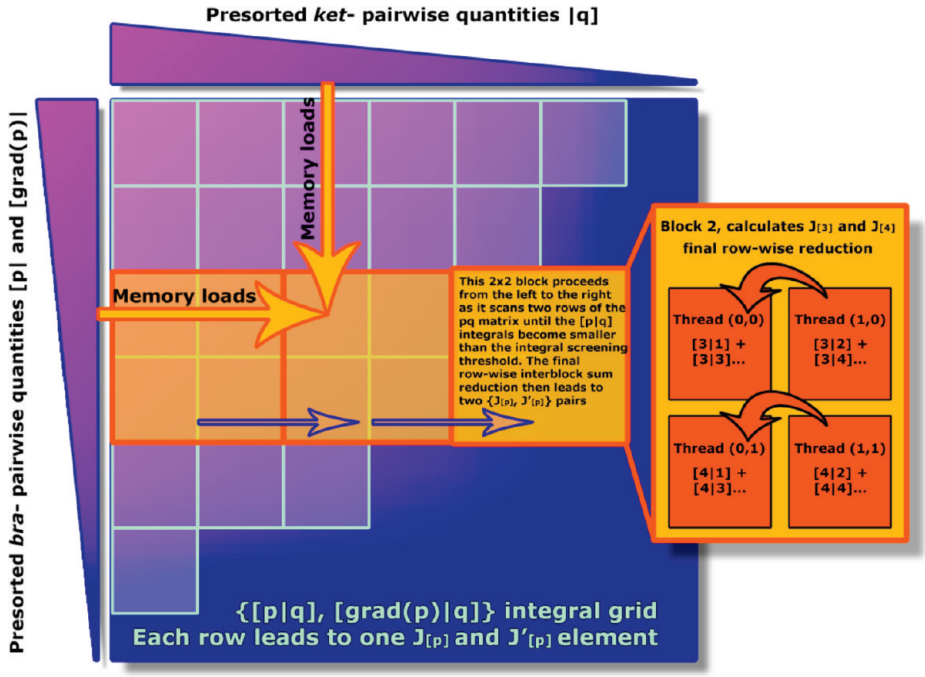

Figure 1. J_p and J'_p calculation algorithm. Each blue-green-bordered square represents four sets of primitive Hermitian integrals ($[p|q]$ and $[\nabla p|q]$), which need to be contracted with density matrix elements along a *p*-row in order to obtain J_p and J'_p. The upper and left triangles denote the *bra*- and *ket*-pair quantities, sorted according to their Schwartz upper bound (represented by the pink-blue coloration). A GPU 2 × 2 thread block is delineated by a large orange-bordered square comprised of four smaller yellow-bordered squares (GPU threads). The yellow arrows labeled "memory loads" represent elements of the [p| and |q] linear arrays loaded into local memory by the corresponding GPU threads. The blue arrows show the direction along which the block scans the two neighboring rows, leading to two $\{J_p, J'_p\}$ pairs of elements. The scan is aborted once each thread in a block encounters a $[p|q]$ integral whose Schwartz upper bound is smaller than 10^{-11} au.

Following the usual practice,[19] we calculate the Schwartz upper bound in eqs 16 and 17 assuming *p* and *q* have zero angular momentum. The grouping and sorting of the lists of PQs as well as the calculation of the relevant quantities is carried out on the CPU.

When combined, ($[p|$, $[\nabla p|$), and $|q]$ lead to the ($[p|q]$, $[\nabla p|q]$) integrals required by eqs 14 and 15. These are depicted by blue-green bordered squares in Figure 1 with one square denoting all ($[p|q]$, $[\nabla p|q]$) integrals computed simultaneously by one GPU thread. For example, there are $4 \times N_q$ such integrals ($[\Lambda_0|q]$, $[\Lambda_x|q]$, $[\Lambda_y|q]$, and $[\Lambda_z|q]$) if both μ and ν functions have zero angular momentum, and the $|q]$ batch has N_q PQs ($N_q = 1$, 4, 10 for *ss*-, *sp*-, *pp*-type batches, respectively). Because of the grouping by angular momentum, the resulting $[p|q]$ integral grid consists of nine rectangular segments of different integral types ($[ss|ss]$, $[ss|sp]$, and so on, up to $[pp|pp]$). Each of these integral grids is handled by a different GPU kernel, optimized for the specific angular momentum. For simplicity, only one such grid segment is presented in Figure 1. The pink-blue coloration in Figure 1 represents the density-weighted magnitude of the Schwartz bounds for the pair-quantities and the resulting $[p|q]$ integrals.

It can be easily seen from eqs 14 and 15 that J_p and J'_p can be computed by summing all integral contributions in one row *p*. Because we use the 1T1PI mapping, each blue-green bordered square in Figure 1 is mapped to a GPU thread depicted as a small orange square. For simplicity, four such GPU threads are depicted, organized into a 2 × 2 thread block. In practice, we use 8 × 8 thread blocks in the implementation. One such block then processes two rows of the integral matrix in column-by-column fashion, accumulating partial results in the corresponding GPU threads with double precision accuracy. Thus, the integrals and integral derivatives are computed in single precision, but the accumulation is done in double precision (on the GPU). As shown previously,[15] this procedure avoids unnecessary precision loss with minimal cost. After the block reaches integrals with a Schwartz upper bound smaller than 10^{-11} au, the scan is aborted, and subsequent intrablock row-wise sum reduction leads to two J_p and J'_p elements. The PQ presorting step guarantees that integrals omitted during the scan are even smaller than 10^{-11} au and thus can be safely disregarded. Because J_p is exactly the same quantity calculated in our previously described Coulomb matrix formation algorithm,[15] while J'_p merely adds some additional terms to be calculated, we simply modified the *J*-matrix GPU kernels to incorporate these additional terms.

B. Exchange Contribution. The exchange contribution

$$\nabla_A^{Exch} E_{HF} = -\frac{1}{2} \sum_{\mu\nu\lambda\sigma} D_{\mu\lambda} D_{\nu\sigma} [\nabla_A(\mu\nu)|\lambda\sigma] \qquad (18)$$

does not allow easy splitting of the work into the $D_p[p|$ and $|q]D_q$ product representation and thus requires different data organization. Therefore, we generate all required PQs and integrals from scratch without reusing any data from the Coulomb step. In addition, we do not preprocess the density matrix elements on CPU. Instead, the density matrix is preprocessed "on the fly" for each integral

$$\nabla_A^{Exch} E_{HF} = -\frac{1}{2}\nabla_A\left(\sum_p E_p^{\mu\nu}\sum_q D'_{pq}[\Lambda_p|\Lambda_q]\right) \quad (19)$$

$$D'_{pq} = E_q^{\lambda\sigma} D_{\mu\lambda} D_{\nu\sigma} \quad (20)$$

Substituting eq 8 into eq 19 leads to the final result for the exchange contribution to the energy gradient

$$\nabla_A^{Exch} E_{HF} = -\frac{1}{2}\left(\sum_p \frac{\alpha_\mu}{\alpha_\mu+\alpha_\nu} K'_p E_p^{\mu\nu} + \sum_p K_p \nabla_R E_p^{\mu\nu}\right) \quad (21)$$

$$K_p = \sum_q D'_{pq}[p|q], K'_p = -\sum_q D'_{pq}[\nabla_r p|q] \quad (22)$$

Unlike in the Coulomb energy gradient calculation, here only the $[\mu\nu|$ pair list is pruned according to $\mu\nu \leftrightarrow \nu\mu$ symmetry (doubling off-diagonal pairs as appropriate), while the $|\lambda\sigma]$ list contains all $O(N^2)$ pairs, where N is the total number of primitive Gaussian-type basis functions. This organization is dictated by the need to carry out prescreening of small integrals in a way that is commensurate with memory access and load balancing requirements on the GPU. Because the $|\lambda\sigma]$ list is not pruned by index-symmetry, there are four rather than three different angular momentum type $\lambda\sigma$ pairs – *ss*, *sp*, *ps*, and *pp*. Thus, the final integral grid contains twelve segments of different integral types (and requires twelve GPU kernels to handle them). Figure 2 provides more details on the GPU implementation of the exchange contribution to energy gradients. For simplicity, only one segment, e.g. *ss*|*ss*, is represented. Other segments are treated in the same way. We group $[\mu\nu|$ and $|\lambda\sigma]$ pairs according to the first index (μ or λ, respectively). This procedure leads to N blocks, each with fixed $[\mu_i. . .|$ and $|\lambda_i. . .]$. Because the $|\lambda\sigma]$ list is not index-symmetry pruned, all $|\lambda_i. . .]$ blocks contain N Gaussian pairs. The number of pairs in the $[\mu_i. . .|$ blocks varies (with a maximum of N) because the $[\mu\nu|$ list *is* index-symmetry pruned. All $[\mu_i. . .|$ and $|\lambda_i. . .]$ blocks are sorted according to the Schwartz upper bound of corresponding $[\mu_i\nu|$ and $|\lambda_i\sigma]$ pairs. Note that no density matrix information is used in this sorting step. These presorted $[\mu_i\nu|$ and $|\lambda_i\sigma]$ PQs are delineated by left and upper triangles in Figure 2, where the pink-blue coloration represents the Schwartz upper bound magnitude (as in Figure 1). After the PQs are sorted, K_p and K'_p are calculated on the GPU by a series of twelve subsequent GPU kernel calls (one for each angular momentum segment). Figure 2 provides details on the GPU implementation. Here, a 2×2 GPU thread block (we use 8×8 blocks in our program) is tasked to calculate two K_p and K'_p quantities by scanning two nearby rows of the integral grid and accumulating (in double precision, as in the Coulomb algorithm) the partial results in the GPU registers. During the scan, each thread monitors the product of Schwartz integral upper bound and the maximum Cartesian density matrix elements among all angular momentum functions in a batch, i.e.

$$([p|p][q|q])^{1/2}|D_{\mu\nu}|^{\max} \quad (23)$$

When this product becomes smaller than 10^{-11} au, a GPU thread aborts processing of the $|\lambda_i. . .]$ row and resumes from segment $|\lambda_{i+1}. . .]$. The fact that $[p|$ and $|q]$ pairs are organized into $|\mu_i. . .]$ and $|\lambda_j. . .]$ blocks guarantees that for all threads in a GPU block the maximum density matrix element $|D_{\mu_i\lambda_j}|^{\max}$ is the same. Thus, all threads in a GPU block will abort the scan of the $|\lambda_i. . .]$ segment simultaneously, avoiding potential problems due to load misbalancing. After the scan is complete, the final result is obtained by intrablock sum reduction.

C. Multi-GPU Parallelization. The energy gradient code is parallelized using POSIX threads in order to use all available GPUs in a workstation. For both Coulomb and exchange contributions, the work is split into 8-row segments and mapped to the devices cyclically. Each GPU thus computes its portion of J_p, J'_p, K_p, and K'_p and sends the results back to the host CPU, where they are postprocessed and the final energy gradient is calculated. This model also seems well suited for implementation on a multiple GPU node cluster (using the MPI framework,[21] for example) since the work is distributed in such a way that each node has all the data required to calculate its portion of the integrals from the very beginning, avoiding expensive internode communication. Preliminary implementations support this conjecture, and work along these lines is in progress.

A series of tests performed on a system with two GeForce 295GTX cards, each having two GPU processors, demonstrated reasonable speedup of $3.0-3.5\times$, relative to a single GPU processor. Reported timings include the time required to calculate and sort the pair-quantities, which is currently performed on a single CPU core, as well as the data transfer time required to copy the PQs to the GPU and then copy the results back to the CPU.

Results and Discussion

To assess the performance of the GPU code, we carried out a series of benchmarks on a representative set of test molecules and compared the results to GAMESS[22] ver. 11 Apr 2008 (R1). The GAMESS code was executed on an Intel Core2 quad-core 2.66 GHz CPU with 8GB main memory, which represents a state-of-the-art desktop computing system. All four CPU cores were used in parallel in order to obtain maximum CPU performance. The GAMESS program was compiled with the GNU Fortran compiler and linked with Intel MKL ver. 10.0.3. Our GPU code ran on the same workstation with two Nvidia GeForce 295GTX cards operating in parallel. All performance results in this article correspond to this "workstation to workstation" comparison rather than "a single GPU to a single CPU core" comparison. This provides a realistic assessment of the real performance gain one can obtain from a GPU system. For brevity, this quad-core CPU machine and the dual-GPU machine are referred to as "CPU" and "GPU", respectively.

Table 1 presents the time required to calculate the Hartree−Fock energy gradient vector for caffeine ($C_8N_4H_{10}O_2$), cholesterol ($C_{27}H_{46}O$), buckyball (C_{60}), taxol ($C_{45}NH_{49}O_{15}$), valinomycin ($C_{54}N_6H_{90}O_{18}$), and olestra ($C_{156}H_{278}O_{19}$) molecules using the 3-21G basis set. Among these test systems, the largest has 2131 basis functions. One can see that even for a small molecule such as caffeine, the GPU outperforms the CPU by a factor of

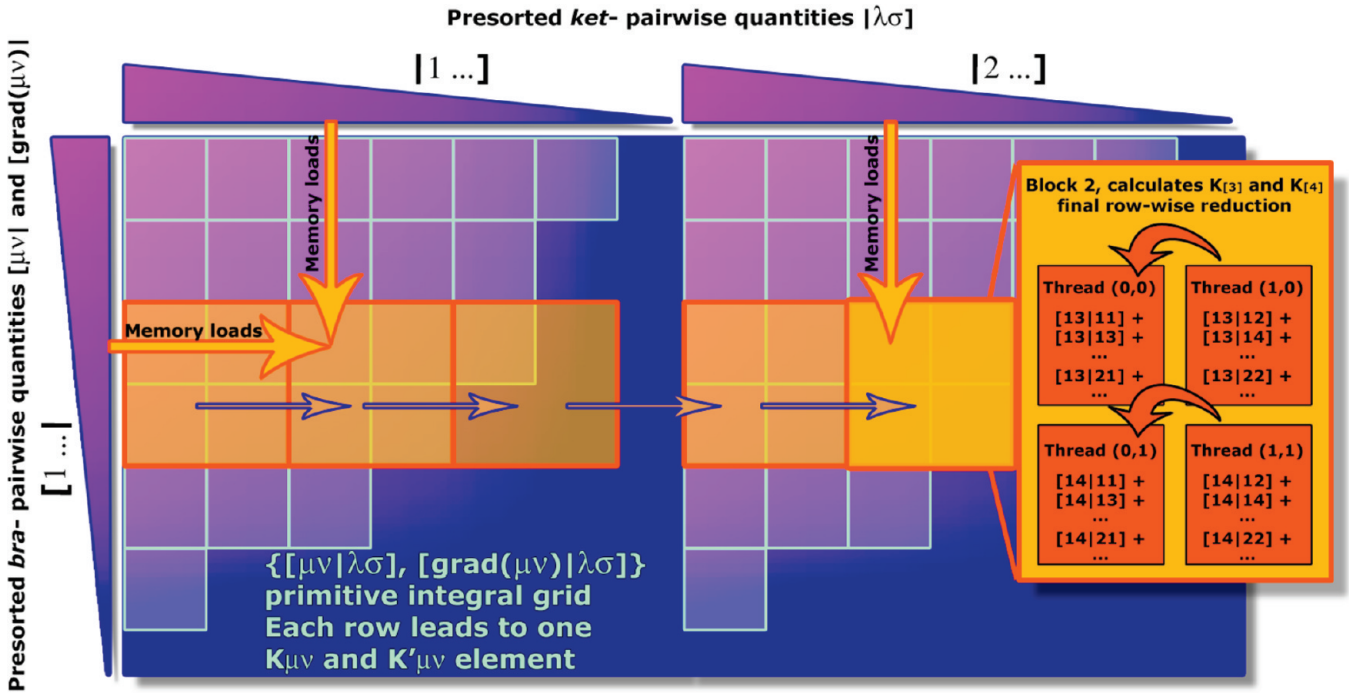

Figure 2. $K_{\mu\nu}$ and $K'_{\mu\nu}$ calculation algorithm. Similar to the *J* calculation shown in Figure 1, but in this case all *ket*-pairs are grouped into *N* segments according to the λ_i index. Two such segments are represented. The GPU thread block scans all the segments sequentially. Once integrals which make contributions smaller than 10^{-11} au are reached, the scan of the particular segment is aborted, and the block proceeds from the next segment.

Table 1. Analytical Energy Gradient Accuracy and Calculation Time for Different Test Molecules Using the 3-21G Basis Set

molecule	caffeine	cholesterol	C_{60}	taxol	valinomycin	olestra
CPU, sec[a]	0.7	7.6	38.5	28.9	66.3	785.1
GPU, sec[b]	0.1	0.4	1.9	1.4	2.6	7.3
Speedup	5.8	19	20	21	26	108
rms_{error}/ 10^{-5} au[c]	1.28	0.61	2.79	1.49	1.24	0.67

[a] GAMESS on Intel Core2 quad-core 2.66 GHz CPU. [b] 2 Nvidia GeForce 295GTX cards. [c] Error in the gradient (atomic units) as defined by eq 24.

Table 2. Total Energy and Gradient Computation Time Using the 3-21G Basis Set[a]

molecule	caffeine	cholesterol	buckyball	taxol	valinomycin	olestra
CPU, sec[b]	7.4	81.8	364	435	1112	22863
GPU, sec[b]	2.0	4.8	11.6	15.7	25.5	125
Iterations	13	11	10	14	13	14
Speedup	3.7	17	31	28	44	182

[a] The number of SCF iterations performed to converge the wave function was exactly the same for GPU and CPU. [b] CPU and GPU are the same machines as in Table 1.

6×. For medium-size molecules the speedup ranges between 20× and 25×, while for large molecules it exceeds 100×. In addition, in Table 2 we present the total program execution time, energy + gradient, for the same set of molecules. In both CPU and GPU calculations the total number of SCF iterations required to converge the wave function was exactly the same, although it differed among the molecules. The resulting speedups range from 4× for small-, 30×−40× for medium-, and up to almost 200× for the largest molecules. Clearly, these results represent a mixture of different coding styles, compiler efficiencies, and hardware architectures. However, it is also obvious that such speedups would never be achievable without the impressive performance gain provided by the GPU, which enables desktop calculation of *ab initio* geometry optimization and molecular dynamics (MD) simulations that were previously only possible on computing clusters with more than a hundred CPUs.

The increased performance of the GPU code clearly improves the quality of molecular dynamics simulation results by allowing longer runs and thus better statistics. However, accuracy is another aspect that needs to be considered, especially when part of the energy and atomic force calculations are performed with single precision. To assess the error introduced by the use of single precision for integral evaluation, we performed three independent tests.

First, we directly calculated the root mean squared error for all components of corresponding single precision (GPU) and double precision (CPU) gradient vectors

$$RMS_{error} = \sqrt{\sum_{i=1}^{3N_{Atoms}} \left(f_i^{CPU} - f_i^{GPU}\right)^2 / 3N_{Atoms}} \qquad (24)$$

for all the benchmark molecules. All molecular geometries along with corresponding energies and gradients, calculated on CPU and GPU, are provided in the Supporting Information. The results, presented in Table 1, demonstrate that the mean error is distributed around 10^{-5} au, which is close to the typical convergence thresholds used in geometry optimization algorithms. It is also important that there is no obvious correlation between the mean error and the size of a molecule, i.e. the error does not increase with the number of atoms.

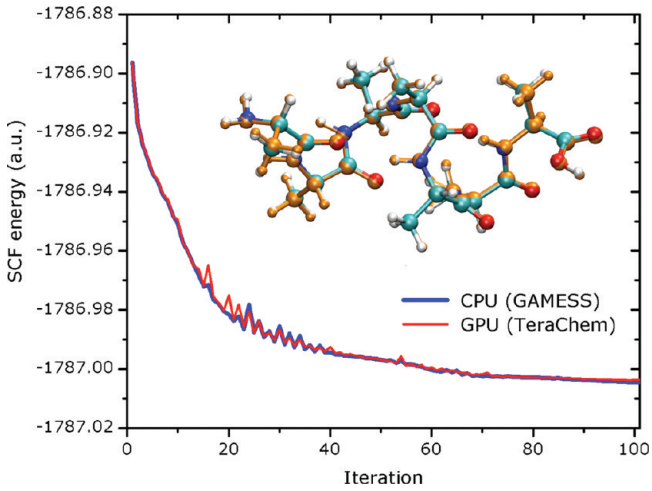

Figure 3. Evolution of the $(ALA)_7$ helix SCF energy during geometry optimization on the CPU (blue) and GPU (red) using 3-21G basis set. Both curves follow each other very closely, demonstrating that the GPU accuracy is sufficient for solving geometry optimization problems. The final energy difference is 0.50 kcal/mol, of which 0.08 kcal/mol is due to lower accuracy of single point energy calculations on GPU and 0.42 kcal/mol is due to different atomic positions. The inset portrays the two optimized structures overlaid on top of each other (orange: CPU, multicolored: GPU).

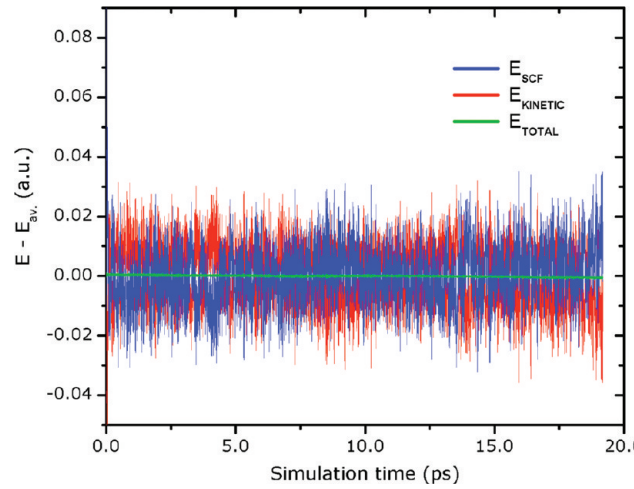

Figure 4. The SCF (blue), kinetic (red), and total (green) energies of the $H_3O^+(H_2O)_{30}$ cluster during microcanonical ($<E_{kin}>$ 301 K) Born−Oppenheimer Hartree−Fock molecular dynamics simulation using the 3-21G basis set on two Nvidia GeForce 295GTX GPUs. Two electron integrals and their derivatives are calculated on the GPU in single precision, and their contributions are accumulated in double precision. The total energy drift is 0.022 kcal/mol·ps^{-1}, which corresponds to a 0.039 K·ps^{-1} averaged temperature drift.

Second, we carried out geometry optimization of a helical hepta-alanine, i.e. $(ALA)_7$, peptide on the CPU and GPU using the 3-21G basis set. The corresponding SCF energy evolution and resulting structures from GAMESS and our GPU code are compared in Figure 3. In both cases, the same initial geometry and energy minimization algorithm was used (trust region method with BFGS Hessian update), although our implementation was written from scratch and therefore the codes are not necessarily identical. However, we did employ exactly the same trust radius update protocol as implemented in GAMESS. It can be easily seen that the curves follow each other very closely throughout the whole optimization procedure, slightly diverging at the very end. This is a good indicator that GPU can be efficiently used for solving molecular geometry optimization problems. The final energy discrepancy is 0.50 kcal/mol, of which 0.42 kcal/mol is due to different atomic configurations and the rest is due to lower accuracy of single point energy calculations on GPU. In addition, the optimized structures overlap almost perfectly, as shown in the inset of Figure 3, where the CPU-optimized and GPU-optimized structures are portrayed in orange and multicolored representations, respectively.

Finally, conservation of total energy is a common metric for assessing the energy gradient accuracy in a dynamics algorithm. Therefore, we performed time-reversible[23] Hartree−Fock Born−Oppenheimer molecular dynamics simulation of an $H_3O^+(H_2O)_{30}$ cluster using the 6-31G basis set and microcanonical ensemble. The Newtonian equations of motions were integrated using the velocity Verlet algorithm with a 0.5 fs time step for a total simulation time of 20 ps. Figure 4 shows the resulting time evolution of the kinetic (red), potential (blue), and total (green) energies. Energy is conserved quite well, with a small 0.022 kcal/mol·ps^{-1} total energy drift. Considering, for example, an upper limit of 5% of the initial kinetic energy as a maximum acceptable total energy drift over the entire simulation, this system can be simulated for 200 ps. Even longer time scales can be accessed if one introduces Langevin thermostats and careful adjustment of the damping parameter,[24] although care needs to be taken in this case to ensure that the dynamics is not significantly modified by the damping parameter. To further demonstrate that GPU-based *ab initio* molecular dynamics can treat important phenomena of widespread chemical interest such as proton transfer, we performed AIMD simulations of two systems: a $H_3O^+(H_2O)_{30}$ cluster and a neutral aspartic acid molecule solvated by 147 water molecules.

A. Protonated Water Cluster. A 20-ps long NVT (T = 300 K) molecular dynamics simulation was performed on the $H_3O^+(H_2O)_{30}$ cluster at the RHF/6-31G, BLYP/6-31G, and B3LYP/6-31G levels using a 0.5 fs integration time step. The density functional theory calculations (energy and nuclear energy gradient), which run entirely on GPU and use all available GPU processors in parallel, were recently incorporated into TeraChem, the general purpose GPU-based quantum chemistry package being developed in our group. All three simulations started from an initial cluster geometry where the hydronium ion was located on the cluster surface.

Figure 5 shows the time evolution of the distance between the hydronium ion and the cluster center along with the mean cluster radii represented by dashed lines. The cluster radius was defined as the maximum distance between any of the O-atoms and the cluster center. In all three cases, the ion stayed close to the surface throughout the entire simulation, as previously reported.[25,26] Somewhat larger oscillations of the distance in the RHF

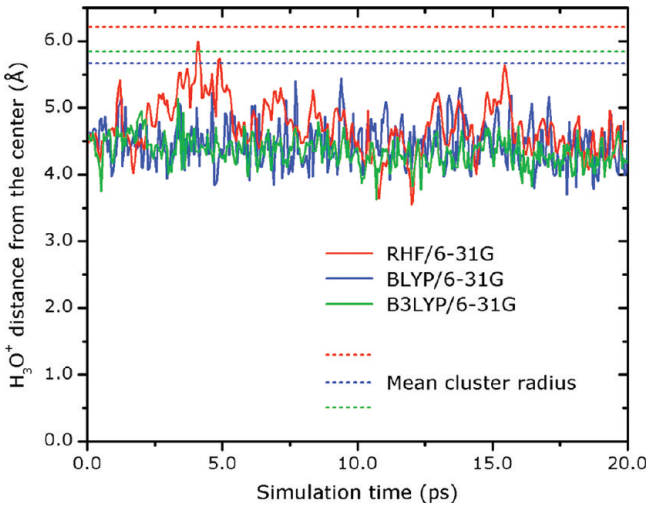

Figure 5. Time evolution of the H_3O^+ ion distance from the cluster center. Each point on the plot is averaged over 100 successive MD steps. The dashed lines represent the cluster radius, averaged over the whole MD simulation run. The radius is defined as the maximum distance between an O-atom and the center of the cluster.

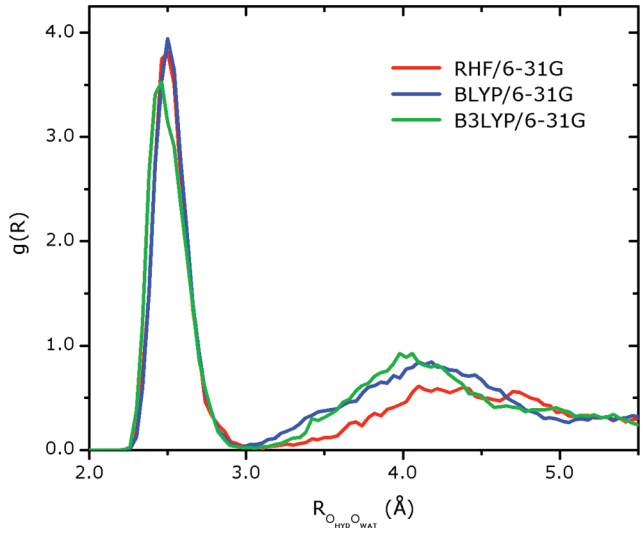

Figure 6. The hydronium-water oxygen−oxygen radial distribution function for the $H_3O^+(H_2O)_{30}$ cluster, using various levels of theory and the 6-31G basis set.

simulation, compared to the DFT results, show that at the HF level of theory the structure of the cluster surface is subject to higher fluctuations. In addition, RHF and BLYP provide similar structures of the first H_3O^+ solvation shell, as can be seen from the oxygen−oxygen radial distribution functions presented in Figure 6 (red and blue lines, respectively). In both cases, the first peak is centered at 2.50 Å, which is smaller than the same value for bulk water (2.8 Å) and is an expected signature of stronger H-bonds. The B3LYP simulation (green line in Figure 6) predicts a somewhat more diffuse first solvation shell, with the first peak centered at 2.46 Å. In addition, the B3LYP RDF reveals stronger bimodal character around the maximum due to continuous interplay between Eigen and Zundel structures. In all three simulations there was an average of three water molecules in the first solvation shell of the hydronium ion.

B. Solvated Aspartic Acid. We also performed a 7-ps simulation of a single neutral aspartic acid molecule solvated by a cluster of 147 water molecules using the 3-21G basis set. The total number of atoms and basis functions in this system was 457 and 2014, respectively. The $C_4NH_7O_4$ molecule was first solvated in a cube of water and equilibrated for 100 ps using the CHARMM force field[27] and periodic boundary conditions. Then, a 10 Å-sphere was sketched around the C_α atom, and those waters whose O-atoms were located inside this sphere (147 molecules) were selected for further modeling with AIMD. The resulting system was equilibrated for 1 ps using AIMD. In both classical and *ab initio* equilibration MD simulations, a 0.5 fs integration time step and the Langevin thermostat[28] ($T = 300$ K, $\tau_{damp} = 1$ ps) were used, and the atoms in the aspartic acid molecule were fixed in order to prevent deprotonation of the carboxylic groups. Finally, a 7-ps production AIMD run was performed on the system in the NVT ensemble using a 0.5 fs integration time step. Figure 7 portrays a snapshot of the system (left panel) along with its highest occupied molecular orbital (HOMO, right panel). The orbitals were calculated by the GPU-accelerated Orbital plugin implemented in VMD.[29]

The aspartic acid molecule has two carboxylic acid functional groups - backbone and side chain, referred to as COOHbb and COOHsc in the following. Because both the groups have low pK_a, one might expect them to deprotonate quickly in aqueous solution. The AIMD simulation mostly confirms the expectations. Figure 8 displays the time evolution of the O−H distance for both COOH groups. One of these (COOHsc, blue line in Figure 8) quickly deprotonates in approximately 400 fs through formation of a short-lived ($\tau \sim 50$ fs) transient Zundel-like structure. The resulting hydronium ion then quickly (within 50 fs) shuttles to the surface of the cluster via two subsequent proton transfer events and stays at the surface for the rest of the simulation. The second deprotonation event (of the COOHbb group) does not occur until significantly later ($t \approx 5.5$ ps) in the simulation, presumably because of the high proton affinity of the resulting doubly negatively charged amino acid ion. An aborted attempt at deprotonation of COOHbb by forming a quasi-stable $COO^- \cdots H_3O^+$ complex is observed after 3.1 ps of simulation (green line in Figure 8). The presence of positive counterions near the molecule (not accounted for in our simulation) would be expected to facilitate faster deprotonation of all carboxylic groups. The carboxyl-water $O_{carboxyl}$-O_w (including both oxygen atoms of each of the COOHsc and COOHbb groups) and amino-water N_{amino}-O_w radial distribution functions, which are proportional to the local water density around these groups, are presented in Figure 9 and provide details on the solvent structure around these functional groups. The $O_{carboxyl}$-O_w RDF (red line in Figure 9) has its first maximum centered at 2.69 Å, and integration to the first minimum of the RDF reveals that on average 2.5 water molecules are present in the first solvation shell of each carboxyl oxygen. In

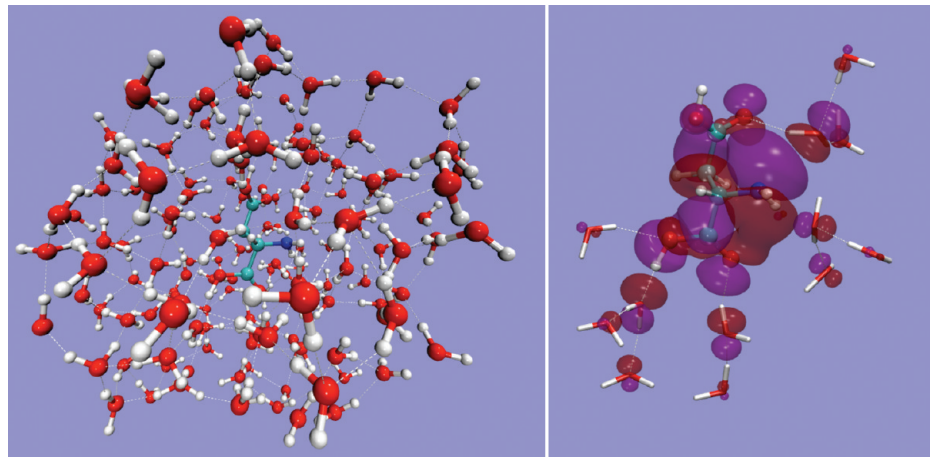

Figure 7. Left: snapshot of the AIMD simulation of the aspartic acid molecule solvated by 147 waters. Right: HOMO along with nearby water molecules among which the orbital is mostly delocalized. The isosurfaces correspond to $\varphi = \pm 0.01$ au .

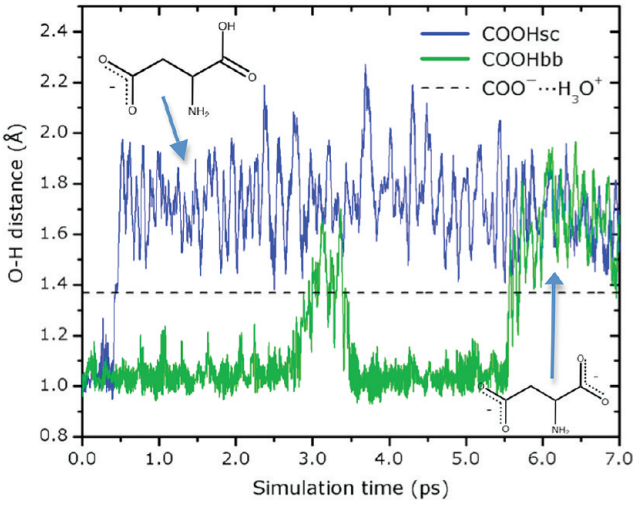

Figure 8. The O−H distance in backbone (COOHbb, green) and side chain (COOHsc, blue) carboxylic acid functional groups. The dashed line represents the O−H distance corresponding to the $COO^{-}\cdots H_3O^{+}$ complex.

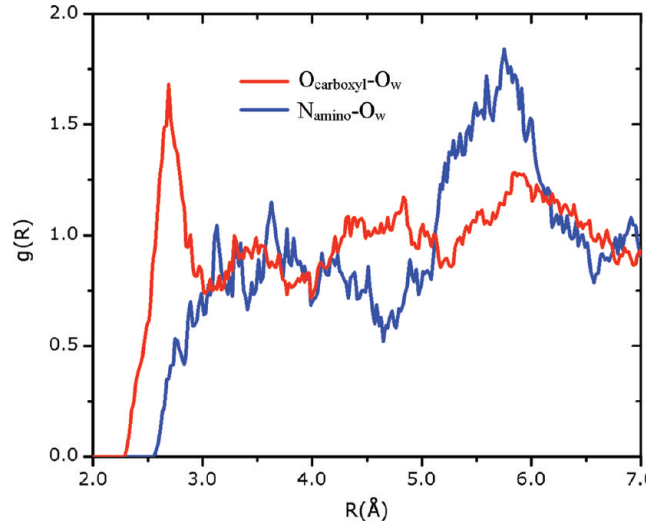

Figure 9. The aspartic acid oxygen (red) and nitrogen (blue) - water oxygen radial distribution functions. The neutral amino group reveals prominent hydrophobic character.

contrast, the N_{amino}-O_w RDF (blue line in Figure 9) indicates a rather hydrophobic character of the amino group, although the group periodically accepts and donates weak H-bonds. In many cases, however, the solvent water forms a hydrophobic cage around NH_2, where waters prefer to donate/accept H-bonds to/from neighboring waters. Those water molecules that sometimes do accept an H-bond from the amino group, in most cases become 5-coordinated. Because such a water solvation structure is less energetically favorable than the 4-coordinated tetrahedral configuration, these water molecules tend to break such H-bonds. Quantitative analysis of the trajectory demonstrates that the amino group donates 0, 1, and 2 H-bonds for 54%, 45%, and 1% of the simulation time, respectively. Similarly, it accepts 0 and 1 H-bonds for 81% and 19% of the simulation time. An H-bond was defined using 3.2 Å O−O distance and 30° (O/N)O_wH_w angle cutoffs. The character of the distribution of H-bonds donated by NH_2 does not vary much during the simulation which accesses three different charge states of the aspartic acid (0, −1, and −2). However, the amino group reveals stronger hydrophilic properties at the −2 charge state by stabilizing the accepted H-bond and reducing the mean N−O_w distance to the nearest water molecule from 3.1 Å (charge state 0 and −1) to 2.8 Å (charge state −2). Figure 10 shows the time evolution of the N−O_w distance, where the COOH group deprotonation events (for COOHsc and COOHbb) are marked by red vertical lines. The black dashed line denotes the mean N−O_w distance for all (0, −1, and −2) charge states of the aspartic acid molecule before and after the deprotonation of COOHbb. For both neutral and singly negative charge states the distance oscillates around 3.1 Å, but it drops to 2.8 Å after COOHbb is deprotonated and the molecule becomes doubly negatively charged. The hydrophobic-like behavior of the amino group in the partially deprotonated aspartic acid molecule does not seem to preclude protonation. In a preliminary simulation (not shown), the NH_2 residue is rapidly protonated if there is a hydronium ion in its first solvation shell. However, we leave detailed analysis of this for future, more extensive, studies.

Conclusions

We have demonstrated that it is possible to achieve up to 200× speedup in energy + gradient calculations by redesigning quantum chemistry algorithms for the GPU.

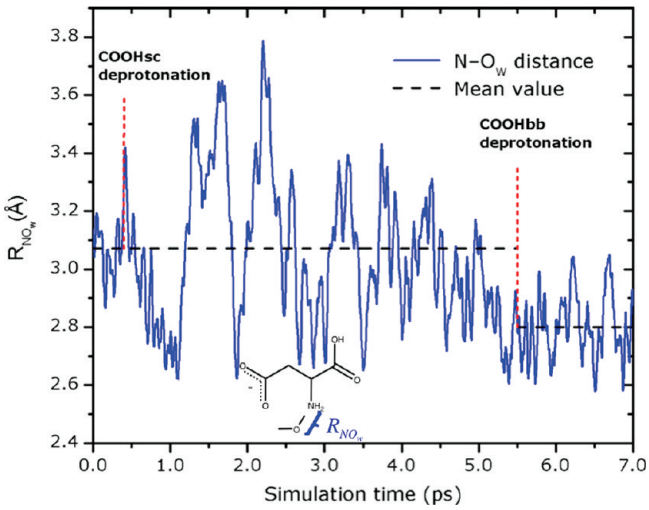

Figure 10. Time evolution of the NO_w distance between the amino nitrogen and the nearest water molecule. Deprotonation events of the carboxylic acid group are marked by red vertical lines. The horizontal dashed lines represent the mean NO_w distance for all (0, −1, and −2) charge states of the aspartic acid molecule. The mean distance is essentially the same (∼3.1 Å) for 0 and −1 charge states but suddenly drops to 2.8 Å as the backbone carboxylic acid group is deprotonated.

The performance gain was assessed by comparing a state-of-the-art Intel Core2 quad-core 2.66 GHz CPU workstation with the same workstation containing two Nvidia GeForce 295GTX graphical cards. Both Hartree−Fock and density functional theory (including generalized gradient and hybrid functionals with exact exchange) electronic structure methods can be used in AIMD simulations with our implementation. The remarkable speedups attained by executing all computationally intensive parts of the code on the GPU rather than on the CPU make it possible to carry out *ab initio* molecular dynamics simulation of large systems containing more than 2000 basis functions at 1400 MD steps/day speed on a single desktop computer.

We show in an example $H_3O^+(H_2O)_{30}$ cluster MD simulation that total energy drift due to limited hardware precision is minor and 100 ps MD simulations can be performed with total energy conservation errors of less than a few percent of the initial kinetic energy. Furthermore, even this minor drift can be compensated by employing a Langevin thermostat and properly adjusting the damping parameter,[24] meaning that even longer *ab initio* MD runs can be performed on the GPU.

We have presented results for preliminary AIMD simulations of proton transfer and transport in solvated clusters that were facilitated by the developments described here. Further study of these phenomena, collecting significant statistics, is underway.

Acknowledgment. This work was supported by the National Science Foundation (CHE-06-26354). TeraChem development was carried out at the University of Illinois. I.S.U. is an Nvidia fellow.

Supporting Information Available: Cartesian coordinates and SCF energies and gradients computed using the CPU (double precision) and GPU (mixed precision) for test molecules listed in Table 1. This material is available free of charge via the Internet at http://pubs.acs.org.

References

(1) Lengyel, J.; Reichert, M.; Donald, B. R.; Greenberg, D. P. *Comput. Graph.* **1990**, *24*, 327.

(2) Bohn, C. A. *Joint Conference on Intelligent Systems 1999 (JCIS'98)*; 1998; Vol. 2, p 64.

(3) Hoff, K. E., II; Culver, T.; Keyser, J.; Ming, L.; Manocha, D. *Computer Graphics Proceedings. SIGGRAPH 99*; 1999; p 277.

(4) NVIDIA CUDA. Compute Unified Device Architecture Programming Guide Version 2.2. http://www.nvidia.com/object/cuda_develop.html (acccessed May 15, 2009).

(5) Stone, J. E.; Phillips, J. C.; Freddolino, P. L.; Hardy, D. J.; Trabuco, L. G.; Schulten, K. *J. Comput. Chem.* **2007**, *28*, 2618.

(6) Anderson, J. A.; Lorenz, C. D.; Travesset, A. *J. Comp. Phys.* **2008**, *227*, 5342.

(7) Liu, W. G.; Schmidt, B.; Voss, G.; Muller-Wittig, W. *Comput. Phys. Commun.* **2008**, *179*, 634.

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

(9) Giupponi, G.; Harvey, M. J.; De Fabritiis, G. *Drug Discovery Today* **2008**, *13*, 1052.

(10) Yasuda, K. *J. Comput. Chem.* **2008**, *29*, 334.

(11) Yasuda, K. *J. Chem. Theory Comput.* **2008**, *4*, 1230.

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

(13) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2008**, *4*, 222.

(14) Ufimtsev, I. S.; Martinez, T. J. *Comput. Sci. Eng.* **2008**, *10*, 26.

(15) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2009**, *5*, 1004.

(16) Pulay, P. *Mol. Phys.* **1969**, *17*, 197.

(17) McMurchie, L. E.; Davidson, E. R. *J. Comp. Phys.* **1978**, *26*, 218.

(18) Gill, P. M. W. *Adv. Quantum Chem.* **1994**, *25*, 141.

(19) Almlof, J.; Faegri, K.; Korsell, K. *J. Comput. Chem.* **1982**, *3*, 385.

(20) Ahmadi, G. R.; Almlof, J. *Chem. Phys. Lett.* **1995**, *246*, 364.

(21) Gropp, W.; Lusk, E.; Skjellum, A. *Using MPI: Portable Parallel Programming with the Message Passing Interface*, 2nd ed.; MIT Press: Cambridge, 1999.

(22) Schmidt, M. W.; Baldridge, K. K.; Boatz, J. A.; Elbert, S. T.; Gordon, M. S.; Jensen, J. H.; Koseki, S.; Matsunaga, N.; Nguyen, K. A.; Su, S. J.; Windus, T. L.; Dupuis, M.; Montgomery, J. A. *J. Comput. Chem.* **1993**, *14*, 1347.

(23) Niklasson, A. M. N.; Tymczak, C. J.; Challacombe, M. *Phys. Rev. Lett.* **2006**, *97*, 4.

(24) Kuhne, T. D.; Krack, M.; Parrinello, M. *J. Chem. Theory Comput.* **2009**, *5*, 235.

(25) Buch, V.; Milet, A.; Vacha, R.; Jungwirth, P.; Devlin, J. P. *Proc. Natl. Acad. Sci. U. S. A.* **2007**, *104*, 7342.

(26) Petersen, M. K.; Iyengar, S. S.; Day, T. J. F.; Voth, G. A. *J. Phys. Chem. B* **2004**, *108*, 14804.

(27) MacKerell, A. D.; Bashford, D.; Bellott, M.; Dunbrack, R. L.; Evanseck, J. D.; Field, M. J.; Fischer, S.; Gao, J.; Guo, H.; Ha, S.; Joseph-McCarthy, D.; Kuchnir, L.; Kuczera, K.; Lau, F. T. K.; Mattos, C.; Michnick, S.; Ngo, T.; Nguyen, D. T.; Prodhom, B.; Reiher, W. E.; Roux, B.; Schlenkrich, M.; Smith, J. C.; Stote, R.; Straub, J.; Watanabe, M.; Wiorkiewicz-Kuczera, J.; Yin, D.; Karplus, M. *J. Phys. Chem. B* **1998**, *102*, 3586.

(28) Schlick, T. *Molecular Modeling and Simulation*; Springer: New York, 2002.

(29) Humphrey, W.; Dalke, A.; Schulten, K. *J. Mol. Graph.* **1996**, *14*, 33.

CT9003004