JCTC Journal of Chemical Theory and Computation

Article

pubs.acs.org/JCTC

Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 2. Explicit Solvent Particle Mesh Ewald

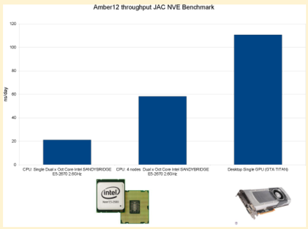
Romelia Salomon-Ferrer,[†] Andreas W. Götz,[†] Duncan Poole,[‡] Scott Le Grand,[‡,∥] and Ross C. Walker*,[†,§]

[†]San Diego Supercomputer Center, University of California, San Diego, 9500 Gilman Drive MC0505, La Jolla, California 92093, United States

[‡]NVIDIA Corporation, 2701 San Tomas Expressway, Santa Clara, California 95050, United States

[§]Department of Chemistry and Biochemistry, University of California, San Diego, 9500 Gilman Drive MC0505, La Jolla, California 92093, United States

S *Supporting Information*



ABSTRACT: We present an implementation of explicit solvent all atom classical molecular dynamics (MD) within the AMBER program package that runs entirely on CUDA-enabled GPUs. First released publicly in April 2010 as part of version 11 of the AMBER MD package and further improved and optimized over the last two years, this implementation supports the three most widely used statistical mechanical ensembles (NVE, NVT, and NPT), uses particle mesh Ewald (PME) for the long-range electrostatics, and runs entirely on CUDA-enabled NVIDIA graphics processing units (GPUs), providing results that are statistically indistinguishable from the traditional CPU version of the software and with performance that exceeds that achievable by the CPU version of AMBER software running on all conventional CPU-based clusters and supercomputers. We briefly discuss three different precision models developed specifically for this work (SPDP, SPFP, and DPDP) and highlight the technical details of the approach as it extends beyond previously reported work [Götz et al., *J. Chem. Theory Comput.* **2012**, DOI: 10.1021/ct200909j; Le Grand et al., *Comp. Phys. Comm.* **2013**, DOI: 10.1016/j.cpc.2012.09.022].We highlight the substantial improvements in performance that are seen over traditional CPU-only machines and provide validation of our implementation and precision models. We also provide evidence supporting our decision to deprecate the previously described fully single precision (SPSP) model from the latest release of the AMBER software package.

1. INTRODUCTION

Classical molecular dynamics (MD) has been extensively used in atomistic studies of biological and chemical phenomena including the study of biological ensembles of proteins, amino acids, lipid bilayers, and carbohydrates.[1−13] With the development of new algorithms and the emergence of new hardware platforms, MD simulations have dramatically increased in size, complexity, and simulation length. In particular, graphics processing units (GPUs) have emerged as an economical and powerful alternative to traditional CPUs for scientific computation.[14−17] GPUs are present in most modern high-end desktops and are now appearing in the latest generation of supercomputers. When programmed correctly, software running on GPUs can significantly outperform that running on CPUs. This is due to a combination of high computational power, in terms of peak floating point operations, and high memory bandwidth. This combination makes GPUs an ideal platform for mathematically intense algorithms that can be expressed in a highly parallel way. On the downside, the inherent parallel nature of the GPU architecture necessitates a decrease in flexibility and an increase in programming complexity in comparison to CPUs.

The success and high demand for GPUs in the gaming and 3D image rendering industries has fueled the sustained development of GPUs for over two decades leading to extremely cost-effective hardware for scientific computations. The first GPU with features specifically targeted for scientific computation was released by NVIDIA in 2007 with a subsequent generation following a year later that provided the first support for double precision floating point arithmetic. At the time of writing, NVIDIA's latest generation of GPUs are based on the Kepler GK104 and GK110 chips. These two chip designs, similar to earlier models, provide very different ratios for single vs double precision performance. The GK104 is targeted at algorithms that rely extensively on single precision, while the GK110 offers more extensive double precision performance. As discussed later, it is necessary to carefully tune the use of single and double precision floating point and ultimately fixed precision arithmetic to achieve high performance across these different hardware designs while not compromising the integrity of the underlying mathematics.

There are a large number of scientific software packages that have been successfully ported to run on GPUs.[12,13,18] In the molecular dynamics field there have been attempts to port major MD packages to GPUs. For a review of the progress, the reader is referred to the review article in ref 12. A number of widely used MD packages designed for the simulation of condensed phase biological systems exist that feature varying degrees of GPU support including NAMD,[19,20] AMBER,[21,22]

Received: April 17, 2013
Published: August 6, 2013

GROMACS,[23,24] LAMMPS,[25] CHARMM,[26] aceMD,[27] and openMM.[28] Of these, one of the most widely used is the Assisted Model Building with Energy Refinement (AMBER) package, the latest version of which (v12) was released in April 2012 with comprehensive support for GPU-accelerated MD using NVIDIA graphics cards.[21,22] For a recent review of the methodology supported in the latest version of AMBER, the reader is directed to the review article in ref 22.

Part 1 of this paper[29] presented our AMBER GPU implementation of Generalized Born implicit solvent MD including details of the precision models used for floating point operations, calculation of covalent terms, SHAKE constraints, and the direct space sum. In this paper, we extend the previous work describing our complete implementation of explicit solvent MD using the PME approach to long-range electrostatics where the entire calculation is conducted on GPUs. In a condensed phase MD simulation, nonbonded interactions are the most numerous contributions to the force and energy with a formal $O(N^2)$ scaling. The calculation of these interactions is thus usually the bottleneck. To reduce the computational cost, the contributions of long-range interactions are usually approximated by schemes with more favorable scaling. Many methods have been devised for this purpose, most involving the use of a cutoff in which nonbonded interactions are only explicitly accounted for within the cutoff. This approximation plus a long-range correction term is usually used for van der Waals interactions (vdW).[30] Coulomb terms are poorly described by this method. These interactions decay very slowly and are not typically convergent with the summation of a finite length. The use of electrostatic cutoffs can also introduce undesirable effects including lack of energy conservation due to abrupt truncations. Approaches to account for contributions beyond the cutoff include the reaction field method,[31] isotropic periodic sum method,[32] and a number of lattice sum methods. The lattice sum methods are the most widely used in condensed matter biomolecular simulations and include the particle mesh Ewald (PME) method,[33] smooth particle mesh Ewald (SPME) method,[34] and particle−particle−particle mesh (P^3M)[35] method. PME is the method that has been used in AMBER simulations and is the approach we have implemented here.

Support for explicit solvent PME calculations on GPUs was first released publicly with AMBER v11 in April 2010. Version 11 was based on the work described here was initially developed to support the most commonly used features of the AMBER MD engine pmemd. The latest version of AMBER (v12) includes not only significant additions to the GPU supported features of pmemd but also introduces a completely new precision model SPFP[36] as discussed later. This uses fixed point integer arithmetic in place of double precision floating point accumulations to achieve a significant performance boost and memory footprint reduction on the latest generation hardware (GK104) at no cost to accuracy.[36] In Section 2, we present the PME theory as required for a discussion of the technical details of the GPU implementation in AMBER in Section 3. Sections 4 and 5 contain a comprehensive series of simulation results addressing the performance and validation of the code, respectively. Concluding remarks are contained in Section 6.

2. THEORY

In classical MD simulations, the majority of the computational effort is spent evaluating the gradient of the potential energy with respect to the coordinates of a given configuration of atoms that has to be repeated for each time step of the simulation. In the case of the AMBER pairwise additive force fields,[37] the potential takes the form

$$V_{\mathrm{AMBER}} = \sum_{i}^{n_{\mathrm{bonds}}} b_i(r_i - r_{i,\mathrm{eq}})^2 + \sum_{i}^{n_{\mathrm{angles}}} a_i(\theta_i - \theta_{i,\mathrm{eq}})^2 + \sum_{i}^{n_{\mathrm{dihedral}}} \sum_{n}^{n_{i,\max}} (V_{i,n}/2)[1 + \cos(n\phi_i - \gamma_{i,n})] + \sum_{i<j}^{n_{\mathrm{atoms}}}{}' \left(\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}}\right) + \sum_{i<j}^{n_{\mathrm{atoms}}}{}' \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}} \tag{1}$$

where the bond and angle terms are represented by a simple harmonic expression with force constants b_i and a_i and equilibrium bond distances/angles $r_{i,\mathrm{eq}}$ and $\theta_{i,\mathrm{eq}}$, respectively. Torsional potentials for the dihedral angles are represented using a truncated Fourier expansion in which the individual terms have a potential $V_{i,n}$ with periodicity n and phase shift $\gamma_{i,n}$. The last two terms are the vdW interaction represented by a Lennard−Jones potential with diatomic parameters A_{ij} and B_{ij} and the electrostatic interaction between atom-centered point charges q_i and q_j separated by the distance r_{ij}. The prime on the summation of the nonbonded interactions indicates that vdW and electrostatic interactions are only calculated for atoms in different molecules or for atoms in the same molecule separated by at least three bonds. Those nonbonded interactions separated by exactly three bonds are reduced by the application of a scale factor that is dependent on the specific version of the force field (2.0 and 1.2, for vdW and electrostatic, respectively, for the ff99SB[38] version of the AMBER force field).

For explicit solvent calculations, the use of periodic boundary conditions is commonly used to avoid finite size artifacts.

$$V_{\mathrm{vdW}} = \sum_{\mathbf{n}} \sum_{i<j}^{n_{\mathrm{atoms}}}{}' \left(\frac{A_{ij}}{r_{ij,\mathbf{n}}^{12}} - \frac{B_{ij}}{r_{ij,\mathbf{n}}^{6}}\right) \tag{2}$$

$$V_{\mathrm{Coulomb}} = \sum_{\mathbf{n}} \sum_{i<j}^{n_{\mathrm{atoms}}}{}' \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij,\mathbf{n}}} \tag{3}$$

where $\mathbf{n}$, corresponds to the index of the periodic copy of the system. For instance, for a cubic box, the image cells will be located at $\mathbf{n}L$, where $\mathbf{n} = (n_1\hat{x}, n_2\hat{y}, n_3\hat{z})$ is the cell coordinate vector, and L is the box size length. From eq 3, it is clear that for large and periodic systems, the number of nonbonded terms grows rapidly and quickly becomes a bottleneck for the calculation. van der Waals terms decay rapidly with particle separation, thus the use of a simple cutoff plus a correcting term[30] is in most cases sufficient. For the Coulomb interactions, however, this approach leads to large errors in the integration of the equations of motion and is therefore not used. The particle mesh Ewald (PME) method[33] offers a solution to this problem by recasting the slowly converging Coulomb term in terms of three fast converging terms: direct sum, reciprocal sum and self-interaction correction.

$$V_{\mathrm{Coulomb}} = \sum_{\mathbf{n}} \sum_{i<j}^{n_{\mathrm{atoms}}}{}' \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij,\mathbf{n}}} = V_{\mathrm{direct}} + V_{\mathrm{reciprocal}} + V_{\mathrm{self}} \tag{4}$$

For the direct sum, the summation of interparticle point charge interactions is substituted by a sum of point charges screened by Gaussian charge distributions of the same magnitude but opposite sign centered at the positions of each particle. This charge distribution modifies the strength of the interaction, guaranteeing it decays rapidly. This allows the use of a relatively small direct space cutoff radius (typically 8−12 Å) leading to a net reduction in computational effort.

$$V_{\mathrm{direct}} = \frac{1}{2}\sum_{\mathbf{n}}\sum_{i,j}^{n_{\mathrm{atoms}}}{}' q_i q_j \frac{\mathrm{erfc}(\alpha r_{ij,\mathbf{n}})}{r_{ij,\mathbf{n}}} \tag{5}$$

where q_i and q_j are the charges of particles i and j, and $r_{ij,\mathbf{n}}$ is the distance between them. $erfc(x)$ is the complementary error function, $\mathrm{erfc}(x) = 1 - \mathrm{erf}(x)$, and α is the Ewald parameter, which determines how strongly the charge is screened. To compensate for the extra charge density introduced, a sum of Gaussian charge distributions is placed in the position of the point charges, with the same magnitude and sign, and solved for with a grid representation of this periodic charge distribution, termed the reciprocal sum. This periodic distribution charge field is easily decomposed mathematically into a Fourier series in the reciprocal space, which converges faster than the summation over the explicit charges that generated it.

$$V_{\mathrm{reciprocal}} = \frac{1}{2\pi v}\sum_{\mathrm{m}\neq 0}\frac{\exp(-(\pi\mathbf{m}/\alpha)^2)}{\mathbf{m}^2}S(\mathbf{m})S(-\mathbf{m}) \tag{6}$$

where v is the volume of the unit cell, and $\mathbf{m}$ is a reciprocal-lattice vector. $S(m)$ is the structure factor, defined as

$$S(\mathbf{m}) = \sum_{i=1}^{n_{\mathrm{atoms}}}{}' q_i \exp(2\pi i\mathbf{m} \times r_i) \tag{7}$$

$S(\mathbf{m})$ can also be approximated by

$$S(\mathbf{m}) \simeq \sum_{k_1,k_2,k_3} Q(k_1, k_2, k_3)\exp\left(2\pi i\left(\frac{m_1k_1}{K_1} + \frac{m_2k_2}{K_2} + \frac{m_3k_3}{K_3}\right)\right) = F(Q)(m_1, m_2, m_3) \tag{8}$$

where Q is the charge matrix built from the interpolation of the point charges to a 3D grid of the same dimensions of the simulation cell, k_1,k_2,k_3. This expression can be written as a convolution in the reciprocal space. $F(Q)$ is the fast Fourier transform (FFT) of Q, used to solve this equation in $NlogN$ time. Because this method scales as $NlogN$, it is a clear improvement over the scaling of the original Coulomb problem (N^2). The final term in the Ewald sum recast of the Coulomb potential, (V_{self}), accounts for a correction term which removes the self-interactions arising from the charges introduced by the reciprocal term.

$$V_{\mathrm{self}} = \frac{-\alpha}{\sqrt{\pi}}\sum_{i=1}^{n_{\mathrm{atoms}}}{}' q_i^2 \tag{9}$$

The direct space and correction term energy can be directly obtained as written above; the contribution from the reciprocal space is obtained from

$$V_{\mathrm{reciprocal}} = \frac{1}{2\pi v}\sum_{\mathrm{m}\neq 0}\frac{\exp(-(\pi\mathbf{m}/\alpha)^2)}{\mathbf{m}^2}F(Q)(\mathbf{m})F(Q)(-\mathbf{m}) \tag{10}$$

The contributions from the reciprocal space term to each particle is interpolated back using the same functional form used to generate the charge grid. The force is obtained by analytical differentiation of the three terms in the Coulomb energy.

3. TECHNICAL DETAILS OF THE IMPLEMENTATION

There are only a few papers that deal with GPU acceleration of particle mesh Ewald-based methods for biosimulations. Harvey and De Fabritis[39] and Brown, et.al.[40] have published GPU implementations of SPME as part of aceMD[27] and P3M as part of LAMMPS,[41] respectively. OpenMM[42,43] also supports PME and by extension GROMACS[23,24] and CHARMM,[26] which interfaces to OpenMM for GPU accelerated calculations. NAMD supports GPU-accelerated explicit solvent MD simulations; however, the PME part of the calculation is performed on the CPU, while only the direct space sum is handled by the GPUs.[20] Here, we present a complete explicit solvent MD implementation based on the particle mesh Ewald method (PME) in AMBER, first released in 2010 within AMBER v11. We present the details involved in the calculation of all the energy terms in the sums presented above, with special emphasis on the terms concerning the electrostatic interactions estimated with PME and those that have changed since part 1 of this manuscript was published.[29]

We would like to make it clear that the primary goal of this work was to port the exact equations as they are described in AMBER's CPU code. For this reason, we have deliberately not made any additions or included any further approximations, beyond the use of different precision models (discussed in the next section), in porting the code to the GPU.

3.1. Precision Model. Since the publication of our previous work that covered the details of the implementation of the Generalized Born method for implicit solvent on GPUs,[29] our approach has undergone some important changes relative to the use of varying numerical precision. As a result of our previous work, important concerns were raised about the accuracy of the pure single precision (SPSP) model in which all the operations and accumulations of energy terms and forces were performed with single precision variables. This led us to initially develop a hybrid precision model termed SPDP. However, the massive reduction in double precision performance in the NVIDIA Kepler I (GK104) series of GPUs meant we had to rethink the need for double precision floating point variables. As a result, AMBER v12 now includes a new precision model, SPFP, which combines single precision with 64-bit fixed point arithmetic in place of double precision arithmetic to exploit the new generation of GPU hardware without affecting performance of previous generation hardware.[36] Presently AMBER includes three different precision models: (1) the SPFP (default) model addressed above, (2) the historical SPDP that uses single precision floating point variables for everything except the bonded terms and the accumulation of forces that are calculated in double precision arithmetic, and (3) the reference DPDP model in which double precision arithmetic is used throughout the code.

The use of 64-bit fixed point precision has been covered in detail in a previous paper,[36] so we include here only a short description of the main concept behind it. Briefly with the SPFP model, the relative precision of a number x expressed as a floating point value $\mathrm{fl}_n(x)$ with n significant bits is given as

$$\left|\frac{\mathrm{fl}_n(x) - x}{x}\right| < 2^{-n} \qquad (11)$$

The relative precision for an IEEE754 double precision number with 53 significant bits including a hidden bit is of $\approx 10^{-16}$. Typical MD simulations do not require this level of precision. For this, we assert that fixed point 64-bit integer representations Q*m*.*f* with an appropriate choice of magnitude bits m and fractional bits f can be used instead.

AMBER's SPFP precision model replaces all the double precision force accumulators with Q24.40 fixed point integer variables. This definition of accumulators provides 7 significant figures to the left of the radix point and 12 to the right. Given that the forces typically never exceed numerical values of 100.0 in internal units, this should be enough range for typical stable MD simulations. Energy and virial accumulation on the other hand are carried out using Q34.30 fixed point integers. The reason for this is that while forces are vectors of length 3 × N_{atom}, where N_{atom} is the number of atoms, the energy and virial are scalars with only a single accumulator. This means that, as the system grows, the values for these variables could easily overflow a Q24.40 accumulator. Energies usually do not affect the trajectory and are typically only written to 4 decimal places, and thus, we use Q34.30 FP arithmetic, which provides approximately 10 significant figures to the left of the radix point and 9 to the right.

The SPFP precision model also makes extensive use of integer atomic operations to perform accumulations. Integer atomic operations have improved in performance significantly since the days of the tesla (C2050) GPUs. The Fermi chips improved this a small amount, and then Kepler improved it dramatically. On Fermi, the atomic operation code is actually slower than the original SPDP implementation, but the difference is compensated for with performance improvements in other parts of the code. With Kepler, the atomic operations are sufficiently fast that the simple implementation of summing into fixed precision is just as quick as using the SPDP approach.

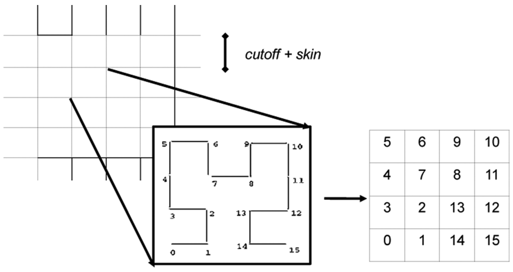
Because SPFP is now the default model and SPDP/DPDP are reserved for regression testing, the description of each energy term will be focused on the SPFP approach. An extensive description of most energy terms for the SPDP and DPDP models is presented in our previous paper.[29] A detailed description of the algorithms involved in PME and neighbor list, valid for all precision models, is presented here.

3.2. Nonbonded Interactions.

The present implementation of AMBER has evolved from the one presented in our previous work.[29] In the initial implementation, a careful use of output buffers allowed the accumulation of the contributions from nonbonded interactions without the risk of race conditions and the benefit of maintaining determinism in the accumulators. However, the release of NVIDIA's GK104 (Kepler I) series of GPUs (e.g., K10/GTX6XX) brought poor DP performance, making it beneficial to evaluate alternatives including integer atomics. The efficient use of atomic functions has significantly reduced the memory requirements of the code (Table 2) compared with that needed by the original SPDP precision model because it negates the need for the accumulation buffers and at the same time improves performance on Kepler hardware, which has approximately 3× faster integer atomics.

3.3. Particle Mesh Ewald. *Direct Sum and Neighbor List.*

The contributions for the direct space sum are accounted for in the same way as nonbonded interactions in the implicit solvent implementation described in ref 29 with the exception that interactions are only calculated for pairs of atoms separated by less than a cutoff and the electrostatic interaction is modulated by erfc as described in eq 5. The interactions included in the N' of the summation in eq 5 are obtained from a neighbor list that is updated heuristically based on a buffer region as used in most CPU MD codes. Calculation of the neighbors in the GPU version of PME is conducted in parallel on the GPU. The neighbor list is constructed in a single phase that combines two sorting mechanisms. The atoms are primarily sorted spatially into boxes of at least the nonbond cutoff plus the skin buffer (*extended cutoff*) in each dimension.The box ID to which each particle belongs is stored in the highest order bits of the sorting key. The number of bits used here is determined dynamically based on the number of nonbond boxes in the calculation. The next 6 bits encode a 4 × 4 × 4 Hilbert curve[44] to improve the spatial locality of adjacent atoms (Figure 1), giving a series of sorting keys for each atom.



5	6	9	10
4	7	8	11
3	2	13	12
0	1	14	15

Figure 1. Determination of the neighbor list based on a Hilbert curve, second order H_2 curve shown here, for a periodic system.

For each box, we generate a Hilbert curve that assigns atom IDs for atoms in the box that gives good spatial locality. Each ID corresponds to the Hilbert curve coordinates calculated from a lookup table (Table 1). This ID is stored in the low 6 bits, as mentioned before, 2 bits for each of the 3 intercellular coordinates. These sorting keys are then used to sort the atom list. We then go through in a single pass in groups of either 16 or 32 (based on GPU specifications and cutoff size) and find all other atoms within the *extended cutoff* of them restricting this to atoms in adjacent boxes on the leading edge, giving interactions with 13 adjacent boxes and itself. Atoms closer than $(extendedcutoff)^2$ are added to the neighbor list. The direct space sum then just involves iterations over the neighbor list in a parallel fashion in sizes of a warp.

An optimization of the neighbor list build that may be included in future versions of the code as described earlier is to use a bit-mask based on the Hilbert ID to exclude certain atoms from the R^2 calculations.

Both electrostatic and vdW contributions are calculated explicitly inside the kernel; no lookup tables are used. For vdW interactions, the individual atom type based parameters are extracted from the combined table in the topology file and then stored as atom type parameters and combined in the kernel as needed. A mask is applied to the calculation of the direct space sum to skip the calculations of 1−2, 1−3, 1−4 interactions and

Table 1. Cell Hash Lookup Table (4 × 4 × 4)

```
static const unsigned int cellHash[CELLHASHCELLS] = {
34,  35,  60,  61,
33,  32,  63,  62,
30,  31,   0,   1,
29,  28,   3,   2,

37,  36,  59,  58,
38,  39,  56,  57,
25,  24,   7,   6,
26,  27,   4,   5,

42,  43,  54,  53,
41,  40,  55,  52,
20,  23,   8,   9,
21,  22,  11,  10,

45,  44,  49,  50,
46,  47,  48,  51,
19,  16,  15,  14,
18,  17,  12,  13,
          };
```

any other interaction that may be listed as excluded in the topology file.

Reciprocal Sum. The implementation of the reciprocal sum entails five principal steps: (1) charge spreading onto a charge grid Q, (2) 3D fast Fourier transform (FFT) of Q from real to complex, (3) energy computation in reciprocal space, (4) 3D fast Fourier transform (FFT) of the convolution from complex to real, and (5) per atom force term computation in real space.

Of all these steps, the most complex part is building the charge grid. This includes projecting each charge onto the corners of the grid box in which it resides. Clearly, this is a prime candidate for race conditions in parallel. Formerly, this was avoided by using a complex series of up to 27 accumulation buffers for each grid point laid out in a repeating 2 × 2 × 2 to 3 × 3 × 3 cuboid lattice matched to the nonbond cells. In the latest version of AMBER, the charge grid is now built using atomic accumulations which avoids this complexity.

All fast Fourier transforms are performed using the CUFFT library[45] included in CUDA on GPU 0 in the case of multi-GPU runs. CUFFT offers the possibility to use single or double precision. The precision used for each model matches the SP or DP of the precision mode prefix, namely, SP for SPFP and SPDP and DP for the DPDP model. 3D transforms are used in all cases.

Energy contributions per atom are calculated as in equation eq 10, one thread per particle. The calculation of the forces are performed by analytical differentiation as described in Section 2, one grid point per thread. Force contributions to each atom are projected out from the grid points in a similar, but inverse, fashion as the charges.

3.4. Bonded and 1–4 Interactions. Bonded and 1–4 interactions are less numerous than the nonbonding interactions but of crucial importance to the evolution of the calculation. In order to exploit the massive parallelism of the GPUs, these interactions are placed in a list ordered by type (bond, angle, dihedral, etc.) and directly divided up across SMs, assigning one task per thread. To avoid race conditions from terms involving one or more coincident atoms, atomic operations are used in the force accumulation. Atomic operations are also used in the accumulation of the energy and virial terms when necessary. Scaled 1–4 electrostatic and vdW interactions are calculated entirely in the same kernel. The reason for explicitly excluding 1–2, 1–3, and 1–4 interactions in the nonbond calculation and then calculating scaled 1–4s along with the dihedral terms rather than calculating them twice and subtracting the difference, which could potentially give slightly better performance, is to avoid the potential loss of precision in the forces that can occur with this approach.

3.5. Harmonic Restraints. Harmonic restraints are expressed in a similar fashion as a bonded interaction between an atom and a fixed virtual particle representing the reference structure. Calculations of the restraint force is handled as part of the bond calculation.

3.6. SHAKE Algorithm. In the present implementation in line with the most common mode of operation used in AMBER simulations, the SHAKE algorithm[46] is only applied to hydrogen atoms. In the implementation, each heavy atom is sorted by the number of attached hydrogen atoms and treated on a different thread. This gives enough parallelism to the algorithm and ensures efficient execution across card thread counts. The SHAKE calculations, to avoid issues associated with taking the small difference of two large numbers, is carried out entirely in double precision. The performance impact of this is minimal given the shake calculation is a very small part of the overall calculation.

3.7. Coordinate Update. The time integration of the trajectory is performed entirely on the GPU to avoid the costly transfer of information between CPU and GPU memory. Each atom is treated independently on a separate thread due to the intrinsic parallel nature of this step of an MD simulation.

3.8. Thermostats. For MD simulations using the Anderson thermostat or the Langevin thermostat it is necessary for the random number generator (RNG) to also be perfectly deterministic in order for any two initially identical simulations to be reproducible. To this end, we use a parallelized version of the Mersenne Twister RNG as implemented in the CURAND library that is available with the CUDA Toolkits since version 3.2.

Currently, the Berendsen, Anderson, and Langevin thermostats are supported. The thermostat is applied after the coordinate update and similarly takes place entirely on the GPU using one thread per particle to avoid the costly transfer of information to the CPU.

3.9. Barostats. PMEMD currently supports the Berendsen barostat, although the Anderson and Nose–Hoover Lagevin barostats have been implemented in a development version of the software for the use with lipid bilayer simulations and scheduled for release with the next major version update. With the Berendsen barostat, the *xx*, *yy*, and *zz* components of the virial tensor are accumulated in fixed precision (to maintain determinancy) using atomic operations at the time of the force calculations. Pressure scaling then takes place entirely on the GPU on a molecule by molecule basis within a series of pressure scaling kernels called as part of the coordinate update. Solvent molecules, which typically massively outnumber solute molecules, are compressed to a single kernel and calculated using one thread per molecule. The solute typically contains hundreds to thousands of atoms per molecule. Calculations proceed by first padding the atoms of each molecule to warp fragments (typically 32). A warp then passes through the atoms of each molecule accumulating molecular virials and center of mass data depending on the kernel. When a new molecule is reached, the kernel dumps all accumulated data to the appropriate molecule's accumulators in an atomic fashion.

3.10. Parallelization Across GPUs. As with the other MD engines included in AMBER, the message passing interface protocol (MPI)[47] is used to run parallel simulations across multiple GPUs in the same or interconnected nodes. At the time of writing, all data structures are replicated on all available GPUs; thus, the memory usage per GPU is roughly identical in serial or parallel simulations. We expect this to improve in later versions as we seek to exploit better communication between GPUs and extensive direct memory copies using peer to peer functionality.

The atoms are evenly distributed among all GPUs, and the nonbond calculations internally load balance these static allocations between SMs within each GPU. Standard MPI reductions are used to globally sum the forces, and then all coordinates are updated on each GPU. The deterministic nature of our SPFP precision model means that it is not necessary to communicate updated coordinates across GPUs, which otherwise would be needed to deal with rounding differences during the integration. While a serial GPU bottleneck, the time step integration is <1% of the total iteration, and so the typical GPU count of 2 to 16 is not critical to performance.

For the PME calculation, the FFT calculation and charge grid interpolation are carried out on GPU 0 of node 0. Nonbond cells are laid out across the GPUs proportionally. If the number of nodes is ≤4, then node 0 is given a proportionally smaller share of direct space. This works surprisingly well because the nonbond calculation, as mentioned above, is already load balanced automatically on each GPU. In the case of >4 GPUs, then GPU 0 does only the reciprocal sum.

3.11. Features of Implementation. *Supported Methods.* The GPU implementation of AMBER supports all of the main features included in pmemd. At the time of writing, the following major options are supported, among others (1) pairwise additive AMBER and CHARMM force fields, (2) extra points, (3) NMR restraints, (4) isotropic periodic sum (IPS), particle mesh Ewald (PME), and Generalized Born (GB) electrostatics models, (5) harmonic restraints, (6) shake on hydrogen atoms, (7) temperature scaling, (8) pressure scaling, and (9) execution in parallel across multiple GPUs, as well as a number of advanced features including temperature replica exchange molecular dynamics (REMD), accelerated molecular dynamics (aMD),[9] umbrella sampling, and simulated annealing.

The current development version of the software also includes support for multidimensional Hamiltonian replica exchange MD running over 1000 GPUs in parallel on the NSF Blue Waters supercomputer. In the near future, we expect to also add support for more advanced features such as constant pH, thermodynamic integration, and self-guided Langevin dynamics.

System Size. GPU memory and the MD simulation parameters determine the maximum system size that can be treated with the GPU implementation. The physical GPU hardware itself, to some extent, affects memory usage because the optimizations used are nonidentical for different GPU types. With respect to MD simulation parameters, the Langevin thermostat and the use of larger cutoffs for vdW and electrostatics, as well as the use of virial dependent barostats for NPT simulations, increase the memory requirements.

Table 2 gives an overview of the approximate maximum atom counts that can be treated with the present version 12 of the code on different GPU hardware.

Table 2. Approximate Maximum Atom Counts That Can Be Treated with GPU Implementation of PME Explicit Solvent Simulations in the Released Version of AMBER 12 Using the DPDP, SPDP, and SPFP Precision Models[a]

GPU		PME (max atoms)		
type	memory	DPDP	SPDP	SPFP
GTX580	3.0 GB	870,000	1,060,000	1,240,000
Tesla M2090	6.0 GB	1,820,000	2,230,000	2,680,000
GTX680	2.0 GB	460,000	710,000	920,000
K10/GTX680	4.0 GB	1,270,000	1,520,000	1,810,000
K20X/GTX-TITAN	6.0 GB	1,890,000	2,270,000	2,710,000

[a] Test systems are cubic boxes of TIP3P water molecules (for details of the simulations, see the Supporting Information). Error-correction code (ECC) was switched off on the Tesla cards (M2090, K10, K20X).

At the time of writing, the memory usage per GPU in parallel runs does not decrease with an increasing number of GPUs due to the replicated data model currently in use.

4. PERFORMANCE

Serial GPU Performance. The present AMBER GPU implementation supports three different precision models, namely, SPDP, SPFP, and DPDP. The DPDP is a reference model equivalent to the approach used on the CPU that uses double precision throughout the code. Because of the design of GPU hardware, the use of double precision can be detrimental to the performance of the GPU implementation. Therefore, the desire to achieve high performance prompts for the use of SP where possible. As our previous work shows,[29] it is critical to use SP carefully in order to not negatively affect the accuracy of the MD. We have shown that the use of SP exclusively throughout the code can give rise to unpredictable artifacts in the simulation and can therefore cause problems in structural properties. For this reason, we designed our hybrid precision SPFP model to achieve performance very close to that achievable with pure SP models but in a manner that does not impact accuracy. The precision model by default in AMBER v12 is therefore the SPFP model. The results of the simulation timings for single GPU runs using the three different precision models available in AMBER v12 are summarized in Table 3 along with the pure single precision (SPSP) performance from AMBER v11.[48] [New NVIDIA hardware (Kepler series) is only supported by AMBER v12. AMBER v12 does not support SPSP and thus cannot be used in the corresponding performance calculations.]. Even for small simulations of 23,558 atoms, the serial GPU version of the code is substantially faster (110.65 ns/day) on a single GPU (GTX-TITAN) than using a state of the art CPU node. Using all cores of one Dual × Oct Core Intel Sandy Bridge E5-2670 2.6 GHz node the CPU achieves a peak performance of 21.13 ns/day. As shown in Table 3, the performance of SPFP is far greater (for the GTX680 and K10) than that for SPDP and DPDP while the accuracy remains practically the same in all three as discussed later. SPFP is never slower than SPDP and is significantly faster for cards where double precision performance is much lower than the single precision performance. Table 4 shows the performance of the SPFP implementation

Table 3. Throughput Timings (ns/day) for AMBER PME Simulations of DHFR (23,558 atoms) with a Time Step of 2 fs Using the Three Different Precision Models of the Serial GPU Version[a]

	PME (DHFR)			
GPU type	SPSP	SPFP	SPDP	DPDP
GTX580	58.00	54.80	53.00	10.00
M2090	49.40	46.60	46.20	16.20
K10 (1 GPU)	NA	52.03	33.06	4.68
GTX680	NA	74.40	45.90	6.60
K20X	NA	89.41	66.47	23.13
GTX-TITAN[b]	NA	110.65	69.85	10.78

[a] For details on the simulations and the hardware and software stack, see the Supporting Information. [b] Running in "Gaming" mode.

for a single GPU card as a function of calculation size for a range of commonly used GPUs.

Table 4. Throughput Timings (ns/day) for AMBER PME Simulations of DHFR (23,558 atoms), Factor IX (90,906 atoms) and Cellulose (408,576 atoms) with a Time Step of 2 fs Using the Serial GPU Version with the SPFP Precision Model[a]

	DHFR	Factor IX	cellulose
GPU type	(23,558 atoms)	(90,906 atoms)	(408,576 atoms)
C2075	36.89	10.62	2.18
M2090	46.60	13.28	2.67
GTX580	54.80	15.47	3.16
GTX680	74.40	22.65	4.36
K10 (1 GPU)	52.03	16.03	3.28
K20X	89.41	25.45	6.16
GTX-TITAN[b]	110.65	31.55	7.69

[a] For details of the simulations, see the Supporting Information. [b] Running in "Gaming" mode.

Table 5. Multi-GPU Throughput Timings (ns/day) for AMBER PME Simulations with a Time Step of 2 fs Using the Parallel CPU Version (16 Intel Xeon E5-2670 cores connected via Dual rail QDR InfiniBand network) and the Parallel GPU Version with the SPFP Precision Model[a]

	DHFR	Factor IX	cellulose
CPU/GPU	(23,558 atoms)	(90,906 atoms)	(408,576 atoms)
GPU version			
2 × GTX-TITAN	125.28	38.05	8.72
1 × GTX-TITAN	110.65	31.55	7.69
2 × K20X	106.54	33.13	7.67
1 × K20X	89.41	25.45	6.16
4 × GTX680	118.88	33.84	7.56
3 × GTX680	101.31	27.90	6.47
2 × GTX680	86.84	23.05	5.30
1 × GTX680	74.40	22.65	4.36
8 × K10	113.82	28.86	7.06
6 × K10	112.73	29.54	7.17
4 × K10	97.95	25.72	6.44
3 × K10	82.70	23.25	5.45
2 × K10	67.02	19.12	4.33
1 × K10	52.03	16.03	3.28
CPU version			
384 × E5-2670	–	–	5.92
256 × E5-2670	–	–	5.88
128 × E5-2670	–	22.68	5.72
96 × E5-2670	–	19.33	4.73
64 × E5-2670	58.19	17.52	3.73
48 × E5-2670	50.25	13.99	2.95
32 × E5-2670	38.49	10.28	2.05
16 × E5-2670	21.13	5.57	1.12

[a] Note: 1 × K10 means 1 GPU of the two available on the card, therefore 8 × K10 means four K10 cards were used. A dash represents slowdown over lower concurrency.

Parallel GPU Performance. Table 5 shows AMBER performance for parallel GPU runs with the SPFP precision model and the traditional CPU implementation. A throughput of up to 118.88 ns/day for DHFR, up to 33.84 ns/day for Factor IX, and up to 7.56 ns/day for cellulose can be achieved with four (≈ \$500 each) NVIDIA GTX680 GPUs in a single node, while 125.28, 38.05, and 8.72, correspondingly, are obtainable with just 2 GTX-TITAN (\$1000 each) in a single node. This is more than a factor of 2 faster than the maximum throughput that can be achieved on a typical supercomputer for DHFR given that the CPU scaling plateaus at 58.19 ns/day long before it reaches the performance achieved by the GPU implementation. In the case of Factor IX, the CPU performance plateaus at 22.69 ns/day, less than two-thirds of the observable single node GPU (2 GTX-TITAN) performance. For cellulose, the maximum performance achieved by the CPU code in our test is 5.92 ns/day, just two-thirds of the single node GPU performance. The present implementation on a single GPU (GTX-TITAN) is still faster than the CPU scaling limit for all tests. Up to this point, we have focused on optimizing single GPU performance in order to support the broadest audience and maximize scientific impact. It should be noted that unlike a number of competing codes our implementation runs entirely on the GPU. As such, four independent calculations can be run on a single workstation containing four GPUs without any reduction of performance. This was in part our motivation for not attempting to obtain the very last bit of performance out of a node by also attempting to use the CPU cores. Future GPU implementations of AMBER will also focus on optimizing algorithms to enhance the multi-GPU performance. As mentioned in Section 3.10, we expect the parallel scaling of the GPU code to improve dramatically in future versions with the use of new technology such as direct GPU to GPU memory copies.

5. VALIDATION

With every new addition to a software package, regression tests must be performed to ensure the latest additions do not affect the outcome of all the previous supported capabilities of the code. AMBER has a very extensive set of regression tests that evaluate every major feature with different combinations of run parameters to ensure the back compatibility of any release of the code with previous versions. These tests aim to prevent the introduction of any new bugs or critical logic errors. The implementation of the full double precision version of the GPU code matches the CPU code to machine precision and is thus used to check for the correctness of the GPU code when using the same regression tests.

For the GPU implementation, there is the need to validate the hybrid precision models SPFP and SPDP as well. This is significantly more complex because it requires a careful evaluation of any subtle differences in the dynamics as well as ensemble properties from converged simulations run in all precision models. In this section, we attempt to extensively validate our GPU code comparing a number of key observables across all three DPDP, SPFP, and SPDP precision models with the reference CPU calculation. We have also included some results obtained from our now deprecated precision model SPSP for completeness.

5.1. Single Point Forces.

The most important values calculated during an MD simulation are the forces acting on each atom that determine in turn the evolution of the simulation. Therefore, the effect of the numerical precision model on the force calculations as compared to the CPU reference is an important metric in the validation of our GPU implementation. The deviations in the forces are summarized in Table 6 for the test systems used in Section 4 plus the satellite tobacco mosaic virus (STMV) (1,066,846 atoms).

Table 6. Deviations of Forces (kcal/(mol Å)) of the AMBER pmemd GPU Implementation Using Different Precision Models for PME Explicit Solvent Simulations as Compared to Reference Values Obtained with the CPU Implementation

	DHFR	Factor IX	Cellulose	STMV
precision model	(23,558 atoms)	(90,906 atoms)	(408,576 atoms)	(1,066,846 atoms)
max deviation				
SPSP	5.3×10^{-3}	1.4×10^{-2}	1.9×10^{-2}	9.4×10^{-2}
SPFP	3.4×10^{-4}	2.2×10^{-3}	1.9×10^{-3}	4.0×10^{-3}
SPDP	3.4×10^{-4}	2.2×10^{-3}	$1.9 \times \times 10^{-3}$	4.0×10^{-3}
DPDP	6.0×10^{-8}	2.1×10^{-6}	4.6×10^{-8}	9.4×10^{-8}
RMS deviation				
SPSP	2.8×10^{-4}	3.6×10^{-4}	9.4×10^{-4}	1.2×10^{-3}
SPFP	2.3×10^{-5}	1.2×10^{-4}	7.8×10^{-5}	1.3×10^{-4}
SPDP	2.3×10^{-5}	1.2×10^{-4}	7.8×10^{-5}	1.3×10^{-4}
DPDP	1.5×10^{-9}	2.1×10^{-7}	1.3×10^{-9}	2.1×10^{-9}

The DPDP model matches the reference forces very closely with maximum deviations not exceeding 10^{-6} kcal/(mol Å) and RMS deviations not exceeding 10^{-7} kcal/(mol Å), even for systems as large as the STMV (1,066,846 atoms). These deviations are entirely due to the different order of execution of the floating point operations in the CPU and GPU implementations. The DPDP GPU implementation of pmemd will thus generate trajectories of precision equivalent to the CPU implementation.

Calculating the force contributions in SP and accumulating them in DP or FP show identical and small deviations in comparison to the reference CPU code. In the following section, we show that the forces obtained from the SPDP and SPFP models are sufficiently accurate to conserve energy in biomolecular MD simulations and present no problems in long time scale MD simulations. Forces obtained with the SPSP precision model here present a deviation in the forces at least 1 order of magnitude larger in all the systems tested compared to SPDP and SPFP. This can lead to numerical instabilities as shown in our previous work and Section 5.2.

5.2. Energy Conservation.

The ability to conserve the constants of motion is an important measure to judge the precision of an MD software package. In an NVE simulation, the energy is one of such constants. We have performed constant energy MD simulations of the first three test systems of increasing size used in Section 4. We collected data for 10 ns (DHFR), 5 ns (Factor IX), and 1 ns (cellulose) after an initial equilibration for 1 ns at 300 K using Langevin dynamics. Three different calculations were run with time steps of 0.5, 1.0, and 2.0 fs, respectively. Simulations using a time step of 2 fs were performed with bonds to hydrogen atoms constrained using the SHAKE algorithm with a relative geometrical tolerance of 10^{-6} Å, while runs using time steps of 0.5 and 1.0 fs were performed without constraints.

The energy drifts, obtained from the slope of a linear regression fit to the energy values obtained during the calculations described above, are summarized in Table 7, while Figure 2 shows a plot of the total energy for the trajectories of DHFR and Factor IX for the different precision

Table 7. Energy Drifts per Degree of Freedom (kT/ns/dof) from Simulations of 10 ns (DHFR), 5 ns (Factor IX), and 1 ns (cellulose)[a]

time step	0.5 fs	1.0 fs	2.0 fs
DHFR (23,558 atoms)			
CPU	-1.3×10^{-7}	1.3×10^{-6}	-4.7×10^{-5}
GPU (DPDP)	1.0×10^{-6}	1.3×10^{-6}	-4.4×10^{-5}
GPU (SPDP)	-5.2×10^{-5}	5.0×10^{-5}	-1.4×10^{-5}
GPU (SPFP)	-5.2×10^{-5}	5.3×10^{-5}	-1.7×10^{-5}
GPU (SPSP)	2.0×10^{-3}	1.2×10^{-3}	1.5×10^{-1}
Factor IX (90,906 atoms)			
CPU	-4.8×10^{-8}	2.1×10^{-6}	-2.2×10^{-5}
GPU (DPDP)	7.8×10^{-7}	3.0×10^{-6}	-2.8×10^{-5}
GPU (SPDP)	-9.9×10^{-5}	6.4×10^{-5}	-4.1×10^{-5}
GPU (SPFP)	-9.9×10^{-5}	6.4×10^{-5}	-4.1×10^{-5}
GPU (SPSP)	2.7×10^{-3}	1.6×10^{-3}	1.3×10^{-1}
cellulose (408,576 atoms)			
CPU	6.2×10^{-6}	3.7×10^{-5}	2.3×10^{-5}
GPU (DPDP)	1.1×10^{-5}	5.1×10^{-5}	2.9×10^{-5}
GPU (SPDP)	6.1×10^{-5}	-3.1×10^{-5}	1.1×10^{-4}
GPU (SPFP)	5.9×10^{-5}	-3.5×10^{-5}	1.2×10^{-4}
GPU (SPSP)	6.5×10^{-3}	3.5×10^{-3}	5.0×10^{-2}

[a] The SHAKE algorithm to constrain bond lengths to hydrogen atoms was used for a time step of 2.0 fs; no constraints were used for smaller time steps.

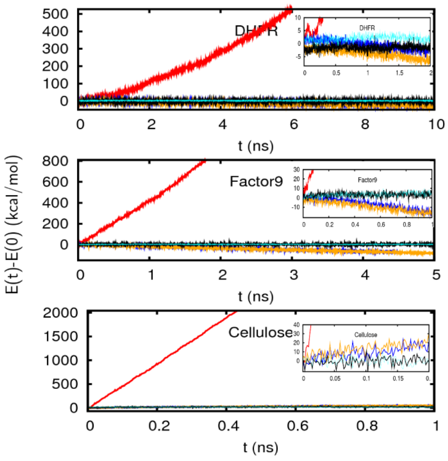


Figure 2. Total energy (kcal/mol) along constant energy trajectories using a time step of 0.5 fs without constraints. Shown are results for DHFR (top), Factor IX (center), and cellulose (bottom) for different precision models. Red SPSP, dark blue SPFP, orange SPDP, black DPDP, and light blue CPU.

models at a time step of 0.5 fs. It can be seen that all precision models presently included in AMBER v12 (SPFP, SPDP, and DPDP) behave reasonably well for all trajectories. Although SPSP has been officially deprecated in AMBER v12, we have included some results for this precision model to offer some critical information on the use of this model. We can see in Figure 2 that while most precision models have a similar very small energy drift, SPSP diverges significantly from the rest. Apart from the magnitude in the energy drift, the corresponding plot for all systems with 1 or 2 fs time steps present a similar behavior and can be found in the Supporting Information. Table 7 shows the magnitude of the energy drift per degree of freedom. The divergence of the energy is significantly larger for the SPSP precision model, being in some instances more than 4 orders of magnitude larger.

By way of comparison, other MD codes have reported values for energy drifts for simulations of DHFR under similar conditions. The literature-reported energy drift values (in units of KT/ns/dof) for some important MD software packages are (for DHFR) 0.011 for GROMACS,[24] 0.0047 for NAMD,[49] and 0.0033 for Desmond[49] for a simulation with a time step of 1 fs. The last two values have been scaled by a factor of 1/600 with respect to the reported values for correct unit conversion. In light of this, AMBER's SPFP value of 0.000053 for DHFR compares very favorably.

Although simulations using full single precision floating point implementations seem stable and are widely used by other groups, the net effect of rounding errors on the distributions sampled is hard to assess. It can be observed in all the previous results that SPFP reproduces the behavior of the higher precision models without presenting any of the potential problems SPSP does. This positive behavior, plus the favorable performance and scaling, favored by the architecture of modern GPU cards, are the reason why SPFP is now the default precision model for AMBER and SPSP has been formally deprecated.

5.3. Structural Properties.

Molecular dynamics has been widely used in the study of protein interaction and function. Most of these studies are based on structural analysis performed during an MD simulation. All these studies depend on the reliability of the structural properties obtained during the simulation.

In order to test the reliability of our implementation for protein dynamics, we present results of MD simulations of ubiquitin with the different accuracy models of our GPU implementation. We present results for the root-mean-square deviations (RMSDs) and root-mean-square fluctuations (RMSFs) of the C_α backbone carbon atoms with respect to the crystal structure (PDB code 1UBQ[50,51]). Residues 71 to 76 have been excluded due to the inherent high flexibility these residues present.

In order to attempt statistically converged results, 60 independent MD trajectories each of 100 ns length were generated at 300 K for each of the precision models of the GPU and for the CPU implementation. The runs were performed at constant temperature, using the Berendsen weak coupling algorithm[52] with a time constant of $\tau_T = 10.0$ ps. Although the use of a thermostat will to some extent cover up numerical noise introduced by numerical inaccuracies in the implementation, the runs were designed to reflect the fact that typical biomolecular AMBER simulations are generally run with some form of temperature control. The ff99SB[38] force field was used for all simulations with a time step of 2 fs and bonds to hydrogen atoms were constrained using the SHAKE algorithm with a relative geometrical tolerance of 10^{-6} Å. Output and trajectory files were saved every 1000 steps (2 ps) for analysis.

The starting point for each of the 60 trajectories was obtained from three 200 ns long CPU runs of ubiquitin each prepared as follows. Starting from the crystal structure an energy minimization using 2000 steps of steepest descent followed by heating and an initial equilibration for 1 ns at 300 K using Langevin dynamics with a collision frequency of $\gamma = 1.0$ ps^{-1} was conducted. A 200 ns long MD run at a constant temperature of 300 K was performed from which the 20 snapshots were selected to be equally separated in simulation time. Each of the 60 resulting snapshots was then assigned random velocities corresponding to a temperature of 10 K followed by heating and equilibration to 300 K using Langevin dynamics for 50 ps and finally the removal of any center of mass motion that may have been introduced. Using the CPU generated restart files in all cases guarantees that any numerical differences observed between the various implementations can be traced back to the numerical precision of the implementation and is not an artifact of different initial conditions.

Figure 3 shows RMSD values for all simulations of ubiquitin using the CPU and all GPU versions of the code. The results are equivalent across all versions showing that the relative mobility and spatial distribution of the protein is maintained. The RMSF values for each residue for the 60 native-state simulations are shown in Figure 4. The results for RMSD (Figure 3) and the corresponding RMSF values (Figure 4) remain very similar for all GPU precision models and the CPU calculations.

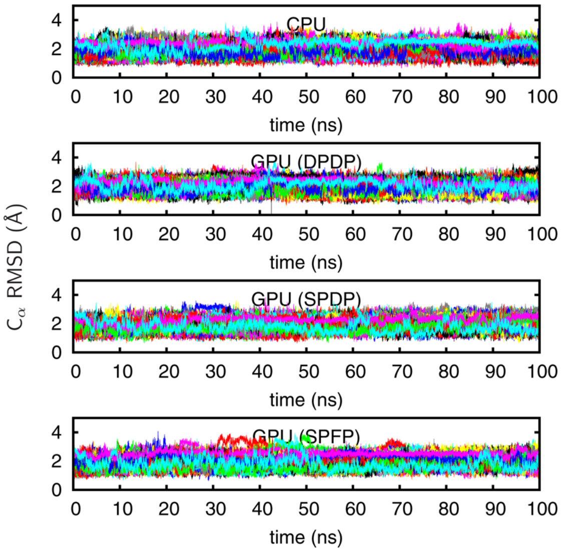


Figure 3. Root-mean-square deviations (RMSDs) of the C_α backbone carbon atoms of ubiquitin (excluding the flexible tail, residues 71 to 76) with respect to the crystal structure for 60 NVT independent trajectories as obtained with the CPU implementation and the GPU implementation of pmemd using different precision models.

6. CONCLUSIONS AND OUTLOOK

We have presented our implementation of a GPU-accelerated MD engine in AMBER (pmemd.cuda), first publicly released in 2010, for explicit solvent simulations using PME for long-range interactions. We have formally presented our GPU implementation, plus some improvements that were made since the first

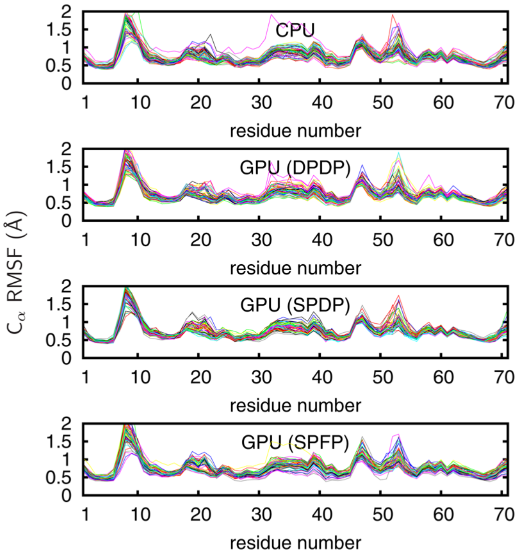


Figure 4. Root-mean-square fluctuations (RMSFs) of the C_α backbone carbon atoms of ubiquitin residues 71 to 76 with respect to the crystal structure for 60 NVT independent trajectories of 100 ns length as obtained with the CPU implementation and the GPU implementation of pmemd using different precision models.

part of this paper describing the implicit solvent GB implementation was published. The implementation is based on AMBER pmemd and includes the majority of the capabilities of the regular CPU MD engine. Input and output files follow the same guidelines for all AMBER MD engines, such that the only change on the user's end is the name of the executable being invoked, thus removing any learning curve. Additionally, all regression tests and analysis tools included in AMBER or developed by other groups offering support for AMBER are completely compatible with the GPU implementation; thus, no extra time should be invested on the user's side to adapt their established pipelines of work. The present implementation offers support for both AMBER and CHARMM force fields as well as several important methods commonly used in MD such as accelerated molecular dynamics (aMD) and replica exchange MD. The implementation runs with single or multiple GPUs in a single node or across multiple nodes via MPI. The peak performance for simulations of typical simulation size is of the order of 100 ns/day, for example, 110.65 ns/day for DHFR (23,558 atoms) running on a single NVIDIA GTX-TITAN GPU card. Similar calculations performed on a state of the art CPU node perform almost 6 times worse, for example, 16 cores of a Dual × Oct Core Intel Sandy Bridge E5-2670 2.6 GHz CPU have a peak performance of 21.1 ns/day. Even on the best CPU clusters, the performance tops off at half of the performance (58.2 ns/day) compared to the GPU code. This simple analysis also reflects the fact that four GPUs can be added to a single node for a total of less than $5000 and can run four independent calculations each at full speed. The GPU accelerated software implementation thus offers the possibility to perform high-performance production runs on commodity hardware with a minimum investment of money and without long queue waiting times. This will surely enhance the natural pipeline of scientific biomolecular and chemical research.

We have shown that a precision model using entirely single precision floating point arithmetic (SPSP) shows force deviations several orders of magnitude larger than mixed precision models. This leads to perceivable drifts in energy for NVE simulations as a result of an accumulation of errors along the trajectory due to rounding. While this behavior can to some extent be compensated in NVT simulations with the use of good thermostats, it is an uncontrolled source of error and its possible effects are hard to foresee. These observations are consistent with the behavior presented in equivalent GB simulations.[29] SPSP has therefore been deprecated from the present implementation of AMBER. SPFP offers an excellent alternative solution as it not only offers numerical stability but in addition performs and scales almost equivalently with a lower memory requirement on modern GPU cards.

GPU technology continues to evolve and improve rapidly due to its vast number of applications in research and commercial areas. The development of better cards and improved intra- and inter-node communication will surely continue to improve the performance and applicability of GPUs for scientific research and high-performance computing. Since the release of our GPU accelerated implementation of AMBER in 2010, the number of users and papers using this engine has been growing rapidly and continuously. GPU accelerated AMBER has proved to be a widely used and appreciated software that is positively impacting the molecular chemical and biological research fields.

■ ASSOCIATED CONTENT

Ⓢ Supporting Information

AMBER input files used for the performance tests of Section 4, for the validation of the accuracy of single point forces of Section 5.1, for the validation of energy conservation of Section 5.2, and for the validation of the numerical accuracy of structural properties of Section 5.3. Plots showing the energy conservation for the trajectories of DHFR, Factor IX, and Cellulose for the different precision models and time steps of 1.0 and 2.0 fs. This material is available free of charge via the Internet at http://pubs.acs.org.

■ AUTHOR INFORMATION

Corresponding Author

*E-mail: ross@rosswalker.co.uk.

Present Address

‖Scott Le Grand: Amazon Web Services, 2201 Westlake Ave., Suite 500, Seattle, Washington 98121, United States.

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

This work was funded in part by the National Science Foundation through the Scientific Software Innovations Institutes program NSF SI2-SSE (NSF1047875 and NSF1148276) grants to R.C.W and also by the University of California (UC Lab 09-LR-06-117792) grant to R.C.W. Computer time was provided by the San Diego Supercomputer Center through National Science Foundation award TG-CHE130010 to R.C.W. and A.W.G. The work was also supported by CUDA fellowships to R.C.W. and S.L.G. from NVIDIA. S.L.G. thanks Amazon Web Services for support and also his late father (Donald George Le Grand) for inspiration.

■ REFERENCES

(1) Grossfield, A. *Biochim. Biophys. Acta* **2011**, *1808*, 1868−1878.

(2) André, S.; Pei, Z.; Siebert, H.-C.; Ramström, I.; Gabius, H.-J. *Bioorg. Med. Chem.* **2006**, *14*, 6314−6326.

(3) Harvey, M. J.; De Fabrittis, G. *Drug Discovery Today* **2012**, *17*, 1059−1062.

(4) Zwier, M. C.; Chong, L. T. *Curr. Opin. Pharmacol.* **2010**, *10*, 745−752.

(5) Durrant, J. D.; McCammon, J. A. *BMC Biol.* **2011**, *71*, 1−9.

(6) Pérez, A.; Luque, F. J.; Orozco, M. *Acc. Chem. Res.* **2011**, *45*, 196−205.

(7) Wereszczynski, J.; McCammon, J. A. *Q. Rev. Biophys.* **2012**, *45*, 1−25.

(8) Šponer, J.; Cang, X.; Cheatham, T. E., III *Methods* **2012**, *57*, 25−39.

(9) Pierce, L. C.; Salomon-Ferrer, R.; de Oliveira, C. A. F.; McCammon, J. A.; Walker, R. C. *J. Chem. Theory Comput.* **2012**, *8*, 2997−3002.

(10) Skjevik, r. A.; Madej, B. D.; Walker, R.; Teigen, K. *J. Phys. Chem. B* **2012**, *116*, 11124−11136.

(11) Dickson, C. J.; Rosso, L.; Betz, R. M.; Walker, R.; Gould, I. R. *Soft Matter* **2012**, *8*, 9617−9627.

(12) Xu, D.; Williamson, M. J.; Walker, R. C. Advancements in Molecular Dynamics Simulations of Biomolecules on Graphical Processing Units. In *Annual Reports in Computational Chemistry*; Simmerling, C., Ed.; Elsevier: Amsterdam, The Netherlands, 2010; Vol. *6*, Chapter 1, pp 3−19.

(13) Götz, A. W.; Wölfle, T. M.;Walker, R. C. Quantum Chemistry on Graphics Processing Units. In *Annual Reports in Computational Chemistry*; Simmerling, C., Ed.; Elsevier: Amsterdam, The Netherlands, 2010; Vol. *6*, Chapter 2, pp 21−35.

(14) Harvey, M. J.; De Fabritiis, G. *WIREs Comput. Mol. Sci.* **2012**, *2*, 734−742.

(15) Garland, M.; Le Grand, S.; Nickolls, J.; Anderson, J.; Hardwick, J.; Morton, S.; Phillips, E.; Zhang, Y.; Volkov, V. *IEEE Micro.* **2008**, *28*, 13−27.

(16) Preis, T. *Eur. Phys. J.: Spec. Top.* **2011**, *194*, 87−119.

(17) Pratx, G.; Lei, X. *Med. Phys.* **2011**, *38*, 2685−2697.

(18) NVIDIA list of projects in different disciplines that use GPUs. https://developer.nvidia.com/cuda-action-research-apps (accessed October 2012).

(19) Phillips, J. C.; Braun, R.; Wang, W.; Gumbart, J.; Tajkhorshid, E.; Villa, E.; Chipot, C.; Skeel, R. D.; Kale, L.; Schulten, K. *J. Comput. Chem.* **2005**, *26*, 1781−1802.

(20) Stone, J. E.; Phillips, J. C.; Freddolino, P. L.; Hardy, D. J.; Trabuco, L. G.; Schulten, K. *J. Comput. Chem.* **2007**, *28*, 2618−2640.

(21) Case, D. et al. *AMBER 12*, University of California, San Francisco, 2012.

(22) Salomon-Ferrer, R.; Case, D. A.; Walker, R. C. *WIREs Comput. Mol. Sci.* **2013**, 3, 198−210.

(23) van der Spoel, D.; Lindahl, E.; Hess, B.; Groenhof, G.; Mark, A. E.; Berendsen, H. J. C. *J. Comput. Chem.* **2005**, *26*, 1701−1718.

(24) Hess, B.; Kutzner, C.; van der Spoel, D.; Lindahl, E. *J. Chem. Theory Comput.* **2008**, *4*, 435−447.

(25) Plimpton, S. *J. Comput. Phys.* **1995**, *117*, 1−19.

(26) Brooks, B. R.; et al. *J. Comput. Chem.* **2009**, *30*, 1545−1615.

(27) Harvey, M. J.; Giupponi, G.; Fabritiis, G. D. *J. Chem. Theory Comput.* **2009**, *5*, 1632−1639.

(28) Eastman, P.; et al. *J. Chem. Theory Comput.* **2013**, *9*, 461−469.

(29) Götz, A. W.; Williamson, M. J.; Xu, D.; Poole, D.; Le Grand, S.; Walker, R. C. *J. Chem. Theory Comput.* **2012**, *8*, 1542−1555.

(30) Allen, M. P.; Tildesley, D. J. *Computer Simulation of Liquids*; Oxford University Press: Oxford, 1991.

(31) Tironi, I. G.; Sperb, R.; Smith, P. E.; van Gunsteren, W. F. *J. Chem. Phys.* **1995**, *102*, 5451−5459.

(32) Wu, X.; Brooks, B. *J. Chem. Phys.* **2005**, *122*, 044107.

(33) Darden, T.; York, D.; Pedersen, L. *J. Chem. Phys.* **1993**, *98*, 10089−10092.

(34) Essmann, U.; Perera, L.; Berkowitz, M.; Darden, T.; Lee, H.; Pedersen, L. *J. Chem. Phys.* **1995**, *103*, 8577−8593.

(35) Hockney, R.; Eastwood, J. *Computer Simulation Using Particles*; Adam Hilger: Bristol, U.K., 1988; Chemistry; Computer Science; Physics (provided by Thomson Reuters).

(36) Le Grand, S.; Götz, A. W.; Walker, R. C. *Comput. Phys. Commun.* **2013**, *184*, 374−380.

(37) Cornell, W. D.; Cieplak, P.; Bayly, C. I.; Gould, I. R.; Merz, K. M.; Ferguson, D. M.; Spellmeyer, D. C.; Fox, T.; Caldwell, J. W.; Kollman, P. A. *J. Am. Chem. Soc.* **1995**, *117*, 5179−5197.

(38) Hornak, V.; Abel, R.; Okur, A.; Strockbine, B.; Roitberg, A.; Simmerling, C. *Proteins* **2006**, *65*, 712−725.

(39) Harvey, M. J.; Fabritiis, G. D. *J. Chem. Theory Comput.* **2009**, *5*, 2371−2377.

(40) Brown, W. M.; Kohlmeyer, A.; Plimpton, S. J.; Tharrington, A. N. *Comput. Phys. Commun.* **2012**, *183*, 449−459.

(41) Brown, W. M.; Wang, P.; Plimpton, S. J.; Tharrington, A. N. *Comput. Phys. Commun.* **2011**, *182*, 898−911.

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

(43) Eastman, P.; Pande, V. S. *J. Comput. Chem.* **2010**, *31*, 1268−1272.

(44) Hilbert, D. *Mathematische Annalen* **1891**, *38*, 459−460.

(45) NVIDIA CUFFT library. https://developer.nvidia.com/cufft (accessed October 2012).

(46) Miyamoto, S.; Kollman, P. A. *J. Comput. Chem.* **1992**, *13*, 952−962.

(47) Message Passing Interface Forum, MPI: *A Message-Passing Interface Standard*, Version 2.2; High-Performance Computing Center Stuttgart, University of Stuttgart: Stuttgart, Germany, 2009.

(48) Case, D. A. et al. *AMBER 11*, University of California: San Francisco, 2010.

(49) Bowers, K. J.; Chow, E.; Xu, H.; Dror, R. O.; Eastwood, M. P.; Gregersen, B. A.; Klepeis, J. L.; Kolossvary, I.; Moraes, M. A.; Sacerdoti, F. D.; Salmon, J. K.; Shan, Y.; Shaw, D. E. Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters. *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, New York, **2006**.

(50) Vijaykumar, S.; Bugg, C. E.; Wilkinson, K. D.; Cook, W. *J. Proc. Nat. Acad. Sci.* **1985**, *82*, 3582−3585.

(51) Vijaykumar, S.; Bugg, C. E.; Cook, W. J. *J. Mol. Bio.* **1987**, *194*, 531−544.

(52) Berendsen, H. J. C.; Postma, J. P. M.; van Gunsteren, W. F.; DiNola, A.; Haak, J. R. *J. Chem. Phys.* **1984**, *81*, 3684−3690.