Enhancing Amber for Use on the Blue Waters High-Performance Computing Resource

*Daniel R. Roe [a], Jason M. Swails [b], Romelia Salomon-Ferrer [c], Adrian E. Roitberg [b], and Thomas E. Cheatham III [a]**

a. Department of Medicinal Chemistry, L.S. Skaggs Pharmacy Institute, University of Utah, Salt Lake City, Utah 84112

b. Department of Chemistry, Quantum Theory Project, University of Florida, Gainesville, Florida 32611

c. San Diego Supercomputing Center, University of California San Diego, La Jolla, California 92093

Abstract: We describe enhancements to the Amber package of molecular simulation software made to take advantage of the parallel compute environment provided by the Blue Waters computational resource under the auspices of the NEIS-P2 program supporting the PRAC "Hierarchical molecular dynamics sampling for assessing pathways and free energies of RNA catalysis, ligand binding, and conformational change". We also discuss some recommendations of simulation settings for achieving better performance on Blue Waters.

Introduction

With the advent of modern high-performance computational resources, data can potentially be generated and analyzed faster and in greater volume than ever. However, existing software must be modified and/or enhanced to take advantage of this hardware, particularly when that software may have been originally developed with different hardware considerations in mind. The Amber molecular simulation software package[1] has been around for approximately three decades and has undergone significant changes, from initially being designed to run efficiently on low-memory single processor machines, to now being able to run on multiple CPUs or GPUs. Given the availability of petascale computational resources like Blue Waters, it is of great interest not only to modify software to run on resources efficiently, but to also use simulation methods that take the most advantage of the massively parallel layout of such resources.

In this report we describe several further changes and additions to various software packages within Amber (namely the MD engines SANDER and PMEMD, and the analysis software CPPTRAJ) made to take advantage of the hardware on Blue Waters. This includes the development of a multi-dimensional replica exchange molecular dynamics simulation method, changes to the Amber NetCDF trajectory and restart formats necessary to implement these methods, and modifications to the analysis software CPPTRAJ to recognize and process these coordinate files. We also discuss some of the issues encountered in implementing these changes, particularly as it relates to the impact of data input/output (IO) on performance, and provide general recommendations for running Amber on a resource like Blue Waters.

Porting of AMBER to Blue Waters

Blue Waters provides access to compilers from three major vendors: Cray, PGI, and GNU. There were several issues encountered when compiling with all three compilers, and no one compiler was able to provide access to all features of Amber. For the purposes of this discussion Amber code will be divided into code that only runs on CPUs (essentially all of Amber) and code that runs on GPUs (in this case only the CUDA code for PMEMD).

GNU compilers were able to compile both the CPU and GPU portions of Amber; however, the CPU versions of PMEMD and SANDER failed all tests related to particle mesh Ewald calculations (i.e. calculations with periodic boundaries). The issue appears to be related to compiling the fast Fourier transform (FFT) library bundled with Amber with optimizations turned on. Because of this, currently only the GPU code (i.e. CUDA PMEMD) is used from GNU compilations. The problems with the CPU can be overcome through use of the Cray FFTW module however we have tended to focus efforts on the PGI version for the CPU code to avoid having to change the build scripts and since the performance is equivalent.

PGI compilers were able to compile only the CPU code. Compilation of GPU code fails due to an incompatibility of PGI compilers with the CUDA header file 'host_defines.h'. However, unlike the GNU compilers, the PGI-compiled CPU code passes all tests. Because of this, currently all non-GPU code used on is PGI-compiled.

Amber initially could not be configured for Cray compilers. A Cray compiler target was created after much trial and error. It was found that for successful compilation, the flags '-emf' for producing module files with lower case names, '-f free' for free-format fortran code, and '-h gnu' for GNU compatibility were required. Despite this, only a handful of programs from Amber were able to be compiled successfully (including PMEMD and CPPTRAJ but not SANDER), as the Cray compilers appear to be much more strict in terms of allowable syntax. Initially, CPU PMEMD was found to fail tests in the same way as the GNU-compiled code. Compiling CPU PMEMD with Blue Waters version of FFTW3 (module load fftw) was found to correct this; therefore vendor-supplied FFTW3 is currently required to compile PMEMD. Compiling GPU code was not attempted with Cray compilers. The recommended configure and compile procedure (CPPTRAJ and PMEMD only) for Cray compilers on Blue Waters is:

```
cd $AMBERHOME
module load fftw
./configure -noX11 -nomtkpp -nofftw3 -norism cray
cd $AMBERHOME/AmberTools/src/cpptraj
make install
cd $AMBERHOME/src/pmemd
make install
```

To compile the MPI version of PMEMD, add the '-mpi' flag to configure and skip compile for CPPTRAJ. Performance of the Cray-compiled MPI-enabled PMEMD compared to GNU-compiled GPU code is shown in Figure 1. It should be noted that these results are from April 2013, and that currently full use of the node (i.e. PPN=32) scales much better (closer to PPN=16), although CPU performance still does not compare to GPU performance.

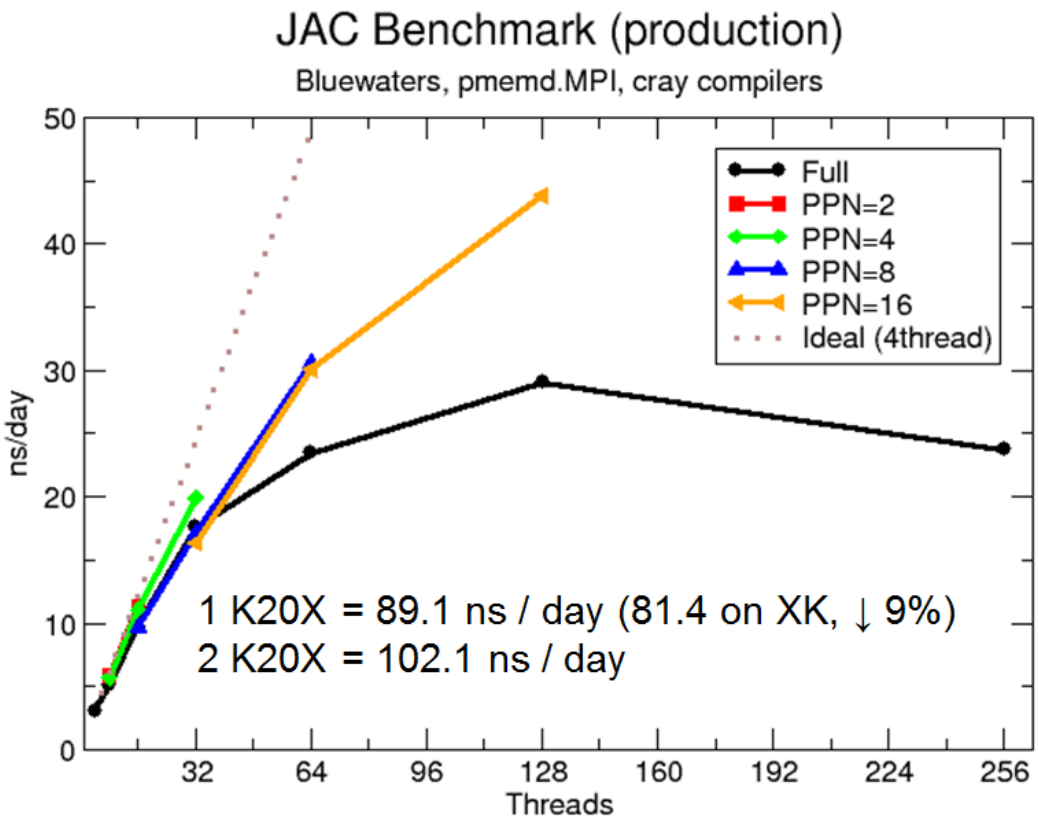

Figure 1: Performance of the AMBER PMEMD engine on Blue Waters for the joint-AMBER-CHARMM DHFR benchmark on the CPU and GPU nodes. PPN is cores running per node. Note that better performance is seen in nVidia K20X nodes when the error correcting is disabled (which is not enabled by default on Blue Waters).

Hamiltonian Replica Exchange: An Efficient Use of Parallel Resources

Traditionally when running calculations like molecular dynamics (MD) simulations in a parallel environment, the factor that limits overall calculation speed tends to be communication overhead between nodes (i.e. parallel scaling). Therefore it is of interest to perform as many calculations as possible on individual nodes while keeping communication between the nodes to a minimum. Replica Exchange MD^2 (REMD) is a commonly used enhanced sampling technique that is well-suited to a parallel environment like Blue Waters. Briefly, REMD involves simulating N non-interacting replicas of a system of interest, each with a Hamiltonian that differs in some way. After a certain amount of time, the probability of exchanging certain replicas is calculated according to Equation 1. The exchange is either accepted or rejected using the Metropolis criterion. The advantage of this method in a parallel environment is that communication between replicas occurs only when an exchange is attempted.

Equation 1: Probability of exchanging two replicas. $\boldsymbol{H_n}$ represents a Hamiltonian at $\boldsymbol{\beta_n}$ and $\boldsymbol{X_n}$ represents a set of atomic coordinates.

$$P_{exch}\left(H_i(X_1), H_j(X_2)\right) = exp\left[-\beta_i\left(H_i(X_2) - H_i(X_1)\right) - \beta_j\left(H_j(X_1) - H_j(X_2)\right)\right]$$

The most common type of REMD is temperature REMD (T-REMD), where every replica has the same Hamiltonian simulated at a different temperature. T-REMD has the advantage of being simple to implement. Since the only difference between Hamiltonians is temperature (i.e. $\boldsymbol{H_i}$ equals $\boldsymbol{H_j}$), only the temperature of each replica needs to be swapped when an exchange is performed and the velocities scaled accordingly, resulting in minimal communication between replicas. This means that the coordinate trajectory written out by each replica will contain frames at different temperatures, so the replica trajectories must be post-processed to obtain trajectories where each frame in the trajectory is at the same temperature (i.e. temperature trajectories). This is currently accomplished with either PTRAJ or CPPTRAJ[3] analysis software from Amber.

However, changing temperature may not always be sufficient to enhance sampling of certain processes, or one may be interested in how changing other parts of the Hamiltonian can enhance sampling. Therefore it is desirable to be able to perform a REMD simulation where the differences in the Hamiltonian can be anything from changes in torsional energy barriers to changing solvation models. In order to accomplish this, for each exchange attempt between Hamiltonian ***i*** with coordinates ***1***, $\boldsymbol{H_i(X_1)}$, and Hamiltonian ***j*** with coordinates ***2***, $\boldsymbol{H_j(X_2)}$, each Hamiltonian requires both $\boldsymbol{X_1}$ and $\boldsymbol{X_2}$ (see Equation 1), so $\boldsymbol{H_i}$ must be sent $\boldsymbol{X_2}$ and $\boldsymbol{H_j}$ must be sent $\boldsymbol{X_1}$, regardless of whether the exchange is accepted or not. This means there is significantly more communication between replicas compared to T-REMD, so for this method to remain efficient a relatively fast network interconnect is required. However, the benefit of this generalized Hamiltonian REMD (H-REMD) is that users have much more flexibility when choosing how to enhance sampling in their systems. In addition, since the coordinates are swapped whenever an exchange is accepted each replica writes out a sorted trajectory for a given Hamiltonian; no post processing is required. Both T-REMD and H-REMD are currently implemented in Amber 12.

Extension of Accelerated Molecular Dynamics for use with H-REMD

Accelerated molecular dynamics[4] (AMD) is another enhanced sampling technique that functions by applying a so-called "boost" potential to the system when the energy is below a certain threshold in order to "accelerate" the system out of energy minima. This increases sampling by inducing the system to spend less time in minima (where it may become "trapped"), allowing energy barriers to be crossed more frequently. However, obtaining unbiased distributions obtained from an AMD simulation can be challenging[5]. A potential solution to this issue is to combine AMD with H-REMD, where the first replica is not "boosted" and so no explicit unbiasing of results is required. The AMD+H-REMD approach has been implemented previously using SANDER and external python wrappers[6]. The reason for this is most likely because although AMD is currently implemented in SANDER and PMEMD[7], the AMD boost energy is not included in the overall potential energy, which is required for H-REMD to function properly (otherwise the Hamiltonians are identical).

While the external wrapper approach may work, it certainly does not make efficient use of parallel resources since at every exchange execution of the code must be stopped and then

restarted after the exchange is completed. Therefore PMEMD was modified so that the AMD boost energy was included in the total potential energy. This involved modifying the AMD calculation routines to return total boost energy, extending the energy arrays in PMEMD to include a place for the boost energy, ensuring the boost energy was summed into the total for both GB and PME force calculations, and modifying the output so that the boost energy term was reported. These changes are currently in the Amber GIT master branch. Local copies of the Amber GIT are currently on Blue Waters in our project space:

```
/mnt/b/projects/sciteam/jn6/GIT
```

This code will be released to the general community with the next official AMBER release and in the meantime can be requested from the AMBER developers or Cheatham.

Note that PMEMD was chosen for modification as it has a significant speed advantage over SANDER. Note that currently only the CPU version of PMEMD works for AMD+H-REMD. Future plans include modifying the GPU AMD routines to return AMD boost energy as well.

It is important to note that when running AMD+H-REMD the choice of parameters controlling AMD is not straightforward. Parameters must be chosen with care so that the potential energy distribution of Hamiltonians overlaps such that a reasonable number of exchanges are accepted. Unlike a single AMD simulation where parameters are chosen so the boost potential is only activated some of the time[5], in H-REMD parameters should be chosen so that in the boosted replicas the boost potential is always active (otherwise the Hamiltonians will overlap with the unbiased one). Thus far, we have not been able to develop a precise scheme for determining AMD parameters for achieving certain exchange acceptance rates in AMD+H_REMD beyond trial and error, although this remains an active area of research.

Development of Multi-dimensional Replica Exchange

In the descriptions of T-REMD and H-REMD above, exchanges occur in only one dimension, i.e. only one part of the Hamiltonian is being altered between replicas. However, it may be desirable to attempt exchanges in two (or more) dimensions. For example, in one dimension replicas could differ by temperature, and in another dimension they could differ by AMD boost potentials. This type of simulation where multiple replica exchange dimensions are coupled is referred to as multi-dimensional REMD (M-REMD). Because conformational sampling can be enhanced in more than one dimension in M-REMD, aspects of the system dynamics that may not be improved by one dimension (such as protein folding and temperature) could potentially be improved by the other dimension. Since increasing the number of dimensions can rapidly increase the number of replicas required, a larger number of computational nodes are also required compared to one-dimensional REMD simulations, and if any of the dimensions are Hamiltonian (which is likely) the communication between those nodes needs to be fast. Also, if any dimension is temperature, trajectories will have to be sorted if data is to be processed at a single temperature. This requires that every trajectory frame contain information on which dimensions it is in. Similarly, the restart files used to checkpoint the simulation must contain the same information. This means both the trajectory and restart formats must be extended and any analysis software must be modified to read and process the new formats.

M-REMD has been implemented in the Amber MD engines SANDER and PMEMD and makes use of the existing T-REMD and H-REMD exchange functions. Exchanges are attempted in each dimension in turn so that the first exchange is attempted in the first dimension, the second exchange is attempted in the second dimension, and so on. This is illustrated in the following pseudo-code:

```
multid_exchange(n_exchange) {
  my_dim = n_exchange % remd_dimension;
  switch ( remd_types[my_dim] ) {
    case 1: temperature_exchange(); break;
    case 3: hamiltonian_exchange(); break;
  }
```

Here `n_exchange` is the number of the exchange being attempted, `remd_dimension` is the total number of replica dimensions, and `remd_types` is an array of size n_dimension containing a numeric index corresponding to the type of exchange to be performed in that dimension. Currently the code is only set up for temperature or Hamiltonian exchange, but could be extended to have different exchange types in the future.

M-REMD is enabled by specifying '-rem -1' on the command line. As with T-REMD or H-REMD, individual replicas are defined in a file called the group file, with each line defining the input file, input coordinates, topology file, etc. for each replica. In M-REMD there is an additional file called the dimension file, which defines both the replica dimensions and which replicas are allowed to exchange within each dimension. This file follows the same Fortran name list format that Amber MD input files use; there is a title followed by a '`&multirem`' name list for each replica dimension with format:

```
Dimension X Title
&multirem
  exch_type = <Exchange Type>,
  group(1,:) = <Group 1 Replica List>,
  …
  group(N,:) = <Group N Replica List>
  desc = 'Dimension X description'
/
```

Exchange type (**`exch_type`**) is currently restricted to either 'TEMPERATURE' or 'HAMILTONIAN'. A "Replica List" consists of a comma-separated list of integers corresponding to positions in the group file, where replica 1 is the first entry in the group file. A "Replica List" defines which replicas are allowed to exchange within that dimension.

In order to facilitate post-processing of trajectories from M-REMD simulations, each frame must record where it is in the ensemble (i.e. its index in each replica dimension). Extending the Amber ASCII trajectory/restart formats would require that every Amber trajectory/restart parser (including third-party ones) would have to be updated. Since NetCDF files are by their nature orthogonal adding additional information to the format does not break existing parsers (the additional information is simply ignored). Primarily because of this, it was decided that only the

Amber NetCDF trajectory and restart formats would be updated for M-REMD, and thus all M-REMD runs require use of the Amber NetCDF trajectory and restart formats.

Only three additional pieces of data were required to be added to the NetCDF trajectory/restart formats: an integer containing the number of replica dimensions (N), an integer array (of dimension N) containing the type of each dimension, and an array (of dimension FxN where F is the total number of frames) containing the indices of a replica in each dimension for each frame. All of these changes have been made and are currently available in the Amber GIT master branch.

Modification of Analysis Software for Processing M-REMD Trajectories

As mentioned above, the Amber NetCDF trajectory format was modified for M-REMD simulations to include information on replica dimensions to aid in post-processing trajectories. This required also modifying analysis software to recognize and use the extra information. Since CPPTRAJ is able to process NetCDF trajectories faster than PTRAJ, only CPPTRAJ was modified to read M-REMD trajectories.

Currently, CPPTRAJ has the ability to read in an ensemble of replica trajectories and process only frames at a user-specified position in the ensemble (referred to as replica trajectory processing). For example, to process frames at 300.0 K from a T-REMD ensemble of trajectories of which **`rem.001`** represents the first replica, the following CPPTRAJ input would be used:

```
trajin rem.001 remdtraj remdtrajtemp 300.0
```

Since an M-REMD trajectory may contain any number of dimensions, a slightly more general syntax is needed for choosing frames that correspond to only one index in each dimension. This is now accomplished with the remdtrajidx keyword:

```
remdtrajidx <dim1 index>,<dim2 index>,…,<dimN index>
```

For example, given M-REMD trajectories in which the first dimension is temperature and the second dimension is Hamiltonian, to pick frames at the first temperature and 4th Hamiltonian the following CPPTRAJ input would be used:

```
trajin rem.001 remdtraj remdtrajidx 1,4
```

Note that CPPTRAJ has been updated to also be able to read the new M-REMD NetCDF restart format as well.

Ensemble Trajectory Processing

Instead of processing frames at one level of the ensemble, in some cases it may be desirable to process all levels of the ensemble at once. This has been implemented via the '**`ensemble`**' command in CPPTRAJ. It is similar to the original replica trajectory processing in that an ensemble of replica trajectories is read in. However, instead of only one frame at a user-specified target being selected from the ensemble for processing, all frames are used for processing. Sorting of frames is performed if necessary, then any specified actions are run on every member

of the sorted ensemble. The output from a given action for all members of the ensemble is written to the same file. If coordinate output is desired, coordinates are written to sorted trajectories with numeric suffixes indicating their overall position in the ensemble. Ensemble analysis can be performed with T-REMD, H-REMD, or M-REMD trajectories.

For example, consider an ensemble of 4 replica trajectories from a T-REMD simulation named rem.1, rem.2, rem.3, and rem.4 and the following CPPTRAJ input:

```
ensemble rem.1
strip :WAT
rms first out rmsd.agr @CA
trajout nowat.nc netcdf
```

This would (in order) read in all four trajectories and sort them by temperature, remove water residues, calculate the coordinate RMSD of the sorted replica trajectories to the first frame based on CA atom positions, and write four sorted NetCDF trajectories named **`nowat.nc.X`** where **`X`** ranges from 0-3 and corresponds to temperatures in the ensemble from lowest to highest.

On Blue Waters, ensemble trajectory processing initially proved to be very slow since all coordinate IO was being done via only one node. Because of this, ensemble analysis was parallelized with MPI so that one node per replica was responsible for reading and processing trajectory frames. First, frame ***X,M*** (X = frame #, M = frame position in the ensemble) from replica trajectory ***N*** is read by node ***N***. If the ensemble needs to be sorted, every node sends to every other node the position in the overall ensemble ***M*** of the frame it read. Then every replica that does not have the correctly sorted frame (i.e. ***M*** does not equal ***N***) sends its coordinates to node ***M*** and receives its coordinates from the node containing frame ***N***. When coordinate sorting is complete node ***N*** has the frame with ***M*** equal to ***N***. On each node space is allocated for two frames (referred to as frame 0 and frame 1). The frame ***X,M*** is initially read into frame 0. If coordinates need to be sorted, the sorted frame received is stored in frame 1. An index is set to 1 or 0 depending on whether a frame was received or not respectively; further trajectory processing uses this frame.

Table 1: Timings for serial vs parallel ensemble processing with CPPTRAJ.

Type	Dimensions	Replicas	Frames	Serial (s)	Parallel (s)	Speedup
H-REMD	1	8	60000	25672	3910	6.57
T-REMD	1	60	100	2859	65	43.98
M-REMD	2	192	1000	2507	42	59.69

The performance of ensemble processing in parallel versus in serial is shown in Table 1. The speedup for processing H-REMD trajectories of 6.57 is close to the ideal speedup of 8 since these trajectories do not require sorting (hence communication between nodes is minimal). The speedup for processing T-REMD and M-REMD trajectories are far less than ideal since multiple coordinate exchanges happen on average for every frame processed, but the speedup is still an incredible 2 orders of magnitude improvement over the serial timings.

NetCDF: Implications for Performance

When running REMD simulations on Blue Waters, it was found that overall performance (measured in terms of ns of simulation time per day) would vary over the course of the simulation by an extreme amount; e.g. for one system run performance would fluctuate from ~60 ns/day to ~120 ns/day (see Figure 2).

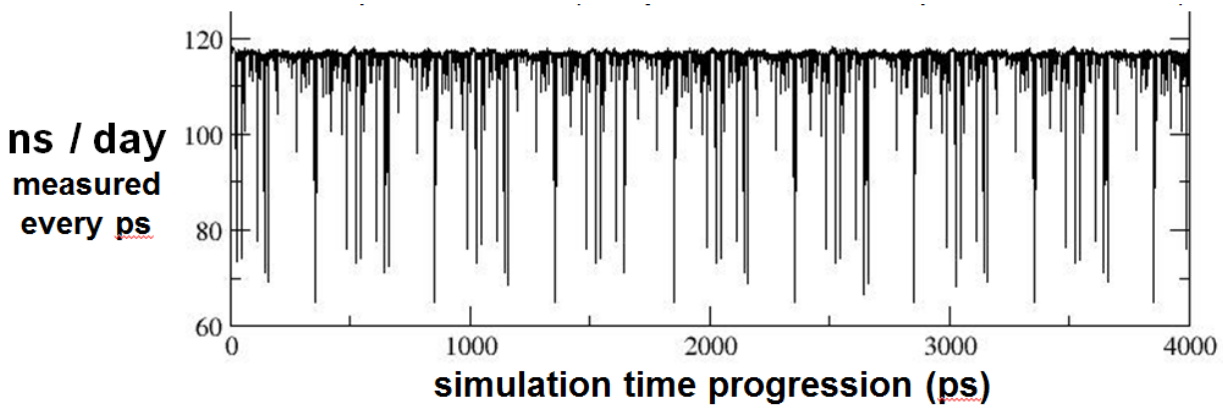

Figure 2: Variable performance on Blue Waters due to I/O issues.

To ascertain if this was related to communication lag between nodes or input/output (IO), two REMD simulations were run, one with exchanges (requiring communication between nodes) and without IO, the other without exchanges and with IO. The simulation with exchanges and without IO showed no slowdown, indicating the issue was related to IO. During the course of a simulation, several files are written, including the output file which contains various simulation data (chiefly energies), the trajectory file containing coordinates, and the restart file which is used to checkpoint the simulation. To ascertain exactly what IO was causing the observed slowdown several test REMD simulations were run using a solvated single nucleic acid (8000 atoms): one with all IO disabled, one with all IO enabled, one with only the output file being written, and one with the output and trajectory files being written; the results are shown in Table 2. It was found that appreciable slowdown occurred only when writing of restart files was enabled. Since writing NetCDF trajectory files did not appear to slow down simulations, a final test was run where all IO was enabled but NetCDF restarts were written instead of ASCII ones. It was found that the use of NetCDF restarts resulted in almost no slowdown. Therefore it is recommended that when running any simulations with Amber both NetCDF trajectories and restarts be used.

Table 2: REMD simulation timings (no exchanges) for various IO combinations.

Simulation	**Restart Type**	**Speed (ns/day)**
All IO Disabled	n/a	66-68
All IO Enabled	ASCII	20-23
Only output file	n/a	67-68
Only output/trajectory files	n/a	64-66
All IO Enabled	NetCDF	62-64

Conclusion

In summary, changes to the Amber code functionality and performance were engineered to more efficiently utilize the Blue Waters computational resource. With the new technologies, the associated groups are pushing forward to better understand biomolecular structure, dynamics and interactions with a current focus on improving and understanding the multi-dimensional replica-exchange methods and also assessing, validating and improving nucleic acid force fields.

Corresponding Author

*Address correspondence to tec3@utah.edu

Author Contributions

The manuscript was written through contributions of all authors. All authors have given approval to the final version of the manuscript. Roe implemented M-REMD trajectory and NetCDF restart formats, enabled AMD+HREMD, and all related analysis code in CPPTRAJ. Salomon-Ferrer implemented the original version of AMD. Swails implemented H-REMD and M-REMD code for SANDER and PMEMD.

Acknowledgements

This research is part of the Blue Waters sustained-petascale computing project and its NEIS-P2 effort, which is supported by the National Science Foundation (award number OCI 07-25070) and the state of Illinois and also the NSF PRAC award “Hierarchical molecular dynamics sampling for assessing pathways and free energies of RNA catalysis, ligand binding, and conformational change” (award number OCI 1036208). Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

References

1. Case, D. A.; Cheatham, T. E., 3rd; Darden, T.; Gohlke, H.; Luo, R.; Merz, K. M., Jr.; Onufriev, A.; Simmerling, C.; Wang, B.; Woods, R. J., The Amber biomolecular simulation programs. *Journal of Computational Chemistry* **2005,** *26* (16), 1668-1688.

2. Sugita, Y.; Okamoto, Y., Replica-exchange molecular dynamics method for protein folding. *Chemical Physics Letters* **1999,** *314* (1–2), 141-151.
3. Roe, D. R.; Cheatham, T. E., 3rd, PTRAJ and CPPTRAJ: Software for Processing and Analysis of Molecular Dynamics Trajectory Data. *Journal of Chemical Theory and Computation* **2013,** *Just Accepted Manuscript*.
4. Hamelberg, D.; Mongan, J.; McCammon, J. A., Accelerated molecular dynamics: A promising and efficient simulation method for biomolecules. *The Journal of Chemical Physics* **2004,** *120* (24), 11919-11929.
5. Markwick, P. R. L.; McCammon, J. A., Studying functional dynamics in bio-molecules using accelerated molecular dynamics. *Physical Chemistry Chemical Physics* **2011,** *13* (45), 20053-20065.
6. Arrar, M.; de Oliveira, C. A. F.; Fajer, M.; Sinko, W.; McCammon, J. A., w-REXAMD: A Hamiltonian Replica Exchange Approach to Improve Free Energy Calculations for Systems with Kinetically Trapped Conformations. *Journal of Chemical Theory and Computation* **2012,** *9* (1), 18-23.
7. Pierce, L. C. T.; Salomon-Ferrer, R.; Augusto F. de Oliveira, C.; McCammon, J. A.; Walker, R. C., Routine Access to Millisecond Time Scale Events with Accelerated Molecular Dynamics. *Journal of Chemical Theory and Computation* **2012,** *8* (9), 2997-3002.