# NAMD User's Guide

Version Git-2025-12-04

Rafael Bernardi, Milind Bhandarkar, Abhinav Bhatele, Eric Bohm, W. Michael Brown, Robert Brunner, Ronak Buch, Floris Buelens, Haochuan Chen, Christophe Chipot, Jaemin Choi, Han-Yi Chou, David Clark, Andrew Dalke, Surjit Dixit, Giacomo Fiorin, Peter Freddolino, Haohao Fu, Paul Grayson, Justin Gullingsrud, Attila Gursoy, David Hardy, Chris Harrison, Jerome Hénin, William Humphrey, David Hurwitz, Antti-Pekka Hynninen, Barry Isralewitz, Nikhil Jain, Wei Jiang, Neal Krawetz, Sameer Kumar, David Kunzman, Jonathan Lai, Chee Wai Lee, Julio Maia, Tareq Malas, Ryan McGreevy, Chao Mei, Marcelo Melo, Sarah Moe, Mark Nelson, James Phillips, Brian Radak, Joao Ribeiro, Till Rudack, Osman Sarood, Aritomo Shinozaki, Mohammad Soroush Barhaghi, David Tanner, Josh Vermaas, Peng Wang, Yi Wang, David Wells, Ke Yue, Gengbin Zheng, Fangqiang Zhu

December 4, 2025

Theoretical Biophysics Group
University of Illinois and Beckman Institute
405 N. Mathews
Urbana, IL 61801

## Description

The NAMD *User's Guide* describes how to run and use the various features of the molecular dynamics program NAMD. This guide includes the capabilities of the program, how to use these capabilities, the necessary input files and formats, and how to run the program both on uniprocessor machines and in parallel.

NAMD development is supported by National Institutes of Health grants NIH P41-GM104601 and NIH R24-GM145965.

# NAMD Version Git-2025-12-04

Authors: Rafael Bernardi, Milind Bhandarkar, Abhinav Bhatele, Eric Bohm, W. Michael Brown, Robert Brunner, Ronak Buch, Floris Buelens, Haochuan Chen, Christophe Chipot, Jaemin Choi, Han-Yi Chou, David Clark, Andrew Dalke, Surjit Dixit, Giacomo Fiorin, Peter Freddolino, Haohao Fu, Paul Grayson, Justin Gullingsrud, Attila Gursoy, David Hardy, Chris Harrison, Jerome Hénin, William Humphrey, David Hurwitz, Antti-Pekka Hynninen, Barry Isralewitz, Nikhil Jain, Wei Jiang, Neal Krawetz, Sameer Kumar, David Kunzman, Jonathan Lai, Chee Wai Lee, Julio Maia, Tareq Malas, Ryan McGreevy, Chao Mei, Marcelo Melo, Sarah Moe, Mark Nelson, James Phillips, Brian Radak, Joao Ribeiro, Till Rudack, Osman Sarood, Aritomo Shinozaki, Mohammad Soroush Barhaghi, David Tanner, Josh Vermaas, Peng Wang, Yi Wang, David Wells, Ke Yue, Gengbin Zheng, Fangqiang Zhu

Theoretical and Computational Biophysics Group, Beckman Institute, University of Illinois.

## NAMD Molecular Dynamics Software
## Non-Exclusive, Non-Commercial Use License

### Introduction

The University of Illinois at Urbana-Champaign has created its molecular dynamics software, NAMD, developed by the Theoretical and Computational Biophysics Group ("TCBG") at Illinois' Beckman Institute available free of charge for non-commercial use by individuals, academic or research institutions and corporations for in-house business purposes only, upon completion and submission of the online registration form presented when attempting to download NAMD at the web site http://www.ks.uiuc.edu/Research/namd/.

Commercial use of the NAMD software, or derivative works based thereon, REQUIRES A COMMERCIAL LICENSE. Commercial use includes: (1) integration of all or part of the Software into a product for sale, lease or license by or on behalf of Licensee to third parties, or (2) distribution of the Software to third parties that need it to commercialize product sold or licensed by or on behalf of Licensee. The University of Illinois will negotiate commercial-use licenses for NAMD upon request. These requests can be directed to namd@ks.uiuc.edu

### Online Download Registration Requirements

In completing the online registration form presented before downloading individuals may register in their own name or with their institutional or corporate affiliations. Registration information must include name, title, and e-mail of a person with signature authority to authorize and commit the individuals, academic or research institution, or corporation as necessary to the terms and conditions of the license agreement.

All parts of the information must be understood and agreed to as part of completing the form. Completion of the form is required before software access is granted. Pay particular attention to the authorized requester requirements above, and be sure that the form submission is authorized by the duly responsible person.

## UNIVERSITY OF ILLINOIS
## NAMD MOLECULAR DYNAMICS SOFTWARE LICENSE AGREEMENT

Upon execution of this Agreement by the party identified below ("Licensee"), The Board of Trustees of the University of Illinois ("Illinois"), on behalf of The Theoretical and Computational Biophysics Group ("TCBG") in the Beckman Institute, will provide the molecular dynamics software NAMD in Executable Code and/or Source Code form ("Software") to Licensee, subject to the following terms and conditions. For purposes of this Agreement, Executable Code is the compiled code, which is ready to run on Licensee's computer. Source code consists of a set of files which contain the actual program commands that are compiled to form the Executable Code.

1. The Software is intellectual property owned by Illinois, and all right, title and interest, including copyright, remain with Illinois. Illinois grants, and Licensee hereby accepts, a restricted, non-exclusive, non-transferable license to use the Software for academic, research and internal business purposes only e.g. not for commercial use (see Paragraph 7 below), without a fee. Licensee agrees to reproduce the copyright notice and other proprietary markings on all copies of the Software. Licensee has no right to transfer or sublicense the Software to any unauthorized person or entity. However, Licensee does have the right to make complimentary works that interoperate with NAMD, to freely distribute such complimentary works, and to direct others to the TCBG server to obtain copies of NAMD itself.

2. Licensee may, at its own expense, modify the Software to make derivative works, for its own academic, research, and internal business purposes. Licensee's distribution of any derivative work is also subject to the same restrictions on distribution and use limitations that are specified herein for Illinois' Software. Prior to any such distribution the Licensee shall require the recipient of the Licensee's derivative work to first execute a license for NAMD with Illinois in accordance with the terms and conditions of this Agreement. Any derivative work should be clearly marked and renamed to notify users that it is a modified version and not the original NAMD code distributed by Illinois.

3. Except as expressly set forth in this Agreement, THIS SOFTWARE IS PROVIDED "AS IS" AND ILLINOIS MAKES NO REPRESENTATIONS AND EXTENDS NO WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OR MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY PATENT, TRADEMARK, OR OTHER RIGHTS. LICENSEE ASSUMES THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE AND/OR ASSOCIATED MATERIALS. LICENSEE AGREES THAT UNIVERSITY SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, OR INCIDENTAL DAMAGES WITH RESPECT TO ANY CLAIM BY LICENSEE OR ANY THIRD PARTY ON ACCOUNT OF OR ARISING FROM THIS AGREEMENT OR USE OF THE SOFTWARE AND/OR ASSOCIATED MATERIALS.

4. Licensee understands the Software is proprietary to Illinois. Licensee agrees to take all reasonable steps to insure that the Software is protected and secured from unauthorized disclosure, use, or release and will treat it with at least the same level of care as Licensee would use to protect and secure its own proprietary computer programs and/or information, but using no less than a reasonable standard of care. Licensee agrees to provide the Software only to any other person or entity who has registered with Illinois. If licensee is not registering as an individual but as an institution or corporation each member of the institution or corporation who has access to or uses Software must understand and agree to the terms of this license. If Licensee becomes aware of any unauthorized licensing, copying or use of the Software, Licensee shall promptly notify Illinois in

## Contact Information

The best contact path for licensing issues is by e-mail to namd@ks.uiuc.edu or send correspondence to:

NAMD Team
Theoretical and Computational Biophysics Group
Beckman Institute
University of Illinois
405 North Mathews MC-251
Urbana, Illinois 61801 USA

# Contents

## List of Figures

## 1 Introduction

NAMD is a parallel molecular dynamics program for UNIX platforms designed for high-performance simulations in structural biology. This document describes how to use NAMD, its features, and the platforms on which it runs. The document is divided into several sections:

**Section 1** gives an overview of NAMD.

**Section 2** lists the basics for getting started.

**Section 3** describes NAMD file formats.

**Section 4** explains PSF file generation with psfgen.

**Section 5** presents the potential functions, non-bonded interactions, and full electrostatics.

**Section 6** explains Generalized Born implicit solvent simulations.

**Section 7** lists standard minimization and dynamics parameters.

**Section 16** lists performance tuning parameters.

**Section 8** explains user defined forces. conformation change calculations.

**Section 9** describes collective variable-based calculations.

**Section 10** explains alchemical free energy calculations.

**Section 11** presents accelerated sampling methods.

**Section 15** lists runtime analysis options.

**Section 17** provides hints for X-PLOR users.

**Section 18** provides sample configuration files.

**Section 19** gives details on running NAMD.

**Section 20** gives details on installing NAMD.

### 1.1 NAMD and molecular dynamics simulations

Molecular dynamics (MD) simulations compute atomic trajectories by solving equations of motion numerically using empirical force fields, such as the CHARMM force field, that approximate the actual atomic force in biopolymer systems. Detailed information about MD simulations can be found in several books such as [1, 48]. In order to conduct MD simulations, various computer programs have been developed including X-PLOR [12] and CHARMM [11]. These programs were originally developed for serial machines. Simulation of large molecules, however, require enormous computing power. One way to achieve such simulations is to utilize parallel computers. In recent years, distributed memory parallel computers have been offering cost-effective computational power. NAMD was designed to run efficiently on such parallel machines for simulating large molecules. NAMD is particularly well suited to the increasingly popular Beowulf-class PC clusters, which are quite similar to the workstation clusters for which is was originally designed. Future versions of NAMD will also make efficient use of clusters of multi-processor workstations or PCs.

NAMD has several important features:

- **Force Field Compatibility**
  The force field used by NAMD is the same as that used by the programs CHARMM [11] and X-PLOR [12]. This force field includes local interaction terms consisting of bonded interactions between 2, 3, and 4 atoms and pairwise interactions including electrostatic and van der Waals forces. This commonality allows simulations to migrate between these three programs.

- **Efficient Full Electrostatics Algorithms**
  NAMD incorporates the Particle Mesh Ewald (PME) algorithm, which takes the full electrostatic interactions into account. This algorithm reduces the computational complexity of electrostatic force evaluation from $O(N^2)$ to $O(N \log N)$.

- **Multiple Time Stepping**
  The velocity Verlet integration method [1] is used to advance the positions and velocities of the atoms in time. To further reduce the cost of the evaluation of long-range electrostatic forces, a multiple time step scheme is employed. The local interactions (bonded, van der Waals and electrostatic interactions within a specified distance) are calculated at each time step. The longer range interactions (electrostatic interactions beyond the specified distance) are only computed less often. This amortizes the cost of computing the electrostatic forces over several timesteps. A smooth splitting function is used to separate a quickly varying short-range portion of the electrostatic interaction from a more slowly varying long-range component. It is also possible to employ an intermediate timestep for the short-range non-bonded interactions, performing only bonded interactions every timestep.

- **Input and Output Compatibility**
  The input and output file formats used by NAMD are identical to those used by CHARMM and X-PLOR. Input formats include coordinate files in PDB format [4], structure files in X-PLOR PSF format, and energy parameter files in either CHARMM or X-PLOR formats. Output formats include PDB coordinate files and binary DCD trajectory files. These similarities assure that the molecular dynamics trajectories from NAMD can be read by CHARMM or X-PLOR and that the user can exploit the many analysis algorithms of the latter packages.

- **Dynamics Simulation Options**
  MD simulations may be carried out using several options, including
  - Constant energy dynamics,
  - Constant temperature dynamics via
    - Velocity rescaling,
    - Velocity reassignment,
    - Langevin dynamics,
  - Periodic boundary conditions,
  - Constant pressure dynamics via
    - Berendsen pressure coupling,
    - Nosé-Hoover Langevin piston,
  - Energy minimization,
  - Fixed atoms,

  - Rigid waters,
  - Rigid bonds to hydrogen,
  - Harmonic restraints,
  - Spherical or cylindrical boundary restraints.

- **Easy to Modify and Extend**
  Another primary design objective for NAMD is extensibility and maintainability. In order to achieve this, it is designed in an object-oriented style with C++. Since molecular dynamics is a new field, new algorithms and techniques are continually being developed. NAMD's modular design allows one to integrate and test new algorithms easily. If you are contemplating a particular modification to NAMD you are encouraged to contact the developers for guidance.

- **Interactive MD simulations**
  A system undergoing simulation in NAMD may be viewed and altered with VMD; for instance, forces can be applied to a set of atoms to alter or rearrange part of the molecular structure. For more information on VMD, see `http://www.ks.uiuc.edu/Research/vmd/`.

- **Load Balancing**
  An important factor in parallel applications is the equal distribution of computational load among the processors. In parallel molecular simulation, a spatial decomposition that evenly distributes the computational load causes the region of space mapped to each processor to become very irregular, hard to compute and difficult to generalize to the evaluation of many different types of forces. NAMD addresses this problem by using a simple uniform spatial decomposition where the entire model is split into uniform cubes of space called *patches*. An initial load balancer assigns patches and the calculation of interactions among the atoms within them to processors such that the computational load is balanced as much as possible. During the simulation, an incremental load balancer monitors the load and performs necessary adjustments.

## 1.2 Acknowledgments

NAMD development is supported by the National Institutes of Health (NIH P41-GM104601) and relies on computational resources funded by the National Science Foundation and the Department of Energy.

The authors would particularly like to thank the members of the Theoretical and Computational Biophysics Group, past and present, who have helped tremendously in making suggestions, pushing for new features, and testing bug-ridden code.

# 2 Getting Started

## 2.1 What is needed

Before running NAMD, explained in section 19, the following are be needed:

- A CHARMM force field in either CHARMM or X-PLOR format.
- An X-PLOR format PSF file describing the molecular structure.
- The initial coordinates of the molecular system in the form of a PDB file.
- A NAMD configuration file.

NAMD provides the `psfgen` utility, documented in Section 4, which is capable of generating the required PSF and PDB files by merging PDB files and guessing coordinates for missing atoms. If `psfgen` is insufficient for your system, we recommend that you obtain access to either CHARMM or X-PLOR, both of which are capable of generating the required files.

## 2.2 NAMD configuration file

Besides these input and output files, NAMD also uses a file referred to as the *configuration file*. This file specifies what dynamics options and values that NAMD should use, such as the number of timesteps to perform, initial temperature, etc. The options and values in this file control how the system will be simulated. The NAMD configuration file is specified on the NAMD command line, either before or after the various parallel execution options described in section 19.

A NAMD configuration file contains a set of options and values. The options and values specified determine the exact behavior of NAMD, what features are active or inactive, how long the simulation should continue, etc. Section 2.2.1 describes how options are specified within a NAMD configuration file. Section 2.2.5 lists the parameters which are required to run a basic simulation. Section 17 describes the relation between specific NAMD and X-PLOR dynamics options. Several sample NAMD configuration files are shown in section 18.

During execution NAMD will change to the directory containing the configuration file so that all file paths in the configuration file are relative to the configuration file directory. Multiple configuration files may be specified on the command line and the will be read in order, but all file paths will be relative to the first configuration file to call a "run" (or "minimize" or "startup") command, or to the last configuration file if "run" is not called.

Commands or parameters may also be specified directly on the command line via `--keyword value` argument pairs, for example `--outputenergies 100 --run 100 -- checkpoint`. This may be used to include multiple configuration files without altering the working directory via `--source /path/to/second.conf`. Note that escaping or quoting of command line parameter values containing spaces may be difficult or impossible on some systems due to multiple levels of scripts called during the NAMD parallel launch process and because the keyword and value are simply merged into a single string that is passed to the Tcl interpreter.

If the argument `--tclmain` is present, all following arguments will be passed to the Tcl interpreter as a script file and arguments accessible via the standard argc and argv variables. Note that Charm++ arguments such as `+pemap` are processed during Charm++ startup and will not be passed to Tcl.

If the first argument is `+tclsh`, Charm++ argument parsing and startup are not performed, the Tcl interpreter is initialized without NAMD scripting features, and all following arguments are passed to Tcl. Statically linked packages such as psfgen are available via "package require ...".

### 2.2.1 Configuration parameter syntax

Each line in the configuration files consists of a *keyword* identifying the option being specified, and a *value* which is a parameter to be used for this option. The keyword and value can be separated by only white space:

```
keyword            value
```

or the keyword and value can be separated by an equal sign and white space:

```
keyword      =     value
```

Blank lines in the configuration file are ignored. Comments are prefaced by a `#` and may appear on the end of a line with actual values:

```
keyword            value          #  This is a comment
```

or may be at the beginning of a line:

```
#  This entire line is a comment . . .
```

Some keywords require several lines of data. These are generally implemented to either allow the data to be read from a file:

```
keyword            filename
```

or to be included inline using Tcl-style braces:

```
keyword {
  lots of data
}
```

The specification of the keywords is case insensitive so that any combination of upper and lower case letters will have the same meaning. Hence, `DCDfile` and `dcdfile` are equivalent. The capitalization in the values, however, may be important. Some values indicate file names, in which capitalization is critical. Other values such as `on` or `off` are case insensitive.

### 2.2.2 Tcl scripting interface and features

When compiled with Tcl (all released binaries) the config file is parsed by Tcl in a fully backwards compatible manner with the added bonus that any Tcl command may also be used. This alone allows:

- the "source" command to include other files (works w/o Tcl too!),
- the "print" command to display messages ("puts" to stdout fails on some platforms),
- environment variables through the env array ("$env(USER)"), and

- user-defined variables (“set base sim23”, “dcdfile $base.dcd”).

Additional features include:

- The “run” command takes a number of steps to run (overriding the now optional numsteps parameter, which defaults to 0) and can be called repeatedly. You can “run 0” just to get energies. Normally the preceeding timestep is repeated to account for any modifications to the energy function; this can be avoided with “run norepeat” followed by the next number of steps to take.

- The “minimize” command is similar to “run” and performs minimization for the specified number of force evaluations.

- The “callback” command takes a 2-parameter Tcl procedure which is then called with a list of labels and a list of values, including but not limited to the current step, energies, temperature, pressure and volume, after every “run” procedure, allowing analysis, formatting, whatever. The following example illustrates using callbacks while clarifying use of the earlier commands. A 100-step minimization is first performed, followed by running 1000 steps of dynamics, with a Tcl callback function invoked every 100 steps.

  ```
  minimize 100
  proc my_callback_func {labels values} {
    print $labels
    print $values
  }
  callback my_callback_func
  run 0
  for {set i 0} {$i < 1000} {incr i 100} {
    run norepeat 100
  }
  ```

  Note the use of “run norepeat” to avoid reevaluating each first step when the coordinates have not changed.

- The “startup” command will trigger simulation startup as would the first “run” or “minimize” command, but without any force/energy evaluation.

- Configuration file parameter introspection is supported by invoking a (case-insensitive) parameter keyword with no argument (e.g., “numsteps”) and by the helper commands “isset” and “istrue”. Note that keywords are not parsed until the first “run” command, and before this values are treated as unformatted strings, so for example “eFieldOn” and “eField” may return “yes” and “1 2 3” before the first “run” command, but “1” and “1.0 2.0 3.0” after parsing (“istrue eFieldOn” would return “1” in both cases). Similarly, “isset badparam” will return “0” before parsing but raise an “unknown parameter” error after.

- Between “run” commands the reassignTemp, rescaleTemp, and langevinTemp parameters can be changed to allow simulated annealing protocols within a single config file.

The useGroupPressure, useFlexibleCell, useConstantArea, useConstantRatio, LangevinPiston, LangevinPistonTarget, LangevinPistonPeriod, LangevinPistonDecay, LangevinPistonTemp, SurfaceTensionTarget, BerendsenPressure, BerendsenPressureTarget, BerendsenPressureCompressibility, and BerendsenPressureRelaxationTime parameters may be changed to allow pressure equilibration. The fixedAtoms, constraintScaling, and nonbondedScaling parameters may be changed to preserve macromolecular conformation during minimization and equilibration (fixedAtoms may only be disabled, and requires that fixedAtomsForces is enabled to do this). The consForceScaling parameter may be changed to vary steering forces or to implement a time-varying electric field that affects specific atoms. The eField, eFieldFreq, and eFieldPhase parameters may be changed to implement at time-varying electric field that affects all atoms. The updateGridforceScale parameter may be called to change the scaling factor applied to gridforces potentials. The alchLambda and alchLambda2 parameters may be changed during alchemical free energy runs. The DCDfile may be changed to write binary coordinate trajectory output to separate files. The restartname may be changed to write restart output to separate files.

- The “checkpoint” and “revert” commands (no arguments) allow a scripted simulation to save and restore (in memory) to a single prior state. The “output” and “reinitatoms” commands support multiple saved states using files. Multiple saved states in memory are supported by the commands “checkpointStore”, “checkpointLoad”, “checkpointSwap”, and “checkpointFree”, all of which take a string key as an argument, plus an optional second argument that is either replica index (the checkpoint is stored asynchronously on the target replica) or the keyword “global” (the target replica is computed as a hash of the key).
- The “output” command takes an output file basename and causes .coor, .vel, and .xsc files to be written with that name. Alternatively, “output withforces” and “output onlyforces” will write a .force file either in addition to or instead of the regular files.
- The “reinitatoms” command reinitializes coordinates, velocities, and periodic cell dimensions to those initially read in (random velocities are generated if they were not read from a file). An optional file basename argument (matching that passed to the output command) causes .coor, .vel, and .xsc files to be read, assuming the format indicated by the binaryoutput parameter.
- The “move” command repositions individual atoms, including fixed atoms. Arguments are a 1-based atom ID, “to” or “by”, and a list of three numbers, e.g., “move 1 by {0.4 0.2 -0.1}”. Atoms may not be moved by more than a single patch between “run” commands.
- The “exit” command writes output files and exits cleanly.
- The “abort” command concatenates its arguments into an error message and exits immediately without writing output files.
- The “numPes”, “numNodes”, and “numPhysicalNodes” commands allow performance-tuning parameters to be set based on the parallel execution environment.
- The “reinitvels” command reinitializes velocities to a random distribution based on the given temperature.
- The “rescalevels” command rescales velocities by the given factor.

- The “reloadCharges” command reads new atomic charges from the given file, which should contain one number for each atom, separated by spaces and/or line breaks.
- The “consForceConfig” command takes a list of 0-based atom indices and a list of forces which replace the existing set of constant forces (constantForce must be on).
- The “measure” command allows user-programmed calculations to be executed in order to facilitate automated methods. (For example, to revert or change a parameter.) A number of measure commands are included in the NAMD binary; the module has been designed to make it easy for users to add additional measure commands.
- The “coorfile” command allows NAMD to perform force and energy analysis on trajectory files. “coorfile open dcd `filename`” opens the specified DCD file for reading. “coorfile read” reads the next frame in the opened DCD file, replacing NAMD’s atom coordinates with the coordinates in the frame, and returns 0 if successful or -1 if end-of-file was reached. “coorfile skip” skips past one frame in the DCD file; this is significantly faster than reading coordinates and throwing them away. “coorfile close” closes the file. The “coorfile” command is not available on the Cray T3E.

  Force and energy analysis are especially useful in the context of pair interaction calculations; see Sec. 15.1 for details, as well as the example scripts in Sec. 18.

Please note that while NAMD has traditionally allowed comments to be started by a # appearing anywhere on a line, Tcl only allows comments to appear where a new statement could begin. With Tcl config file parsing enabled (all shipped binaries) both NAMD and Tcl comments are allowed before the first “run” command. At this point only pure Tcl syntax is allowed. In addition, the “;#” idiom for Tcl comments will only work with Tcl enabled. NAMD has also traditionally allowed parameters to be specified as “param=value”. This is supported, but only before the first “run” command. Some examples:

```
# this is my config file                          <- OK
reassignFreq 100 ; # how often to reset velocities  <- only w/ Tcl
reassignTemp 20 # temp to reset velocities to     <- OK before "run"
run 1000                                          <- now Tcl only
reassignTemp 40 ; # temp to reset velocities to   <- ";" is required
```

NAMD has also traditionally allowed parameters to be specified as “param=value” as well as “param value”. This is supported, but only before the first “run” command. For an easy life, use “param value”.

### 2.2.3 Multiple-copy/replica-exchange scripting interface

Multiple-copy (or replica-based) algorithms are supported by the following commands, which utilize two-sided semantics modeled on MPI:

- myReplica
- numReplicas
- replicaBarrier

- replicaSend *data dest*
- replicaRecv *source*
- replicaSendrecv *data dest source*
- replicaAtomSend *dest*
- replicaAtomRecv *source*
- replicaAtomSendrecv *dest source*

The replicaSend/Sendrecv *data* argument may be any string, and hence any Tcl object (e.g., a list) that can be represented as a string. Data received from the *source* replica is returned by replicaRecv/Sendrecv. In order to ensure message ordering, replicaSend/Sendrecv will block until the corresponding remote receive call (except when replicaSend is called from inside replicaEval, as discussed below).

The parameter replicaUniformPatchGrids must be true for atom exchange (replicaAtom...) or remote checkpointing (checkpoint... with a second argument, see below).

The following additional commands utilize one-sided semantics, and should provide a complete feature set for running a simulation with fewer NAMD replica partitions than logical replicas:

- checkpointStore *key* ?*replica* or global?
- checkpointLoad *key* ?*replica* or global?
- checkpointSwap *key* ?*replica* or global?
- checkpointFree *key* ?*replica* or global?
- replicaEval *replica script*
- replicaYield ?*seconds*?
- replicaDcdFile *index*—off ?*filename*?

The *key* can be any string. By default the checkpoint is stored in the memory of the replica the command is called on. If you specify a replica index the checkpoint is stored asynchronously in that replica's memory. If you specify "global" a hash is computed based on the key to select the replica on which to store the checkpoint. You can have checkpoints with the same key stored on multiple replicas at once if you really want to. The checkpoint... commands will not return until the checkpoint operation has completed.

Storing checkpoints is not atomic. If two replicas try to store a checkpoint with the same key on the same replica at the same time you may end up with a mix of the two (and probably duplicate/missing atoms). If one replica tries to load a checkpoint while another replica is storing it the same may happen. You cannot store a checkpoint on a replica until that replica has created its own patch data structures. This can be guaranteed by calling "startup" and "replicaBarrier" before any remote checkpoint calls.

The replicaEval command asynchronously executes its script in the top-level context of the target replica's Tcl interpreter and returns the result or error. This should be general enough to build any kind of work scheduler or shared data structure you need. If you want to call replicaEval repeatedly, e.g., to check if some value has been set, you should call "replicaYield *seconds*" in between, as this will introduce a delay but still enable processing of asynchronous calls from other replicas. Potentially blocking functions such as replicaRecv should not be called from within replicaEval, nor should functions such as run, checkpointLoad/Store, and replicaAtomSend/Recv that would require the simulation of the remote replica to be halted. It is allowed to call replicaSend (but not replicaSendrecv) from within replicaEval, since replicaSend is non-blocking and one-sided (but potentially overtaking) in this context. Rather than polling a remote replica (e.g., for work) via replicaEval, it is more efficient to register a request via replicaEval and then call replicaRecv to wait for notification.

The replicaDcdFile command is similar to the dcdFile command in that it changes the trajectory output file, but the file is actually opened by a different replica partition and may be written to by any other partition that calls replicaDcdFile with the same index but no filename argument. If a filename argument is given, any file currently associated with the index is closed and a new file created, even if the new and old filenames are the same. The new file is created only when the next trajectory frame is written, not during the replicaDcdFile command itself. The caller must ensure that an index is not used before it is associated with a filename, and that each index is in use by only one replica at a time. The keyword "off" will return to writing the local trajectory file set by the dcdFile command.

### 2.2.4 Python scripting interface and features

NAMD may be compiled with an embedded Python interpreter via the config script option --with-python. Both Python 2.x and 3.x versions are supported, with 3.x the default if found. The config script option --python-prefix can be used to specify the path to the python installation to be used. The default embedded Tcl interpreter is also required to enable Python scripting. Released NAMD binaries do not support Python scripting at this time due to portability issues with the extensive Python library.

Python scripting is accessed via the Tcl "python" command, which functions in either expression mode or script mode. When passed a single-line string, the Python interpreter will evaluate the expression in the string and return the result. Python containers (tuples, lists, etc.) are converted to Tcl lists and all other values are converted to their string representation, which is typically compatible with Tcl. For example, "[python ( 1 + 1, 'abc' + '123' )]" evaluates to the Tcl list "2 abc123".

When the python command is passed a single multi-line string (typically enclosed in braces), the Python interpreter will execute the code in the string and return nothing. Because of Python's indentation-sensitive syntax the enclosed code can not be indented.

Calls to Tcl from Python code are supported by the tcl module functions tcl.call(), which takes the Tcl command name and its arguments as separate arguments and performs limited container and string conversions as described above, and tcl.eval(), which passes a single string unmodified to the Tcl interpreter. Both functions return the result as a string, so numerical results must be explicitly cast to float or int as appropriate.

NAMD simulation parameters and commands are wrapped for convenience by the "namd" object. Any NAMD simulation parameter may be set by assigning to the corresponding case-insensitive attribute of the namd object, e.g., "namd.timestep = 1.0", and similarly read (as a string) by access, e.g., "ts = float(namd.TimeStep)". Assignment corresponds exactly to normal config file parsing, i.e., "timestep 1.0", and hence multiple assignment will generate an error just as would repeated parameters. For convenience, multiple parameters may be set at once by passing them as keyword arguments, e.g., "namd(langevin=True, langevinDamping=5., langevin-Temp=100.)". NAMD (and other) commands in the Tcl interpreter may be called as a method of the namd object, e.g., "namd.run(1000)" and "namd.output('myfile')".

The NAMD `1-4scaling` parameter is incompatible with Python syntax, and so has been deprecated and replaced by `oneFourScaling`. The original parameter name is still accessible several other ways, e.g., "namd.param('1-4scaling',1.0)", "tcl.call('1-4scaling',1.0)", or "tcl.eval('1-4scaling 1.0')".

The namd object is available as the namd *module*, which can be accessed from user-written Python modules by the standard import statement (i.e, "import namd").

The following example illustrates various aspects of the Python scripting interface:

```
set a 1
cutoff 12.0
python {
# do not indent
namd.pairlistDist = float(namd.Cutoff) + float(tcl.eval("set a")) # cast strings to float
b = 2
namd(switching=True, switchdist = float(namd.cutoff) - b) # case insensitive
}
set c [python $a + b]
```

### 2.2.5 Required NAMD configuration parameters

The following parameters are *required* for every NAMD simulation:

- `numsteps` (page 89),
- `coordinates` (page 26),
- `structure` (page 26),
- `parameters` (page 26),
- `exclude` (page 59),
- `outputname` (page 27),
- one of the following three:
  - `temperature` (page 89),
  - `velocities` (page 27),
  - `binvelocities` (page 27).

These required parameters specify the most basic properties of the simulation. In addition, it is highly recommended that `pairlistdist` be specified with a value at least one greater than `cutoff`.

# 3 Input and Output Files

NAMD was developed to be compatible with existing molecular dynamics packages, especially the packages X-PLOR [12] and CHARMM [11]. To achieve this compatibility, the set of input files which NAMD uses to define a molecular system are identical to the input files used by X-PLOR and CHARMM. Thus it is trivial to move an existing simulation from X-PLOR or CHARMM to NAMD. A description of these molecular system definition files is given in Section 3.1.

In addition, the output file formats used by NAMD were chosen to be compatible with X-PLOR and CHARMM. In this way the output from NAMD can be analyzed using X-PLOR, CHARMM, or a variety of the other tools that have been developed for the existing output file formats. Descriptions of the output files formats are also given in Section 3.1.

## 3.1 File formats

### 3.1.1 PDB files

The PDB (Protein Data Bank) format is used for coordinate, velocity, force, or other data being read or written by NAMD. This is the standard format for coordinate data for most other molecular dynamics programs as well, including X-PLOR and CHARMM. A full description of this file format can be obtained from the PDB web site at `http://www.rcsb.org/pdb/`. Positions in PDB files are stored in Å. Velocities in PDB files are stored in Å/ps and may be divided by PDBVELFACTOR=20.45482706 to convert to the NAMD internal units used in DCD and NAMD binary files. Forces in PDB files are stored in kcal/mol/Å. NAMD binary files (below) should be preferred to PDB files in most cases due to their higher precision.

### 3.1.2 X-PLOR format PSF files

NAMD uses the same protein structure files that X-PLOR does. These files may be generated with psfgen, VMD, X-PLOR, or CHARMM. CHARMM can generate an X-PLOR format PSF file with the command "`write psf card xplor`".

### 3.1.3 CHARMM19, CHARMM22, and CHARMM27 parameter files

NAMD supports CHARMM19, CHARMM22, and CHARMM27 parameter files in both X-PLOR and CHARMM formats. (X-PLOR format is the default, CHARMM format parameter files may be used given the parameter "`paraTypeCharmm on`".) For a full description of the format of commands used in these files, see the X-PLOR and CHARMM User's Manual [12].

### 3.1.4 DCD trajectory files

NAMD produces DCD trajectory files in the same format as X-PLOR and CHARMM. The DCD files are single precision binary FORTRAN files, so are transportable between computer architectures. The file readers in NAMD and VMD can detect and adapt to the endianness of the machine on which the DCD file was written, and the utility program `flipdcd` is also provided to reformat these files if needed. The exact format of these files is very ugly but supported by a wide range of analysis and display programs. The timestep is stored in the DCD file in NAMD internal units and must be multiplied by TIMEFACTOR=48.88821 to convert to fs. Positions in DCD files are stored in Å. Velocities in DCD files are stored in NAMD internal units and must be multiplied by PDBVELFACTOR=20.45482706 to convert to Å/ps. Forces in DCD files are stored in kcal/mol/Å.

### 3.1.5 NAMD binary files

NAMD uses a trivial double-precision binary file format for coordinates, velocities, and forces. Due to its high precision this is the default output and restart format. VMD refers to these files as the "namdbin" format. The file consists of the atom count as a 32-bit integer followed by all three position or velocity components for each atom as 64-bit double-precision floating point, i.e., NXYZXYZXYZXYZ... where N is a 4-byte int and X, Y, and Z are 8-byte doubles. If the number of atoms the file contains is known then the atom count can be used to determine endianness. The file readers in NAMD and VMD can detect and adapt to the endianness of the machine on which the binary file was written, and the utility program `flipbinpdb` is also provided to reformat these files if needed. Positions in NAMD binary files are stored in Å. Velocities in NAMD binary files are stored in NAMD internal units and must be multiplied by PDBVELFACTOR=20.45482706 to convert to Å/ps. Forces in NAMD binary files are stored in kcal/mol/Å.

## 3.2 NAMD configuration parameters

### 3.2.1 Input files

- `coordinates` < coordinate PDB file >
  **Acceptable Values:** UNIX filename
  **Description:** The PDB file containing initial position coordinate data. Note that path names can be either absolute or relative. Only one value may be specified.

- `structure` < PSF file >
  **Acceptable Values:** UNIX filename
  **Description:** The X-PLOR format PSF file describing the molecular system to be simulated. Only one value may be specified.

- `parameters` < parameter file >
  **Acceptable Values:** UNIX filename
  **Description:** A CHARMM19, CHARMM22, or CHARMM27 parameter file that defines all or part of the parameters necessary for the molecular system to be simulated. At least one parameter file must be specified for each simulation. Multiple definitions (but only one file per definition) are allowed for systems that require more than one parameter file. The files will be read in the order that they appear in the configuration file. If duplicate parameters are read, a warning message is printed and the last parameter value read is used. Thus, the order that files are read can be important in cases where duplicate values appear in separate files.

- `paraTypeXplor` < Is the parameter file in X-PLOR format? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `on`
  **Description:** Specifies whether or not the parameter file(s) are in X-PLOR format. X-PLOR format is the default for parameter files! Caveat: The PSF file should be also constructed with X-PLOR in case of an X-PLOR parameter file because X-PLOR stores information about the multiplicity of dihedrals in the PSF file. See the X-PLOR manual for details.

- `paraTypeCharmm` < Is the parameter file in CHARMM format? >
  **Acceptable Values:** `on` or `off`

**Default Value:** `off`
**Description:** Specifies whether or not the parameter file(s) are in CHARMM format. X-PLOR format is the default for parameter files! Caveat: The information about multiplicity of dihedrals will be obtained directly from the parameter file, and the full multiplicity will be used (same behavior as in CHARMM). If the PSF file originates from X-PLOR, consecutive multiple entries for the same dihedral (indicating the dihedral multiplicity for X-PLOR) will be ignored.

- `velocities` < velocity PDB file >
  **Acceptable Values:** UNIX filename
  **Description:** The PDB file containing the initial velocities for all atoms in the simulation. This is typically a restart file or final velocity file written by NAMD during a previous simulation. Either the `temperature` or the `velocities/binvelocities` option must be defined to determine an initial set of velocities. Both options cannot be used together.

- `binvelocities` < binary velocity file >
  **Acceptable Values:** UNIX filename
  **Description:** The binary file containing initial velocities for all atoms in the simulation. A binary velocity file is created as output from NAMD by activating the `binaryrestart` or `binaryoutput` options. The `binvelocities` option should be used as an alternative to `velocities`. Either the `temperature` or the `velocities/binvelocities` option must be defined to determine an initial set of velocities. Both options cannot be used together.

- `bincoordinates` < binary coordinate restart file >
  **Acceptable Values:** UNIX filename
  **Description:** The binary restart file containing initial position coordinate data. A binary coordinate restart file is created as output from NAMD by activating the `binaryrestart` or `binaryoutput` options. Note that, in the current implementation at least, the `bincoordinates` option must be used in addition to the `coordinates` option, but the positions specified by `coordinates` will then be ignored.

- `cwd` < default directory >
  **Acceptable Values:** UNIX directory name
  **Description:** The default directory for input and output files. If a value is given, all filenames that do not begin with a / are assumed to be in this directory. For example, if `cwd` is set to `/scr`, then a filename of `outfile` would be modified to `/scr/outfile` while a filename of `/tmp/outfile` would remain unchanged. If no value for `cwd` is specified, than all filenames are left unchanged *but are assumed to be relative to the directory which contains the configuration file given on the command line.*

### 3.2.2 Output files

- `outputname` < output file prefix >
  **Acceptable Values:** UNIX filename prefix
  **Description:** At the end of every simulation, NAMD writes two files, one containing the final coordinates and another containing the final velocities of all atoms in the simulation. This option specifies the file prefix for these two files as well as the default prefix for trajectory and restart files. The position coordinates will be saved to a file named as this prefix with `.coor`

appended. The velocities will be saved to a file named as this prefix with `.vel` appended. For example, if the prefix specified using this option was `/tmp/output`, then the two files would be `/tmp/output.coor` and `/tmp/output.vel`.

- `binaryoutput` < use binary output files? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `yes`
  **Description:** Enables the use of binary output files. If this option is not set to `no`, then the final output files will be written in binary rather than PDB format. Binary files preserve more accuracy between NAMD restarts than ASCII PDB files, but the binary files are not guaranteed to be transportable between computer architectures. (The atom count record is used to detect wrong-endian files, which works for most atom counts. The utility program `flipbinpdb` is provided to reformat these files if necessary.)

- `restartname` < restart files prefix >
  **Acceptable Values:** UNIX filename prefix
  **Default Value:** *outputname*`.restart`
  **Description:** The prefix to use for restart filenames. NAMD produces restart files that store the current positions and velocities of all atoms at some step of the simulation. This option specifies the prefix to use for restart files in the same way that `outputname` specifies a filename prefix for the final positions and velocities. If `restartname` is defined then the parameter `restartfreq` must also be defined.

- `restartfreq` < frequency of restart file generation >
  **Acceptable Values:** positive integer
  **Description:** The number of timesteps between the generation of restart files.

- `restartsave` < use timestep in restart filenames? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Appends the current timestep to the restart filename prefix, producing a sequence of restart files rather than only the last version written.

- `binaryrestart` < use binary restart files? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `yes`
  **Description:** Enables the use of binary restart files. If this option is not set to `no`, then the restart files will be written in binary rather than PDB format. Binary files preserve more accuracy between NAMD restarts than ASCII PDB files, but the binary files are not guaranteed to be transportable between computer architectures. (The atom count record is used to detect wrong-endian files, which works for most atom counts. The utility program `flipbinpdb` is provided to reformat these files if necessary.)

- `DCDfile` < coordinate trajectory output file >
  **Acceptable Values:** UNIX filename
  **Default Value:** *outputname*`.dcd`
  **Description:** The binary DCD position coordinate trajectory filename. This file stores the trajectory of all atom position coordinates using the same format (binary DCD) as X-PLOR. If `DCDfile` is defined, then `DCDfreq` must also be defined.

- `DCDfreq` < timesteps between writing coordinates to trajectory file >
  **Acceptable Values:** positive integer
  **Description:** The number of timesteps between the writing of position coordinates to the trajectory file. The initial positions will not be included in the trajectory file. Positions in DCD files are stored in Å.

- `DCDUnitCell` < write unit cell data to dcd file? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `yes` if periodic cell
  **Description:** If this option is set to `yes`, then DCD files will contain unit cell information in the style of Charmm DCD files. By default this option is enabled if the simulation cell is periodic in all three dimensions and disabled otherwise.

- `velDCDfile` < velocity trajectory output file >
  **Acceptable Values:** UNIX filename
  **Default Value:** *outputname*`.veldcd`
  **Description:** The binary DCD velocity trajectory filename. This file stores the trajectory of all atom velocities using the same format (binary DCD) as X-PLOR. If `velDCDfile` is defined, then `velDCDfreq` must also be defined.

- `velDCDfreq` < timesteps between writing velocities to trajectory file >
  **Acceptable Values:** positive integer
  **Description:** The number of timesteps between the writing of velocities to the trajectory file. The initial velocities will not be included in the trajectory file. Velocities in DCD files are stored in NAMD internal units and must be multiplied by PDBVELFACTOR=20.45482706 to convert to Å/ps.

- `forceDCDfile` < force trajectory output file >
  **Acceptable Values:** UNIX filename
  **Default Value:** *outputname*`.forcedcd`
  **Description:** The binary DCD force trajectory filename. This file stores the trajectory of all atom forces using the same format (binary DCD) as X-PLOR. If `forceDCDfile` is defined, then `forceDCDfreq` must also be defined.

- `forceDCDfreq` < timesteps between writing force to trajectory file >
  **Acceptable Values:** positive integer
  **Description:** The number of timesteps between the writing of forces to the trajectory file. The initial forces will not be included in the trajectory file. Forces in DCD files are stored in kcal/mol/Å. *In the current implementation only those forces that are evaluated during the timestep that a frame is written are included in that frame. This is different from the behavior of TclForces and is likely to change based on user feedback. For this reason it is strongly recommended that forceDCDfreq be a multiple of fullElectFrequency.*

- `crashOutputFlag` < bitmask controling the output of atom positions and velocities when crashing (only used in the GPU-resident mode) >
  **Acceptable Values:** positive integer
  **Default Value:** 1
  **Description:** The value of `crashOutputFlag` is used as a bitmask to determine if CSV

files are saved with atom positions and velocities when crashing. Setting it to 1 will make NAMD write the CSV files when atoms are moving too fast.

- `crashFile` < The CSV filename prefix for saving atom positions and velocities when NAMD crashes >
  **Acceptable Values:** UNIX filename
  **Default Value:** *outputname*`.crash.csv`
  **Description:** The CSV filename for saving atom positions and velocities when NAMD crashes. The CSV file contains 7 columns containing the atom IDs in the PDB indexing, positions and velocities.

### 3.2.3 Standard output

NAMD logs a variety of summary information to standard output. The standard units used by NAMD are Angstroms for length, kcal/mol for energy, Kelvin for temperature, and bar for pressure. Wallclock or CPU times are given in seconds unless otherwise noted.

BOUNDARY energy is from spherical boundary conditions and harmonic restraints, while MISC energy is from external electric fields and various steering forces. TOTAL is the sum of the various potential energies, and the KINETIC energy. TOTAL2 uses a slightly different kinetic energy that is better conserved during equilibration in a constant energy ensemble. Although TOTAL2 is calculated internally for producing TOTAL3, its value is no longer output during simulation. TOTAL3 is another variation with much smaller short-time fluctuations that is also adjusted to have the same running average as TOTAL2. Defects in constant energy simulations are much easier to spot in TOTAL3 than in TOTAL or TOTAL2.

PRESSURE is the pressure calculated based on individual atoms, while GPRESSURE incorporates hydrogen atoms into the heavier atoms to which they are bonded, producing smaller fluctuations. The TEMPAVG, PRESSAVG, and GPRESSAVG are the average of temperature and pressure values since the previous ENERGY output; for the first step in the simulation they will be identical to TEMP, PRESSURE, and GPRESSURE.

Performance for NAMD's new GPU-resident mode is improved by avoiding calculation of energy and virial reductions except when needed. This means that the averages are no longer calculated over all values since the previous ENERGY output. Instead, a moving average is calculated over the previously output values, using a fixed window size defined by a config file parameter. GPU-resident mode replaces TOTAL3 with TOTALAVG, a moving average over the previous TOTAL energy values.

- `outputEnergies` < timesteps between energy output >
  **Acceptable Values:** positive integer
  **Default Value:** 1
  **Description:** The number of timesteps between each energy output of NAMD. This value specifies how often NAMD should output the current energy values to **stdout** (which can be redirected to a file). By default, this is done every step. For long simulations, the amount of output generated by NAMD can be greatly reduced by outputting the energies only occasionally. For GPU-resident mode, the default value is raised to 100, although performance might be improved by setting it higher.

- `movingAverageWindowSize` < fixed window size for calculating moving averages >
  **Acceptable Values:** positive integer

**Default Value:** 20
**Description:** In GPU-resident mode, the energy, temperature, and pressure averages are calculated as moving averages over the previously outputted values. Changing the window size does not affect performance.

- `computeEnergies` < timesteps between energy evaluation >
  **Acceptable Values:** positive integer
  **Default Value:** outputEnergies
  **Description:** The number of timesteps between each energy evaluation of NAMD. This value specifies how often NAMD should compute the current energy values. The default value is to keep the same as `outputEnergies`. Other operations requiring the energies, involving `outputEnergies`, `alchOutFreq` and the period of calling a `callback` TCL procedure, should be a multiple of this value. In the GPU build of NAMD, increasing the value of this option can improve the performance since the evaluation of energies requires double-precision mathematical operations.

- `outputEnergiesPrecision` < energy output precision >
  **Acceptable Values:** positive integer
  **Default Value:** 4
  **Description:** The number of decimal digits used when printing NAMD energies. Increasing the output precision will skew the aligned 80-column format that NAMD has traditionally provided. To keep energy output aligned, increase the terminal width by 5 columns for each additional decimal digit beyond 4.

- `mergeCrossterms` < add crossterm energy to dihedral? >
  **Acceptable Values:** yes or no
  **Default Value:** yes
  **Description:** If crossterm (or CMAP) terms are present in the potential, the energy is added to the dihedral energy to avoid altering the energy output format. Disable this feature to add a separate "CROSS" field to the output.

- `outputMomenta` < timesteps between momentum output >
  **Acceptable Values:** nonnegative integer
  **Default Value:** 0
  **Description:** The number of timesteps between each momentum output of NAMD. If specified and nonzero, linear and angular momenta will be output to **stdout**.

- `outputPressure` < timesteps between pressure output >
  **Acceptable Values:** nonnegative integer
  **Default Value:** 0
  **Description:** The number of timesteps between each pressure output of NAMD. If specified and nonzero, atomic and group pressure tensors will be output to **stdout**.

- `outputTiming` < timesteps between timing output >
  **Acceptable Values:** nonnegative integer
  **Default Value:** the greater of `firstLdbStep` or 10× `outputEnergies`
  **Description:** The number of timesteps between each timing output of NAMD. If nonzero, CPU and wallclock times and memory usage will be output to **stdout**. These data are from node 0 only; CPU times and memory usage for other nodes may vary.

- `outputPerformance` < Print performance statistics? >
  **Acceptable Values:** "on" or "off"
  **Default Value:** on
  **Description:** When enabled, every "TIMING:" line is immediately followed by a "PERFORMANCE:" line that states the average ns/day performance.

### 3.2.4 Output Selection

For the cases when it is preferable to have trajectory output of a subset of the atoms, selections can be defined and assigned independent output frequencies and output file names.

Atom selections are defined in one of two ways.

The first method is by creating a PDB file where the **temperaturefactor** field (AKA **beta** in **VMD**) is assigned a value greater than zero for each atom in the selection. For example, when the objective is to have a more frequent output trajectory containing no solvent, the following **VMD** commands will produce an appropriate selection PDB.

(*Note: example assumes structure and coordinates are already loaded*)

```
set allatoms [atomselect top "all"]
set soluteatoms [atomselect top "not water and not ions"]
$allatoms set beta 0
$soluteatoms set beta 1
$allatoms writepdb solute.pdb
$soluteatoms writepsf solute.psf
```

The additional PSF file in the prior example is not necessary for **NAMD**, but would be useful in **VMD**, or other subsequent analysis, to examine the trajectory associated with given selection of atoms.

The second method is by creating a binary index file containing the binary formatted atom indices within the selection, preceded by a short version field and the total count of atom indices. The **VMD** indexfile plugin is recommended for this task. See **VMD** tcl code below. Note, **NAMD** supports either endian choice for the file, but it must be consistent within the file.

(*Note: example assumes structure and coordinates are already loaded*)

```
 set soluteatoms [atomselect top "not water and not ions"]

#export via the VMD plugin

 package require indexfile
 indexfile::write_indexfile $soluteatoms solute.idx

#or use custom tcl code

 set myList [$soluteatoms get index]
 set $fh [open "solute.idx" wb]
 set idxversion 1
 set size [$soluteatoms num]
 puts -nonewline $fh [binary format s $idxversion]
 puts -nonewline $fh [binary format i $size]
```

```
foreach num $myList
{
  puts -nonewline $fh [binary format i $num]
}
close $fh
```

Such selections can be used within **NAMD** using the arguments described below. A maximum of 16 such selections is allowed. There are no restrictions regarding atoms belonging to multiple selections. Each selection is identified by a user defined string tag to bind the **frequency** and **input file** and **output file** definitions together for each selection. The tag is an alphanumeric string without spaces, up to 80 characters in length, which identifies to which selection the specified field applies.

- `DcdSelection` < enable user defined output? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Specifies whether or not any user defined selections are being used.

- `DcdSelectionInputfile` < tag > < user selection PDB or IDX input file >
  **Acceptable Values:** UNIX filename
  **Default Value:** *inputfilename*`.pdb or .idx`
  **Description:** The PDB file containing a record for each atom in the system with the beta field set greater than zero for the atoms in the user selection and left zero for the atoms output the user selection. Or the IDX file containing only the integer indices of the atoms in the selection in binary format. If `DcdSelectionInputfile` is defined, then `DcdSelectionfile` and `DcdSelectionfreq` must also be defined.

- `DcdSelectionfile` < tag > < user selection coordinate trajectory output file >
  **Acceptable Values:** UNIX filename
  **Default Value:** *outputname*`.dcd`
  **Description:** The binary DCD position coordinate trajectory filename for the user selection. This file stores the trajectory of all atom position coordinates using the same format (binary DCD) as X-PLOR. If `DcdSelectionfile` is defined, then `DcdSelectionfreq` must also be defined.

- `DcdSelectionfreq` < tag > < timesteps between writing coordinates to user selection trajectory file >
  **Acceptable Values:** positive integer
  **Description:** The number of timesteps between the writing of position coordinates to the trajectory file. The initial positions will not be included in the trajectory file. Positions in DCD files are stored in Å.

In the case of the example above, the following configuration would be valid:

```
dcdSelection yes
dcdSelectionInputfile soluteonly solute.pdb
dcdSelectionfreq soluteonly 100
dcdSelectionfile soluteonly solute.dcd
```

**NOTE:** Using this feature with special memory-optimized NAMD builds requires the binary index file form of selection instead of using a PDB. https://www.ks.uiuc.edu/Research/namd/wiki/index.cgi?NamdMemoryReduction.

See lib/replica/examples/mplexjob0-dcdselect.conf for replica usage.

## 3.3 AMBER file and force field support

AMBER format PARM file and coordinate file can be read by NAMD, which allows one to use AMBER force field to carry out all types of simulations that NAMD has supported. NAMD can read PARM files in either the format used in AMBER 6 or the new format defined in AMBER 7. The output of the simulation (restart file, DCD file, etc.) will still be in traditional format that has been used in NAMD.

- `amber` < use AMBER format force field? >
  **Acceptable Values:** yes or no
  **Default Value:** no
  **Description:** If `amber` is set to on, then `parmfile` must be defined, and `structure` and `parameters` should not be defined.

- `oldParmReader` < use the old AMBER parm file reader? >
  **Acceptable Values:** yes or no
  **Default Value:** no
  **Description:** If `oldParmReader` is set to on, then NAMD will read the AMBER parameter file by the old reader, which does not support the CMAP terms in the PARM7 file. AMBER ff19SB turns on the CMAP terms by default, so you should NOT turn on this option if you are using ff19SB or similar AMBER force fields. If you are using NAMD with the PARM (not PARM7) format file generated by some tools predate approximately 2002, then you need to turn on this option.

- `parmfile` < AMBER format PARM file >
  **Acceptable Values:** UNIX filename
  **Description:** This file contains complete topology and parameter information of the system.

- `ambercoor` < AMBER format coordinate file >
  **Acceptable Values:** UNIX filename
  **Description:** This file contains the coordinates of all the atoms. Note that `coordinates` can also be used for PDB format coordinate file. When `amber` is set to on, either `ambercoor` or `coordinates` must be defined, but not both.

- `readexclusions` < Read exclusions from PARM file? >
  **Acceptable Values:** yes or no
  **Default Value:** yes
  **Description:** PARM file explicitly gives complete exclusion (including 1-4 exclusions) information. When `readexclusions` is set to on, NAMD will read all exclusions from PARM file and will not add any more; alternatively, if `readexclusions` is set to off, NAMD will ignore the exclusions in PARM file and will automatically generate them according to the exclusion policy specified by `exclude`.

- `scnb` < VDW 1-4 scaling factor >
  **Acceptable Values:** decimal $\geq 1.0$
  **Default Value:** 2.0
  **Description:** Same meaning as SCNB in AMBER. Note that in NAMD, 1-4 vdw interactions are DIVIDED by `scnb`, whereas 1-4 electrostatic interactions are MULTIPLIED by `oneFourScaling`. So `oneFourScaling` should be set to the inverse of SCEE value used in AMBER.

Caveat:
1. Polarizable parameters in AMBER are not supported.
2. NAMD does not support the 10-12 potential terms in some old AMBER versions. When non-zero 10-12 parameter is encountered in PARM file, NAMD will terminate.
3. NAMD has several exclusion policy options, defined by `exclude`. The way AMBER dealing with exclusions corresponds to the "scaled1-4" in NAMD. So for simulations using AMBER force field, one would specify "exclude scaled1-4" in the configuration file, and set `oneFourScaling` to the inverse value of SCEE as would be used in AMBER.
4. NAMD does not read periodic box lengths in PARM or coordinate file. They must be explicitly specified in NAMD configuration file.
5. By default, NAMD applies switching functions to the non-bond interactions within the cut-off distance, which helps to improve energy conservation, while AMBER does not use switching functions so it simply truncates the interactions at cutoff. However, if "authentic" AMBER cutoff simulations are desired, the switching functions could be turned off by specifying "switching off" in NAMD configuration file.
6. NAMD and AMBER may have different default values for some parameters (e.g., the tolerance of SHAKE). One should check other sections of this manual for accurate descriptions of the NAMD options.

Following are two examples of the NAMD configuration file to read AMBER force field and carry out simulation. They may help users to select proper NAMD options for AMBER force field. For the convenience of AMBER users, the AMBER 6 sander input files are given in the left for comparison, which would accomplish similar tasks in AMBER.

Example 1: Non-periodic boundary system, cutoff simulation

```
---AMBER----      ---NAMD---

 TITLE
 &cntrl
  ntb=0, igb=2,   # non-periodic, use cutoff for non-bond
  nstlim=1000,    numsteps       1000  # Num of total steps
  ntpr=50,        outputEnergies 50  # Energy output frequency
  ntwr=50,        restartfreq    50  # Restart file frequency
  ntwx=100,       DCDfreq        100  # Trajectory file frequency
  dt=0.001,       timestep       1  # in unit of fs (This is default)
  tempi=0.,       temperature    0  # Initial temp for velocity assignment
  cut=10.,        cutoff         10
                  switching      off  # Turn off the switching functions
  scee=1.2,       exclude        scaled1-4
                  oneFourScaling 0.833333  # =1/1.2, default is 1.0
  scnb=2.0        scnb           2  # This is default
 &end
                  amber          on  # Specify this is AMBER force field
                  parmfile       prmtop  # Input PARM file
                  ambercoor      inpcrd  # Input coordinate file
                  outputname     md  # Prefix of output files
```

Example 2: Periodic boundary system, PME, NVE ensemble, using SHAKE algorithm

```
---AMBER----     ---NAMD---

TITLE
&cntrl
 ntc=2, ntf=2,   # SHAKE to the bond between each hydrogen and it mother atom
                 rigidBonds     all
 tol=0.0005,     rigidTolerance 0.0005  # Default is  0.00001
 nstlim=500,     numsteps       500  # Num of total steps
 ntpr=50,        outputEnergies 50  # Energy output frequency
 ntwr=100,       restartfreq    100  # Restart file frequency
 ntwx=100,       DCDfreq        100  # Trajectory file frequency
 dt=0.001,       timestep       1  # in unit of fs (This is default)
 tempi=300.,     temperature    300  # Initial temp for velocity assignment
 cut=9.,         cutoff         9
                 switching      off  # Turn off the switching functions
&end
&ewald           PME            on  # Use PME for electrostatic calculation
                 # Orthogonal periodic box size
 a=62.23,        cellBasisVector1   62.23  0  0
 b=62.23,        cellBasisVector2   0  62.23  0
 c=62.23,        cellBasisVector3   0  0  62.23
 nfft1=64,       PMEGridSizeX   64
 nfft2=64,       PMEGridSizeY   64
 nfft3=64,       PMEGridSizeZ   64
 ischrgd=1,      # NAMD doesn't force neutralization of charge
&end
                 amber          on  # Specify this is AMBER force field
                 parmfile       FILENAME  # Input PARM file
                 ambercoor      FILENAME  # Input coordinate file
                 outputname     PREFIX  # Prefix of output files
                 exclude        scaled1-4
                 oneFourScaling 0.833333  # =1/1.2, default is 1.0
```

### 3.4 GROMACS file support

NAMD has the ability to load GROMACS ASCII topology (.top) and coordinate (.gro) files, which allows you to run most GROMACS simulations in NAMD. All simulation output will still be in the traditional NAMD formats.

- `gromacs` < use GROMACS format force field? >
  **Acceptable Values:** on or off
  **Default Value:** off
  **Description:** If `gromacs` is set to on, then `grotopfile` must be defined, and `structure` and `parameters` should not be defined.

- `grotopfile` < GROMACS format topology/parameter file >
  **Acceptable Values:** UNIX filename
  **Description:** This file contains complete topology and parameter information of the system.

- `grocoorfile` < GROMACS format coordinate file >
  **Acceptable Values:** UNIX filename
  **Description:** This file contains the coordinates of all the atoms. Note that `coordinates` can also be used for PDB format coordinate file. When `gromacs` is set to `on`, either `grocoorfile` or `coordinates` must be defined, but not both.

However, NAMD does not have support for many GROMACS-specific options:

- Dummies (fake atoms with positions generated from the positions of real atoms) are not supported.
- The GROMACS `pairs` section, where explicit 1–4 parameters are given between pairs of atoms, is not supported, since NAMD calculates its 1–4 interactions exclusively by type.
- Similarly, `exclusions` are not supported. The biggest problem here is that GROMACS RB dihedrals are supposed to imply exclusions, but NAMD does not support this.
- Constraints, restraints, and `settles` are not implemented in NAMD.
- In some cases, it may not work to override some but not all of the parameters for a bond, atom, etc. In this case, NAMD will generate an error and stop. The parser will sometimes not tolerate correct GROMACS files or fail to detect errors in badly formatted files.
- NAMD does not support all the types of bond potentials that exist in GROMACS, but approximates them with harmonic or sinusoidal potentials.
- NAMD does not read periodic box lengths in the coordinate file. They must be explicitly specified in the NAMD configuration file.

# 4 Creating PSF Structure Files

The `psfgen` structure building tool consists of a portable library of structure and file manipulation routines with a Tcl interface. Current capabilities include

- reading CHARMM topology files
- reading psf files in X-PLOR/NAMD format
- extracting sequence data from single segment PDB files
- generating a full molecular structure from sequence data
- applying patches to modify or link different segments
- writing NAMD and VMD compatible PSF structure files
- extracting coordinate data from PDB files
- constructing (guessing) missing atomic coordinates
- deleting selected atoms from the structure
- writing NAMD and VMD compatible PDB coordinate files

We are currently refining the interface of `psfgen` and adding features to create a complete molecular building solution. We welcome your feedback on this new tool.

## 4.1 New commands and Functionalities

The version 2.0 of `psfgen` was extensively modified and improved to meet the current standards in the size of the structures, and the modern versions of additive CHARMM force field, and polarizable DRUDE force field (`http://mackerell.umaryland.edu/charmm_ff.shtml`).

New functionalities include:

- hydrogen mass repartition
- structure preparation for Drude force field
- structure preparation containing colinear lone pairs (halogen atoms in the latest additive CHARMM force field version)
- psfgen log file to store all the information printed to the console

To use the Drude force field, one only needs to load the Drude topology files and prepare the structures as per usual. Most commands are available for both lone pairs in the additive and polarizable force fields, although some operations are not yet available for Drude particles. Atom modification operations, e.g., psfset, and queries with the segment command on the drude particles, are not implemented. We advise the user to use VMD to assign beta and occupancy values during the structure preparation. writemol and readmol commands are not compatible with structure preparation for Drude Force field.

The new psfgen log file allows the user to save all the information regularly printed out during a psfgen execution script to a file. It is possible to open and close multiple log files in a psfgen script, but only one file is active at any given moment. An example of an application of multiple log files is to save the information of the loading process of the topology files to one log file and the rest of information of structure preparation to another file, as demonstrated below:

```
psfgen_logfile "load_topoplogy.log"

topology top_all22_prot.rtf
topology top_all36_carb.rtf
topology top_all36_lipid.rtf
topology top_all36_prot.rtf
topology top_all36_cgenff.rtf
topology toppar_water_ions.str

psfgen_logfile close

psfgen_logfile "structure_preparation.log"

segment BPTI {
 pdb output/6PTI_protein.pdb
}

patch DISU BPTI:5 BPTI:55
patch DISU BPTI:14 BPTI:38
patch DISU BPTI:30 BPTI:51

pdbalias atom ILE CD1 CD
coordpdb output/6PTI_protein.pdb BPTI

pdbalias residue HOH TIP3
segment SOLV {
 auto none
 pdb output/6PTI_water.pdb
}

pdbalias atom HOH O OH2
coordpdb output/6PTI_water.pdb SOLV

guesscoord

writepsf output/bpti.psf
writepdb output/bpti.pdb

psfgen_logfile close
```

## 4.2 Ordinary Usage

psfgen is currently distributed in two forms. One form is as a standalone program implemented as a Tcl interpreter which reads commands from standard output. You may use loops, variables, etc. as you would in a VMD or NAMD script. You may use psfgen interactively, but we expect it to be run most often with a script file redirected to standard input. The second form is as a Tcl package which can be imported into any Tcl application, including VMD. All the commands available to the standalone version of psfgen are available to the Tcl package; using psfgen within VMD lets you harness VMD's powerful atom selection capability, as well as instantly view the result of your structure building scripts. Examples of using psfgen both with and without VMD are provided in this document.

Generating PSF and PDB files for use with NAMD will typically consist of the following steps:

1. Preparing separate PDB files containing individual segments of protein, solvent, etc. before running psfgen.
2. Reading in the appropriate topology definition files and aliasing residue and atom names found in the PDB file to those found in the topology files. This will generally include selecting a default protonation state for histidine residues.
3. Generating the default structure using segment and pdb commands.
4. Applying additional patches to the structure.
5. Reading coordinates from the PDB files.
6. Deleting unwanted atoms, such as overlapping water molecules.
7. Guessing missing coordinates of hydrogens and other atoms.
8. Writing PSF and PDB files for use in NAMD.

### 4.2.1 Preparing separate PDB files

Many PDB files in the PDB databank contain multiple chains, corresponding to protein subunits, water, and other miscellaneous groups. Protein subunits are often identified by their chain ID in the PDB file. In psfgen, each of these groups must be assigned to their own *segment*. This applies most strictly in the case of protein chains, each of which must be assigned to its own segment so that N-terminal and C-terminal patches can be applied. You are free to group water molecules into whatever segments you choose.

Chains can be split up into their own PDB files using your favorite text editor and/or Unix shell commands, as illustrated in the BPTI example below. If you are using VMD you can also use atom selections to write pieces of the structure to separate files:

```
# Split a file containing protein and water into separate segments.
# Creates files named myfile_water.pdb, myfile_frag0.pdb, myfile_frag1.pdb,...
# Requires VMD.
mol load pdb myfile.pdb
set water [atomselect top water]
$water writepdb myfile_water.pdb
```

```
set protein [atomselect top protein]
set chains [lsort -unique [$protein get pfrag]]
foreach chain $chains {
  set sel [atomselect top "pfrag $chain"]
  $sel writepdb myfile_frag${chain}.pdb
}
```

### 4.2.2 Deleting unwanted atoms

The `delatom` command described below allows you to delete selected atoms from the structure. It's fine to remove atoms from your structure before building the PSF and PDB files, but you should never edit the PSF and PDB files created by `psfgen` by hand as it will probably mess up the internal numbering in the PSF file.

Very often the atoms you want to delete are water molecules that are either too far from the solute, or else outside of the periodic box you are trying to prepare. In either case VMD atom selections can be used to select the waters you want to delete. For example:

```
# Load a pdb and psf file into both psfgen and VMD.
resetpsf
readpsf myfile.psf
coordpdb myfile.pdb
mol load psf myfile.psf pdb myfile.pdb
# Select waters that are more than 10 Angstroms from the protein.
set badwater1 [atomselect top "name OH2 and not within 10 of protein"]
# Alternatively, select waters that are outside our periodic cell.
set badwater2 [atomselect top "name OH2 and (x<-30 or x>30 or y<-30 or>30
                               or z<-30 or z>30)"]
# Delete the residues corresponding to the atoms we selected.
foreach segid [$badwater1 get segid] resid [$badwater1 get resid] {
  delatom $segid $resid
}
# Have psfgen write out the new psf and pdb file (VMD's structure and
# coordinates are unmodified!).
writepsf myfile_chopwater.psf
writepdb myfile_chopwater.pdb
```

## 4.3 BPTI Example

To actually run this demo requires

- the program `psfgen` from any NAMD distribution,
- the CHARMM topology and parameter files `top_all22_prot.inp` and `par_all22_prot.inp` from `http://mackerell.umaryland.edu/charmm_ff.shtml`, and
- the BPTI PDB file `6PTI.pdb` available from the Protein Data Bank at `http://www.pdb.org/` by searching for `6PTI` and downloading the complete structure file in PDB format.

**Building the BPTI structure**

In this demo, we create the files `bpti.psf` and `bpti.pdb` in the output directory which can then be used for a simple NAMD simulation.

```
# File: bpti_example.tcl
# Requirements: topology file top_all22_prot.inp in directory toppar
#               PDB file 6PTI.pdb in current directory

# Create working directory; remove old output files
mkdir -p output
rm -f output/6PTI_protein.pdb output/6PTI_water.pdb

# (1) Split input PDB file into segments}
grep -v '^HETATM' 6PTI.pdb > output/6PTI_protein.pdb
grep 'HOH' 6PTI.pdb > output/6PTI_water.pdb

# (2) Embed the psfgen commands in this script
psfgen << ENDMOL

# (3) Read topology file
topology toppar/top_all22_prot.inp

# (4) Build protein segment
segment BPTI {
 pdb output/6PTI_protein.pdb
}

# (5) Patch protein segment
patch DISU BPTI:5 BPTI:55
patch DISU BPTI:14 BPTI:38
patch DISU BPTI:30 BPTI:51

# (6) Read protein coordinates from PDB file
pdbalias atom ILE CD1 CD    ; # formerly "alias atom ..."
coordpdb output/6PTI_protein.pdb BPTI

# (7) Build water segment
pdbalias residue HOH TIP3   ; # formerly "alias residue ..."
segment SOLV {
 auto none
 pdb output/6PTI_water.pdb
}

# (8) Read water coordinaes from PDB file
pdbalias atom HOH O OH2     ; # formerly "alias atom ..."
coordpdb output/6PTI_water.pdb SOLV
```

```
# (9) Guess missing coordinates
guesscoord

# (10) Write structure and coordinate files
writepsf output/bpti.psf
writepdb output/bpti.pdb

# End of psfgen commands
ENDMOL
```

Step-by-step explanation of the script:

**(1) Split input PDB file into segments.** 6PTI.pdb is the original file from the Protein Data Bank. It contains a single chain of protein and some PO4 and H2O HETATM records. Since each segment must have a separate input file, we remove all non-protein atom records using grep. If there were multiple chains we would have to split the file by hand. Create a second file containing only waters.

**(2) Embed the psfgen commands in this script.** Run the psfgen program, taking everything until "ENDMOL" as input. You may run psfgen interactively as well. Since psfgen is built on a Tcl interpreter, you may use loops, variables, etc., but you must use `$$` for variables when inside a shell script. If you want, run psfgen and enter the following commands manually.

**(3) Read topology file.** Read in the topology definitions for the residues we will create. This must match the parameter file used for the simulation as well. Multiple topology files may be read in since psfgen and NAMD use atom type names rather than numbers in psf files.

**(4) Build protein segment.** Actually build a segment, calling it BPTI and reading the sequence of residues from the stripped pdb file created above. In addition to the pdb command, we could specify residues explicitly. Both angles and dihedrals are generated automatically unless "auto none" is added (which is required to build residues of water). The commands "first" and "last" may be used to change the default patches for the ends of the chain. The structure is built when the closing } is encountered, and some errors regarding the first and last residue are normal.

**(5) Patch protein segment.** Some patch residues (those not used to begin or end a chain) are applied after the segment is built. These contain all angle and dihedral terms explicitly since they were already generated. In this case we apply the patch for a disulfide link three separate times.

**(6) Read protein coordinates from PDB file.** The same file used to generate the sequence is now read to extract coordinates. In the residue ILE, the atom CD is called CD1 in the pdb file, so we use "pdbalias atom" to define the correct name. If the segment names in the pdb file match the name we gave in the segment statement, then we don't need to specify it again; in this case we do specify the segment, so that all atoms in the pdb file must belong to the segment.

**(7) Build water segment.** Build a segment for the crystal waters. The residue type for water depends on the model, so here we alias HOH to TIP3. Because CHARMM uses an additional H-H bond we must disable generation of angles and dihedrals for segments containing water. Then read the pdb file.

**(8) Read water coordinates from PDB file.** Alias the atom type for water oxygen as well and read coordinates from the file to the segment SOLV. Hydrogen doesn't show up in crystal structures so it is missing from this pdb file.

**(9) Guessing missing coordinates.** The tolopogy file contains default internal coordinates which can be used to guess the locations of many atoms, hydrogens in particular. In the output pdb file, the occupancy field of guessed atoms will be set to 0, atoms which are known are set to 1, and atoms which could not be guessed are set to -1. Some atoms are "poorly guessed" if needed bond lengths and angles were missing from the topology file. Similarly, waters with missing hydrogen coordinates are given a default orientation.

**Write structure and coordinate files.** Now that all of the atoms and bonds have been created, we can write out the psf structure file for the system. We also create the matching coordinate pdb file. The psf and pdb files are a matched set with identical atom ordering as needed by NAMD.

### Using generated files in NAMD.

The files bpti.pdb and bpti.psf can now be used with NAMD, but the initial coordinates require minimization first. The following is an example NAMD configuration file for the BPTI example.

```
# NAMD configuration file for BPTI

# molecular system
structure output/bpti.psf

# force field
paratypecharmm on
parameters toppar/par_all22_prot.inp
exclude scaled1-4
1-4scaling 1.0

# approximations
switching on
switchdist 8
cutoff 12
pairlistdist 13.5
margin 0
stepspercycle 20

#integrator
timestep 1.0
```

```
#output
outputenergies 10
outputtiming 100
binaryoutput no

# molecular system
coordinates output/bpti.pdb

#output
outputname output/bpti
dcdfreq 1000

#protocol
temperature 0
reassignFreq 1000
reassignTemp 25
reassignIncr 25
reassignHold 300

#script

minimize 1000

run 20000
```

## 4.4 Building solvent around a protein

The following script illustrates how `psfgen` and VMD can be used together to add water around a protein structure. It assumes you already have a psf and pdb file for your protein, as well as a box of water which is large enough to contain the protein. For more information on how atomselections can be used within VMD scripts, see the VMD User's Guide.

```
proc addwater { psffile pdbfile watpsf watpdb } {
# Create psf/pdb files that contain both our protein as well as
# a box of equilibrated water.  The water box should be large enough
# to easily contain our protein.
resetpsf
readpsf $psffile pdb $pdbfile
readpsf $watpsf pdb $watpdb

# Load the combined structure into VMD
writepsf combine.psf
writepdb combine.pdb
mol load psf combine.psf pdb combine.pdb

# Assume that the segid of the water in watpsf is QQQ
# We want to delete waters outside of a box ten Angstroms
# bigger than the extent of the protein.
set protein [atomselect top "not segid QQQ"]
set minmax [measure minmax $protein]
foreach {min max} $minmax { break }
foreach {xmin ymin zmin} $min { break }
foreach {xmax ymax zmax} $max { break }
    set xmin [expr $xmin - 10]
    set ymin [expr $ymin - 10]
    set zmin [expr $zmin - 10]
    set xmax [expr $xmax + 10]
    set ymax [expr $ymax + 10]
    set zmax [expr $zmax + 10]

# Center the water on the protein.  Also update the coordinates held
# by psfgen.
set wat [atomselect top "segid QQQ"]
$wat moveby [vecsub [measure center $protein] [measure center $wat]]
foreach atom [$wat get {segid resid name x y z}] {
foreach {segid resid name x y z} $atom { break }
coord $segid $resid $name [list $x $y $z]
}

# Select waters that we don't want in the final structure.
set outsidebox [atomselect top "segid QQQ and (x <= $xmin or y <= $ymin \
or z <= $zmin or x >= $xmax or y >= $ymax or z >= $xmax)"]
set overlap [atomselect top "segid QQQ and within 2.4 of (not segid QQQ)"]

# Get a list of all the residues that are in the two selections, and delete
# those residues from the structure.
set reslist [concat [$outsidebox get resid] [$overlap get resid]]
set reslist [lsort -unique -integer $reslist]

foreach resid $reslist {
delatom QQQ $resid
}

# That should do it - write out the new psf and pdb file.
writepsf solvate.psf
writepdb solvate.pdb

# Delete the combined water/protein molecule and load the system that
# has excess water removed.
mol delete top
mol load psf solvate.psf pdb solvate.pdb

# Return the size of the water box
```

```
return [list [list $xmin $ymin $zmin] [list $xmax $ymax $zmax]]
}
```

### 4.5 New Commands in the version 2.0

- `psfgen_logfile` <*file name*> [`close`]
  **Purpose:** Open or close a log file to store all information printed to the console.
  **Arguments:** <*file name*>: Valid file name in the current directory.
  `close`: Close the active log file. The file name should not be included in the closing command. Example above.
  **Context:** Any part of the script, context independent. May call multiple times.

- `hmassrepart` [*dowater* <*1 0*>] [*mass* <*target hydrogen mass*>]
  **Purpose:** Partition the mass of heavy atoms into the bonded hydrogen atoms.
  **Arguments:** `dowater`: 1 for true, 0 for false. Partition the water molecules. Default value 0.
  `mass`: Target for the hydrogen atoms' mass. Default value 3.024 amu.
  **Context:** After loading or preparing the structure.

- `vpbonds` [1 0]
  **Purpose:** Print the bonds between the virtual particles (drude particles and lone pairs) and their hosts.
  **Arguments:** 1 for true, 0 for false. Default value 1.
  **Context:** Before writing the psf file. May call multiple times. WARNING: To run simulations containing lone pairs or Drude particles on NAMD 2.13, set vpbonds to 0.

### 4.6 List of Commands

- `topology` [`list`] <*file name*>
  **Purpose:** Read in molecular topology definitions from file.
  **Arguments:** <*file name*>: CHARMM format topology file.
  `list`: Lists all currently specified topology files.
  `residues`: Return a list of the known residue topologies.
  `patches`: Return a list of the known residue patches.
  **Context:** Beginning of script, before segment. May call multiple times.

- `topology alias` <*alternate residue name*> <*existing residue name*>
  **Purpose:** Provide alternate names for residues found in topology file. An alternate name used to generate a residue will be used on output. Compare to "pdbalias residue" below, in which the real name is used on output.
  **Arguments:** <*alternate residue name*>: Desired residue name.
  <*existing residue name*>: Residue name found in topology file.
  **Context:** Before reading sequence with pdb. May call multiple times.

- `pdbalias residue` <*alternate name*> <*real name*>
  **Purpose:** Provide translations from residues found in PDB files to proper residue names read in from topology definition files. Proper names from topology files will be used in generated PSF and PDB files. Compare to "topology alias" above, in which the alias is is used as the residue name in generated files. This command also exists under the deprecated name `alias`.

**Arguments:** <*alternate name*>: Residue name found in PDB file.
<*real name*>: Residue name found in topology file or aliases.
**Context:** Before reading sequence with pdb. May call multiple times.

- `segment` [`segids`] [`resids`] [`residue`] [`first`] [`last`] <*segment ID*> [*resid*] [*atom name*] { <*commands*> }
  **Purpose:** Build a segment of the molecule. A segment is typically a single chain of protein or DNA, with default patches applied to the termini. Segments may also contain pure solvent or lipid. Options [`segids`] [`resids`] [`residue`] [`first`] [`last`] are used to query information about the specified segment.
  **Arguments:** `segids`: Return a list of segids for the molecule in the current context.
  `resids`: Return a list of resids for the given segment in the current context.
  `residue`: Return the residue name of the residue in the given segment with the given resid.
  `atoms`: Return a list of atoms for the given segment with the given resid.
  `coordinates`: Return x, y, z coordinates for the given atom.
  `velocities`: Return x, y, z velocities for the given atom.
  `mass`: Return the mass of the given atom.
  `charge`: Return the charge of the given atom.
  `atomid`: Return the one-based atomid of the given atom. These are only assigned/updated when writing a file. Therefore `writepsf`, `writepdb`, or `writemol` must be called to avoid returning old atomid values or zero.
  `first`: Returns the name of the patch that was applied to the beginning of the specified segment.
  `last`: Returns the name of the patch that was applied to the end of the specified segment.
  <*segment ID*>: Unique name for segment, 1–4 characters.
  <*commands*>: Sequence of commands in Tcl syntax to build the primary structure of the segment, including auto, first, last, residue, pdb, etc.
  **Context:** After topology definitions and residue aliases. May call multiple times. Structure information is generated at the end of every segment command.

- `auto` [`angles`] [`dihedrals`] [`none`]
  **Purpose:** Override default settings from topology file for automatic generation of angles and dihedrals for the current segment.
  **Arguments:** `angles`: Enable generation of angles from bonds.
  `dihedrals`: Enable generation of dihedrals from angles.
  `none`: Disable generation of angles and dihedrals.
  **Context:** Anywhere within segment, does not affect later segments.

- `first` <*patch name*>
  **Purpose:** Override default patch applied to first residue in segment. Default is read from topology file and may be residue-specific.
  **Arguments:** <*patch name*>: Single-target patch residue name or `none`.
  **Context:** Anywhere within segment, does not affect later segments.

- `last` <*patch name*>
  **Purpose:** Override default patch applied to last residue in segment. Default is read from topology file and may be residue-specific.

**Arguments:** <*patch name*>: Single-target patch residue name or `none`.
**Context:** Anywhere within segment, does not affect later segments.

- `residue` <*resid*> <*resname*> [*chain*]
  **Purpose:** Add a single residue to the end of the current segment.
  **Arguments:** <*resid*>: Unique name for residue, 1–5 characters, usually numeric. <*resname*>: Residue type name from topology file. <*chain*>: Single-character chain identifier.
  **Context:** Anywhere within segment.

- `pdb` <*file name*>
  **Purpose:** Extract sequence information from PDB file when building segment. Residue IDs will be preserved, residue names must match entries in the topology file or should be aliased before pdb is called.
  **Arguments:** <*file name*>: PDB file containing known or aliased residues.
  **Context:** Anywhere within segment.

- `mutate` <*resid*> <*resname*>
  **Purpose:** Change the type of a single residue in the current segment.
  **Arguments:** <*resid*>: Unique name for residue, 1–5 characters, usually numeric. <*resname*>: New residue type name from topology file.
  **Context:** Within segment, after target residue has been created.

- `patch` [`list`] <*patch residue name*> <*segid:resid*> [*...*]
  **Purpose:** Apply a patch to one or more residues. Patches make small modifications to the structure of residues such as converting one to a terminus, changing the protonation state, or creating disulphide bonds between a pair of residues.
  **Arguments:** `list`: Lists all patches applied explicitey using the command 'patch'.
  `listall`: Lists all currently applied patches including default patches.
  <*patch residue name*>: Name of patch residue from topology definition file.
  <*segid:resid*>: List of segment and residue pairs to which patch should be applied.
  **Context:** After one or more segments have been built.

- `regenerate` [`angles`] [`dihedrals`]
  **Purpose:** Remove all angles and/or dihedrals and completely regenerate them using the segment automatic generation algorithms. This is only needed if patches were applied that do not correct angles and bonds. Segment and file defaults are ignored, and angles/dihedrals for the entire molecule are regenerated from scratch.
  **Arguments:** `angles`: Enable generation of angles from bonds.
  `dihedrals`: Enable generation of dihedrals from angles.
  **Context:** After one or more segments have been built.

- `regenerate` [`resids`]
  **Purpose:** Remove insertion codes and minimally modify resids to retain uniqueness. No modifications will be made in segments that have monotonically increasing resids and do not contain insertion codes. Within a segment, no modifications will be made to residues preceeding the first non-increasing resid or residue with an insertion code.
  **Arguments:** `resids`: Enable regeneration of resids to remove insertion codes.
  **Context:** After one or more segments have been built.

- `multiply` <*factor*> <*segid*[*:resid*[*:atomname*]]> [*...*]
  **Purpose:** Create multiple images of a set of atoms for use in locally enhanced sampling. The beta column of the output pdb file is set to 1...<*factor*> for each image. Multiple copies of bonds, angles, etc. are created. Atom, residue or segment names are not altered; images are distinguished only by beta value. This is not a normal molecular structure and may confuse other tools.
  **Arguments:** <*factor*>:
  <*segid:resid:atomname*>: segment, residue, or atom to be multiplied. If :resid is omitted the entire segment is multiplied; if :atomname is omitted the entire residue is multiplied. May be repeated as many times as necessary to include all atoms.
  **Context:** After one or more segments have been built, all patches applied, and coordinates guessed. The effects of this command may confuse other commands.

- `delatom` <*segid*> [*resid*] [*atomname*]
  **Purpose:** Delete one or more atoms. If only `segid` is specified, all atoms from that segment will be removed from the structure. If both `segid` and `resid` are specified, all atoms from just that residue will be removed. If `segid`, `resid`, and `atomname` are all specified, just a single atom will be removed.
  **Arguments:** <*segid*>: Segment ID of target atom.
  <*resid*>: Residue ID of target atom (optional).
  <*atomname*>: Name of target atom (optional).
  **Context:** After one or more segments have been built.

- `resetpsf`
  **Purpose:** Delete all segments in the structure. The topology definitions and aliases are left intact. If you want to clear the topology and aliases as well, use `psfcontext reset` instead.
  **Arguments:**
  **Context:** After one or more segments have been built.

- `psfcontext` [*context*] [`new`] [`delete`]
  **Purpose:** Switches between complete contexts, including structure, topology definitions, and aliases. If no arguments are provided, the current context is returned. If <*context*> or `new` is specified, a new context is entered and the old context is returned. If `delete` is also specified, the old context is destroyed and "deleted <*old context*>" is returned. An error is returned if the specified context does not exist or if `delete` was specified and the current context would still be in use. *It may be possible to write robust, error-tolerant code with this interface, but it would not be easy. Please employ the following revised* `psfcontext` *usage instead.*
  **Arguments:** <*context*>: Context ID returned by psfcontext.
  **Context:** At any time.

- `psfcontext mixedcase`
  **Purpose:** Make context case sensitive by preserving case of all segment, residue, atom, and patch names on input.
  **Arguments:**
  **Context:** Before reading files requiring case sensitive behavior, normally as the first command.

- `psfcontext allcaps`
  **Purpose:** Make context case insensitive by converting all segment, residue, atom, and patch

names to upper case characters on input. This is the default behavior and should match the behavior of versions prior to 1.5.0.
**Arguments:**
**Context:** Before reading files requiring case insensitive behavior, not needed in normal use.

- `psfcontext reset`
  **Purpose:** Clears the structure, topology definitions, and aliases, creating clean environment just like a new context.
  **Arguments:**
  **Context:** At any time.

- `psfcontext create`
  **Purpose:** Creates a new context and returns its ID, but does not switch to it. This is different from `psfcontext new` above, which switches to the newly created context and returns the current context's ID.
  **Arguments:**
  **Context:** At any time.

- `psfcontext delete` <*context*>
  **Purpose:** Deletes the specified context. An error is returned if the specified context does not exist or would still be in use. This is different from `psfcontext` <***context***> `delete` above, which switches to the specified context and deletes the current one.
  **Arguments:** <*context*>: Context ID returned by psfcontext.
  **Context:** At any time.

- `psfcontext eval` <*context*> { <*commands*> }
  **Purpose:** Evaluates <*commands*> in the specified context, returning to the current context on exit. This should be totally robust, returning to the orignal context in case of errors and preventing its deletion when nested.
  **Arguments:** <*context*>: Context ID returned by `psfcontext create`.
  <*commands*>: Script to be executed in the specified context.
  **Context:** At any time.

- `psfcontext stats`
  **Purpose:** Returns the total numbers of contexts that have been created and destroyed. This is useful for checking if a script is leaking contexts.
  **Arguments:**
  **Context:** At any time.

- `writepsf` [`charmm`] [`x-plor`] [`cmap`] [`nocmap`] [`nopatches`] <*file name*>
  **Purpose:** Write out structure information as PSF file. A simplified session log is listed in the REMARKS section of the PSF file.
  **Arguments:** `charmm`: Use CHARMM format (numbers for atom types).
  `x-plor`: Use X-PLOR format (names for atom types), the default format required by NAMD.
  `cmap`: Write cross-term entries to PSF file if present, the default.
  `nocmap`: Do not write cross-term entries to PSF file, even if present.
  `nopatches`: Do not write list of applied patches to PSF file header.
  <*file name*>: PSF file to be generated.
  **Context:** After all segments have been built and patched.

- `readpsf` <*file name*> [`pdb`] [*pdb file name*] [`namdbin`] [*namdbin file name*] [`velnamdbin`] [*velocity file name*]
  **Purpose:** Read in structure information from PSF file and add it to the structure. Optionally also read coordinates and insertion codes from a PDB file, assuming that the atom order is the same in both files. Optionally also read coordinates a NAMD binary file, assuming that the atom order is the same as the psf file. It is an error if any segments in the PSF file already exist.
  **Arguments:** <*file name*>: PSF file in X-PLOR format (names for atom types).
  `pdb`: Read coordinates and insertion codes from PDB file.
  <*pdb file name*>: PDB file with atoms in same order as PSF file.
  `namdbin`: Read coordinates from NAMD binary file.
  <*namdbin file name*>: NAMD binary file with atoms in same order as PSF file.
  `velnamdbin`: Read velocities from NAMD binary file.
  <*velocity file name*>: NAMD binary velocity file with atoms in same order as PSF file.
  **Context:** Anywhere but within segment.

- `pdbalias atom` <*residue name*> <*alternate name*> <*real name*>
  **Purpose:** Provide translations from atom names found in PDB files to proper atom names read in from topology definition files. Proper names from topology files will be used in generated PSF and PDB files. This command also exists under the deprecated name `alias`.
  **Arguments:** <*residue name*>: Proper or aliased residue name.
  <*alternate name*>: Atom name found in PDB file.
  <*real name*>: Atom name found in topology file.
  **Context:** Before reading coordinates with coordpdb. May call multiple times.

- `coordpdb` <*file name*> [*segid*] [`namdbin`] [*namdbin file name*]
  **Purpose:** Read coordinates from PDB file, matching segment, residue and atom names.
  **Arguments:** <*file name*>: PDB file containing known or aliased residues and atoms.
  <*segid*>: If specified override segment IDs in PDB file.
  `namdbin`: Read coordinates from NAMD binary file.
  <*namdbin file name*>: NAMD binary file with atoms in same order as PDB file.
  **Context:** After segment has been generated and atom aliases defined.

- `guesscoord`
  **Purpose:** Guesses coordinates of atoms for which they were not explicitly set. Calculation is based on internal coordinate hints contained in toplogy definition files. When these are insufficient, wild guesses are attempted based on bond lengths of 1 Å and angles of 109°.
  **Arguments:** None.
  **Context:** After stucture has been generated and known coordinates read in.

- `coord` <*segid*> <*resid*> <*atomname*> <{ $x$ $y$ $z$ }>
  **Purpose: WILL BE DEPRECATED AFTER VERSON 1.6** (use `psfset coord` instead) Set coordinates for a single atom.
  **Arguments:** <*segid*>: Segment ID of target atom.
  <*resid*>: Residue ID of target atom.
  <*atomname*>: Name of target atom.
  <{ $x$ $y$ $z$ }>: Coordinates to be assigned.
  **Context:** After structure has been generated.

- `psfset` <*attribute*> <*segid*> [*resid*] [*atomname*] <*value*>
  **Purpose:** Set an attribute for a given segment, residue, or atom.
  **Arguments:** <*attribute*>: Segment attributes: `segid`: the name of the segment Residue attributes: `resname`: the name of the residue Atom attributes: `name`: the name of the atom, `mass`: the mass of the atom, `charge`: the charge of the atom, `beta`: the PDB bfactor of the atom, `coord`: the coordinates of the atom as {x y z}, `vel`: the velocity of the atom as {vx vy vz}
  <*segid*>: Segment ID of target segment, residue, or atom.
  <*resid*>: Residue ID of target residue or atom.
  <*atomname*>: Name of target atom.
  <*value*>: Value to be assigned.
  **Context:** After structure has been generated or read from file.

- `writepdb` <*file name*>
  **Purpose:** Writes PDB file containing coordinates. Atom order is identical to PSF file generated by writepsf (unless structure has been changed). The O field is set to 1 for atoms with known coordinates, 0 for atoms with guessed coordinates, and -1 for atoms with no coordinate data available (coordinates are set to 0 for these atoms).
  **Arguments:** <*file name*>: PDB file to be written.
  **Context:** After structure and coordinates are complete.

- `writenamdbin` <*file name*> [`velnamdbin`] [*velocity file name*]
  **Purpose:** Writes NAMD binary file containing coordinates. Atom order is identical to PSF file generated by writepsf (unless structure has been changed). Coordinates are set to 0 for atoms with no coordinate data.
  **Arguments:** <*file name*>: NAMD binary file to be written.
  `velnamdbin`: Also write velocities to NAMD binary file.
  <*velocity file name*>: NAMD binary velocity file to be written.
  **Context:** After structure and coordinates are complete.

## 4.7 Example of a Session Log

The command “writepsf” prints a simple session log as “REMARKS” at the beginning of the PSF file. The log contains information about applied patches and used topology files which not stored in the standard records of PSF files. These informations are also available after a PSF file was read by command “readpsf”. Here’a a simple axample:

```
PSF

       1 !NTITLE
 REMARKS original generated structure x-plor psf file
 REMARKS 4 patches were applied to the molecule.
 REMARKS topology 1LOV_autopsf-temp.top
 REMARKS segment P1 { first NTER; last CTER; auto angles dihedrals }
 REMARKS segment O1 { first NONE; last NONE; auto none  }
 REMARKS segment W1 { first NONE; last NONE; auto none  }
 REMARKS defaultpatch NTER P1:1
 REMARKS defaultpatch CTER P1:104
 REMARKS patch DISU P1:10  P1:2
 REMARKS patch DISU P1:103  P1:6

    1704 !NATOM
       1 P1   1    ALA  N    NH3   -0.300000       14.0070           0
...
```

All patches that were applied explicitely using the “patch” command are listed following the keyword “patch”, but the patches that result from default patching like the first and last patches of a segment are marked as “defaultpatch”. Further the segment based patching rules are listed along with the angle/dihedral autogeneration rules.

# 5 Force Field Parameters

## 5.1 Potential energy functions

Evaluating the force is the most computationally demanding part of molecular dynamics. The force is the negative gradient of a scalar potential energy function,

$$\vec{F}(\vec{r}) = -\nabla U(\vec{r}), \tag{1}$$

and, for systems of biomolecules, this potential function involves the summing,

$$U(\vec{r}) = \sum U_{\text{bonded}}(\vec{r}) + \sum U_{\text{nonbonded}}(\vec{r}), \tag{2}$$

over a large number of bonded and nonbonded terms. The bonded potential terms involve 2–, 3–, and 4–body interactions of covalently bonded atoms, with $O(N)$ terms in the summation. The nonbonded potential terms involve interactions between all pairs of atoms (usually excluding pairs of atoms already involved in a bonded term), with $O(N^2)$ terms in the summation, although fast evaluation techniques are used to compute good approximations to their contribution to the potential with $O(N)$ or $O(N \log N)$ computational cost.

### 5.1.1 Bonded potential energy terms

The bonded potential terms involve 2–, 3–, and 4–body interactions of covalently bonded atoms.

The 2–body spring bond potential describes the harmonic vibrational motion between an $(i, j)$–pair of covalently bonded atoms,

$$U_{\text{bond}} = k(r_{ij} - r_0)^2, \tag{3}$$

where $r_{ij} = \|\vec{r}_j - \vec{r}_i\|$ gives the distance between the atoms, $r_0$ is the equilibrium distance, and $k$ is the spring constant.

The 3–body angular bond potential describes the angular vibrational motion occurring between an $(i, j, k)$–triple of covalently bonded atoms,

$$U_{\text{angle}} = k_\theta(\theta - \theta_0)^2 + k_{\text{ub}}(r_{ik} - r_{\text{ub}})^2, \tag{4}$$

where, in the first term, $\theta$ is the angle in radians between vectors $\vec{r}_{ij} = \vec{r}_j - \vec{r}_i$ and $\vec{r}_{kj} = \vec{r}_j - \vec{r}_k$, $\theta_0$ is the equilibrium angle, and $k_\theta$ is the angle constant. The second term is the Urey–Bradley term used to describe a (noncovalent) spring between the outer $i$ and $k$ atoms, active when constant $k_{\text{ub}} \neq 0$, where, like the spring bond, $r_{ik} = \|\vec{r}_k - \vec{r}_i\|$ gives the distance between the pair of atoms and $r_{\text{ub}}$ is the equilibrium distance.

The 4–body torsion angle (also known as dihedral angle) potential describes the angular spring between the planes formed by the first three and last three atoms of a consecutively bonded $(i, j, k, l)$–quadruple of atoms,

$$U_{\text{tors}} = \begin{cases} k(1 + \cos(n\psi + \phi)) & \text{if } n > 0, \\ k(\psi - \phi)^2 & \text{if } n = 0, \end{cases} \tag{5}$$

where $\psi$ is the angle in radians between the $(i, j, k)$–plane and the $(j, k, l)$–plane. The integer constant $n$ is nonnegative and indicates the periodicity. For $n > 0$, $\phi$ is the phase shift angle and $k$ is the multiplicative constant. For $n = 0$, $\phi$ acts as an equilibrium angle and the units of $k$ change to potential/rad$^2$. A given $(i, j, k, l)$–quadruple of atoms might contribute multiple terms to the potential, each with its own parameterization. The use of multiple terms for a torsion angle allows for complex angular variation of the potential, effectively a truncated Fourier series.

### 5.1.2 Nonbonded potential energy terms

The nonbonded potential terms involve interactions between all $(i, j)$–pairs of atoms, usually excluding pairs of atoms already involved in a bonded term. Even using a fast evaluation methods the cost of computing the nonbonded potentials dominates the work required for each time step of an MD simulation.

The Lennard–Jones potential accounts for the weak dipole attraction between distant atoms and the hard core repulsion as atoms become close,

$$U_{\mathrm{LJ}} = (-E_{\min}) \left[ \left( \frac{R_{\min}}{r_{ij}} \right)^{12} - 2 \left( \frac{R_{\min}}{r_{ij}} \right)^{6} \right], \tag{6}$$

where $r_{ij} = \|\vec{r}_j - \vec{r}_i\|$ gives the distance between the pair of atoms. The parameter $E_{\min} = U_{\mathrm{LJ}}(R_{\min})$ is the minimum of the potential term ($E_{\min} < 0$, which means that $-E_{\min}$ is the well-depth). The Lennard–Jones potential approaches 0 rapidly as $r_{ij}$ increases, so it is usually truncated (smoothly shifted) to 0 past a cutoff radius, requiring $O(N)$ computational cost.

The electrostatic potential is repulsive for atomic charges with the same sign and attractive for atomic charges with opposite signs,

$$U_{\mathrm{elec}} = \epsilon_{14} \frac{C q_i q_j}{\epsilon_0 r_{ij}}, \tag{7}$$

where $r_{ij} = \|\vec{r}_j - \vec{r}_i\|$ gives the distance between the pair of atoms, and $q_i$ and $q_j$ are the charges on the respective atoms. Coulomb's constant $C$ and the dielectric constant $\epsilon_0$ are fixed for all electrostatic interactions. The parameter $\epsilon_{14}$ is a unitless scaling factor whose value is 1, except for a modified 1–4 interaction, where the pair of atoms is separated by a sequence of three covalent bonds (so that the atoms might also be involved in a torsion angle interaction), in which case $\epsilon_{14} = \varepsilon$, for a fixed constant $0 \leq \varepsilon \leq 1$. Although the electrostatic potential may be computed with a cutoff like the Lennard–Jones potential, the $1/r$ potential approaches 0 much more slowly than the $1/r^6$ potential, so neglecting the long range electrostatic terms can degrade qualitative results, especially for highly charged systems. There are other fast evaluation methods that approximate the contribution to the long range electrostatic terms that require $O(N)$ or $O(N \log N)$ computational cost, depending on the method.

## 5.2 Non-bonded interactions

NAMD has a number of options that control the way that non-bonded interactions are calculated. These options are interrelated and can be quite confusing, so this section attempts to explain the behavior of the non-bonded interactions and how to use these parameters.

### 5.2.1 Van der Waals interactions

The simplest non-bonded interaction is the van der Waals interaction. In NAMD, van der Waals interactions are always truncated at the cutoff distance, specified by `cutoff`. The main option that effects van der Waals interactions is the `switching` parameter. With this option set to `on`, a smooth switching function will be used to truncate the van der Waals potential energy smoothly at the cutoff distance. A graph of the van der Waals potential with this switching function is shown in Figure 1. If `switching` is set to `off`, the van der Waals energy is just abruptly truncated at the cutoff distance, so that energy may not be conserved.

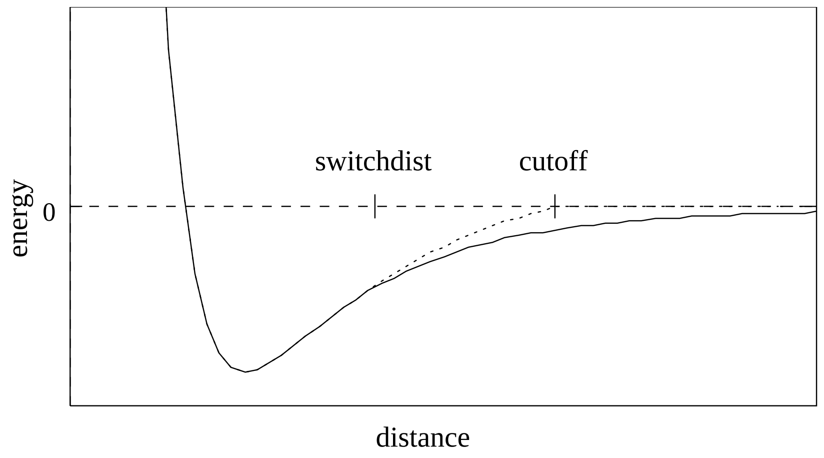


Figure 1: Graph of van der Waals potential with and without the application of the switching function. With the switching function active, the potential is smoothly reduced to 0 at the cutoff distance. Without the switching function, there is a discontinuity where the potential is truncated.

The switching function used is based on the X-PLOR switching function. The parameter `switchdist` specifies the distance at which the switching function should start taking effect to bring the van der Waals potential to 0 smoothly at the cutoff distance. Thus, the value of `switchdist` must always be less than that of `cutoff`.

### 5.2.2 Electrostatic interactions

The handling of electrostatics is slightly more complicated due to the incorporation of multiple timestepping for full electrostatic interactions. There are two cases to consider, one where full electrostatics is employed and the other where electrostatics are truncated at a given distance.

First let us consider the latter case, where electrostatics are truncated at the cutoff distance. Using this scheme, all electrostatic interactions beyond a specified distance are ignored, or assumed to be zero. If `switching` is set to `on`, rather than having a discontinuity in the potential at the cutoff distance, a shifting function is applied to the electrostatic potential as shown in Figure 2. As this figure shows, the shifting function shifts the entire potential curve so that the curve intersects the x-axis at the cutoff distance. This shifting function is based on the shifting function used by X-PLOR.

Next, consider the case where full electrostatics are calculated. In this case, the electrostatic interactions are not truncated at any distance. In this scheme, the `cutoff` parameter has a slightly different meaning for the electrostatic interactions — it represents the *local interaction distance*, or distance within which electrostatic pairs will be directly calculated every timestep. Outside of this distance, interactions will be calculated only periodically. These forces will be applied using a multiple timestep integration scheme as described in Section 7.3.4.

### 5.2.3 Non-bonded force field parameters

- `cutoff` < local interaction distance common to both electrostatic and van der Waals calculations (Å) >
  **Acceptable Values:** positive decimal
  **Description:** See Section 5.2 for more information.

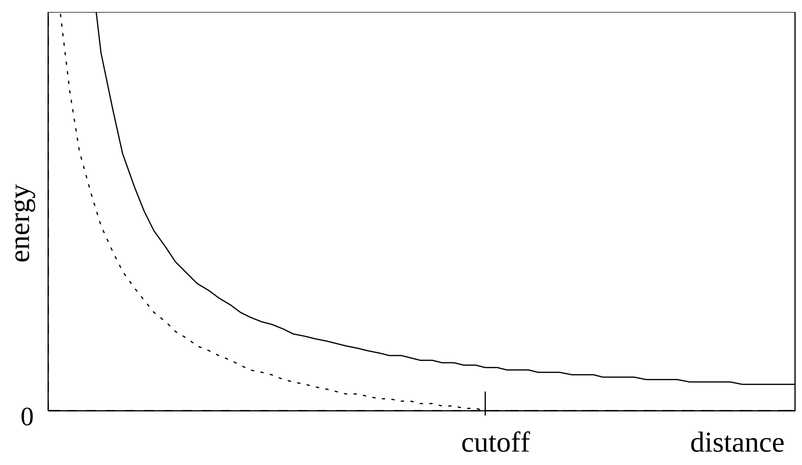


Figure 2: Graph showing an electrostatic potential with and without the application of the shifting function.

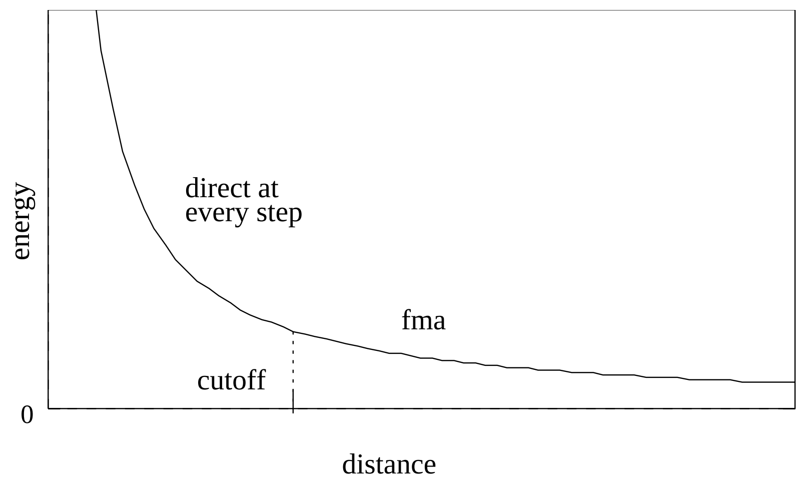


Figure 3: Graph showing an electrostatic potential when full electrostatics are used within NAMD, with one curve portion calculated directly and the other calculated using PME.

- `switching` < use switching function? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `on`
  **Description:** If `switching` is specified to be `off`, then a truncated cutoff is performed. If `switching` is turned `on`, then smoothing functions are applied to both the electrostatics and van der Waals forces. For a complete description of the non-bonded force parameters see Section 5.2. If `switching` is set to `on`, then `switchdist` must also be defined.

- `vdwForceSwitching` < use force switching for VDW? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** If both `switching` and `vdwForceSwitching` are set to `on`, then CHARMM force switching is used for van der Waals forces.

- `switchdist` < distance at which to activate switching/splitting function for electrostatic and van der Waals calculations (Å) >

**Acceptable Values:** positive decimal ≤ `cutoff`
**Description:** Distance at which the switching function should begin to take effect. This parameter only has meaning if `switching` is set to `on`. The value of `switchdist` must be less than or equal to the value of `cutoff`, since the switching function is only applied on the range from `switchdist` to `cutoff`. For a complete description of the non-bonded force parameters see Section 5.2.

- `exclude` < non-bonded exclusion policy to use >
  **Acceptable Values:** `none`, `1-2`, `1-3`, `1-4`, or `scaled1-4`
  **Description:** This parameter specifies which pairs of bonded atoms should be excluded from non-bonded interactions. With the value of `none`, no bonded pairs of atoms will be excluded. With the value of `1-2`, all atom pairs that are directly connected via a linear bond will be excluded. With the value of `1-3`, all `1-2` pairs will be excluded along with all pairs of atoms that are bonded to a common third atom (i.e., if atom A is bonded to atom B and atom B is bonded to atom C, then the atom pair A-C would be excluded). With the value of `1-4`, all `1-3` pairs will be excluded along with all pairs connected by a set of two bonds (i.e., if atom A is bonded to atom B, and atom B is bonded to atom C, and atom C is bonded to atom D, then the atom pair A-D would be excluded). With the value of `scaled1-4`, all `1-3` pairs are excluded and all pairs that match the `1-4` criteria are modified. The electrostatic interactions for such pairs are modified by the constant factor defined by `oneFourScaling`. The van der Waals interactions are modified by using the special 1-4 parameters defined in the parameter files. The value of `scaled1-4` is necessary to enable the modified 1-4 VDW parameters present in the CHARMM parameter files.

- `oneFourScaling` < scaling factor for 1-4 electrostatic interactions >
  **Acceptable Values:** 0 ≤ decimal ≤ 1
  **Default Value:** 1.0
  **Description:** Scaling factor for 1-4 electrostatic interactions. This factor is only used when the `exclude` parameter is set to `scaled1-4`. In this case, this factor is used to modify the electrostatic interactions between 1-4 atom pairs. If the `exclude` parameter is set to anything but `scaled1-4`, this parameter has no effect regardless of its value. Replaces the deprecated `1-4scaling` keyword.

- `dielectric` < dielectric constant for system >
  **Acceptable Values:** decimal ≥ 1.0
  **Default Value:** 1.0
  **Description:** Dielectric constant for the system. A value of 1.0 implies no modification of the electrostatic interactions. Any larger value will lessen the electrostatic forces acting in the system.

- `nonbondedScaling` < scaling factor for nonbonded forces >
  **Acceptable Values:** decimal ≥ 0.0
  **Default Value:** 1.0
  **Description:** Scaling factor for electrostatic and van der Waals forces. A value of 1.0 implies no modification of the interactions. Any smaller value will lessen the nonbonded forces acting in the system.

- `vdwGeometricSigma` < use geometric mean to combine L-J sigmas >
  **Acceptable Values:** `yes` or `no`

**Default Value:** `no`
**Description:** Use geometric mean, as required by OPLS, rather than traditional arithmetic mean when combining Lennard-Jones sigma parameters for different atom types.

- `limitdist` < maximum distance between pairs for limiting interaction strength(Å) >
  **Acceptable Values:** non-negative decimal
  **Default Value:** `0.`
  **Description:** The electrostatic and van der Waals potential functions diverge as the distance between two atoms approaches zero. The potential for atoms closer than `limitdist` is instead treated as $ar^2 + c$ with parameters chosen to match the force and potential at `limitdist`. This option should primarily be useful for alchemical free energy perturbation calculations, since it makes the process of creating and destroying atoms far less drastic energetically. The larger the value of `limitdist` the more the maximum force between atoms will be reduced. In order to not alter the other interactions in the simulation, `limitdist` should be less than the closest approach of any non-bonded pair of atoms; 1.3 Å appears to satisfy this for typical simulations but the user is encouraged to experiment. There should be no performance impact from enabling this feature.

- `LJcorrection` < Apply long-range corrections to the system energy and virial to account for neglected vdW forces? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Apply an analytical correction to the reported vdW energy and virial that is equal to the amount lost due to switching and cutoff of the LJ potential. The correction will use the average of vdW parameters for all particles in the system and assume a constant, homogeneous distribution of particles beyond the switching distance. See [68] for details (the equations used in the NAMD implementation are slightly different due to the use of a different switching function). Periodic boundary conditions are required to make use of tail corrections.

- `LJcorrectionAlt` < Apply alternative long-range corrections method to the system energy and virial to account for neglected vdW forces? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Similar to `LJcorrection`, an analytical correction to the reported vdW energy and virial is applied that is equal to the amount lost due to switching and cutoff of the LJ potential. The correction will use all VDW pair interactions, including atoms with no VDW parameters and self-self interactions and assume a constant, homogeneous distribution of particles beyond the switching distance. Periodic boundary conditions are required to make use of tail corrections.

### 5.2.4 PME parameters

PME stands for Particle Mesh Ewald and is an efficient full electrostatics method for use with periodic boundary conditions. None of the parameters should affect energy conservation, although they may affect the accuracy of the results and momentum conservation.

- `PME` < Use particle mesh Ewald for electrostatics? >
  **Acceptable Values:** `yes` or `no`

**Default Value:** `no`
**Description:** Turns on particle mesh Ewald.

- `PMETolerance` < PME direct space tolerance >
  **Acceptable Values:** positive decimal
  **Default Value:** $10^{-6}$
  **Description:** Affects the value of the Ewald coefficient and the overall accuracy of the results.

- `PMEInterpOrder` < PME interpolation order >
  **Acceptable Values:** positive integer
  **Default Value:** 4 (cubic)
  **Description:** Charges are interpolated onto the grid and forces are interpolated off using this many points, equal to the order of the interpolation function plus one.

- `PMEGridSpacing` < maximum space between grid points >
  **Acceptable Values:** positive real
  **Description:** The grid spacing partially determines the accuracy and efficiency of PME. If any of the grid sizes below are not set, then PMEGridSpacing must be set (recommended value is 1.0 Å) and will be used to calculate them. If a grid size is set, then the grid spacing must be at least PMEGridSpacing (if set, or a very large default of 1.5).

- `PMEGridSizeX` < number of grid points in x dimension >
  **Acceptable Values:** positive integer
  **Description:** The grid size partially determines the accuracy and efficiency of PME. For speed, `PMEGridSizeX` should have only small integer factors (2, 3 and 5).

- `PMEGridSizeY` < number of grid points in y dimension >
  **Acceptable Values:** positive integer
  **Description:** The grid size partially determines the accuracy and efficiency of PME. For speed, `PMEGridSizeY` should have only small integer factors (2, 3 and 5).

- `PMEGridSizeZ` < number of grid points in z dimension >
  **Acceptable Values:** positive integer
  **Description:** The grid size partially determines the accuracy and efficiency of PME. For speed, `PMEGridSizeZ` should have only small integer factors (2, 3 and 5).

- `PMEProcessors` < processors for FFT and reciprocal sum >
  **Acceptable Values:** positive integer
  **Default Value:** larger of x and y grid sizes up to all available processors
  **Description:** For best performance on some systems and machines, it may be necessary to restrict the amount of parallelism used. Experiment with this parameter if your parallel performance is poor when PME is used.

- `FFTWEstimate` < Use estimates to optimize FFT? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Do not optimize FFT based on measurements, but on FFTW rules of thumb. This reduces startup time, but may affect performance.

- `FFTWUseWisdom` < Use FFTW wisdom archive file? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `yes`
  **Description:** Try to reduce startup time when possible by reading FFTW "wisdom" from a file, and saving wisdom generated by performance measurements to the same file for future use. This will reduce startup time when running the same size PME grid on the same number of processors as a previous run using the same file.

- `FFTWWisdomFile` < name of file for FFTW wisdom archive >
  **Acceptable Values:** file name
  **Default Value:** FFTW_NAMD_*version_platform*.txt
  **Description:** File where FFTW wisdom is read and saved. If you only run on one platform this may be useful to reduce startup times for all runs. The default is likely sufficient, as it is version and platform specific.

### 5.2.5 MSM parameters

The multilevel summation method (MSM) [27] is an alternative to PME for calculating full electrostatic interactions. The use of the FFT in PME has two drawbacks: (1) it generally requires the use of periodic boundary conditions, in which the simulation describes an infinite three-dimensional lattice, with each lattice cell containing a copy of the simulated system, and (2) calculation of the FFT becomes a considerable performance bottleneck to the parallel scalability of MD simulations, due to the many-to-many communication pattern employed. MSM avoids the use of the FFT in its calculation, instead employing the nested interpolation in real space of softened pair potentials, which permits in addition to periodic boundary conditions the use of semi-periodic boundaries, in which there is periodicity along just one or two basis vectors, or non-periodic boundaries, in which the simulation is performed in a vacuum. Also, better parallel scaling has been observed with MSM when scaling a sufficiently large system to a large number of processors. See the MSM research web page (`http://www.ks.uiuc.edu/Research/msm/`) for more information.

In order to use the MSM, one need only specify "MSM on" in the configuration file. For production use, we presently recommend using the default "MSMQuality 0" ($C^1$ cubic interpolation with $C^2$ Taylor splitting), which has been validated to correctly reproduce the PME results [27]. At this time, we discourage use of the higher order interpolation schemes (Hermite, quintic, etc.), as they are still under development. With cubic interpolation, MSM now gets roughly half the performance of PME. Comparable performance and better scaling for MSM have been observed with the optimizations described in Ref. [27], which will be available shortly.

For now, NAMD's implementation of the MSM does not calculate the long-range electrostatic contribution to the virial, so use with a barostat for constant pressure simulation is inappropriate. (Note that the experiments in Ref. [27] involving constant pressure simulation with MSM made use of a custom version that is incompatible with some other NAMD features, so is not yet available.) The performance of PME is generally still better for smaller systems with smaller processor counts. MSM is the only efficient method in NAMD for calculating full electrostatics for simulations with semi-periodic or non-periodic boundaries.

The periodicity is defined through setting the cell basis vectors appropriately, as discussed in Sec. 7. The cutoff distance, discussed earlier in this section, also determines the splitting distance between the MSM short-range part, calculated exactly, and long-range part, interpolated from the grid hierarchy; this splitting distance is the primary control for accuracy for a given interpolation and splitting, although most simulations will likely want to keep the cutoff set to the CHARMM-prescribed value of 12 Å.

The configuration options specific to MSM are listed below. A simulation employing non-periodic boundaries in one or more dimensions might have atoms that attempt to drift beyond the predetermined extent of the grid. In the case that an atom does drift beyond the grid, the simulation will be halted prematurely with an error message. Several options listed below deal with defining the extent of the grid along non-periodic dimensions beyond what can be automatically determined by the initial coordinates. It is also recommended for non-periodic simulation to configure boundary restraints to contain the atoms, for instance, through Tcl boundary forces in Sec. 8.11.

- `MSM` < Use multilevel summation method for electrostatics? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Turns on multilevel summation method.

- `MSMGridSpacing` < spacing between finest level grid points (Å) >
  **Acceptable Values:** positive real
  **Default Value:** 2.5
  **Description:** The grid spacing determines in part the accuracy and efficiency of MSM. An error versus cost analysis shows that the best tradeoff is setting the grid spacing to a value close to the inter-particle spacing. The default value works well in practice for atomic scale simulation. This value will be exact along non-periodic dimensions. For periodic dimensions, the grid spacing must evenly divide the basis vector length; the actual spacing for a desired grid spacing $h$ is guaranteed to be within the interval $\left[\frac{4}{5}h, \frac{6}{5}h\right)$.

- `MSMQuality` < select the approximation quality >
  **Acceptable Values:** $0, 1, 2, 3, 4$
  **Default Value:** 0
  **Description:** This parameter offers a simplified way to select higher order interpolation and splitting for MSM. The available choices are:
  - 0 sets $C^1$ cubic ($p = 3$) interpolation with $C^2$ Taylor splitting,
  - 1 sets $C^1$ Hermite ($p = 4$) interpolation with $C^3$ Taylor splitting,
  - 2 sets $C^1$ quintic ($p = 5$) interpolation with $C^3$ Taylor splitting,
  - 3 sets $C^1$ septic ($p = 7$) interpolation with $C^4$ Taylor splitting,
  - 4 sets $C^1$ nonic ($p = 9$) interpolation with $C^5$ Taylor splitting.

  *We presently recommend using the default selection, which has been validated to correctly reproduce the PME results [27], and discourage use of the higher order interpolation schemes, as they are still under development.* With cubic interpolation, MSM now gets roughly half the performance of PME. Comparable performance and better scaling for MSM have been observed with the optimizations described in Ref. [27], which will be available shortly.

  There is generally a tradeoff between quality and performance. Empirical results show that the $C^1$ interpolation schemes offer a little better accuracy than the alternative interpolation schemes that have greater continuity. Also, better accuracy has been observed by using a splitting function with $C^{\lceil (p+1)/2 \rceil}$ continuity where $p$ is the order of the interpolant.

- `MSMApprox` < select the interpolant >
  **Acceptable Values:** $0, 1, \ldots, 7$
  **Default Value:** 0
  **Description:** Select the interpolation scheme:
  - 0 sets $C^1$ cubic ($p = 3$) interpolation,
  - 1 sets $C^1$ quintic ($p = 5$) interpolation,
  - 2 sets $C^2$ quintic ($p = 5$) interpolation,
  - 3 sets $C^1$ septic ($p = 7$) interpolation,
  - 4 sets $C^3$ septic ($p = 7$) interpolation,
  - 5 sets $C^1$ nonic ($p = 9$) interpolation,
  - 6 sets $C^4$ nonic ($p = 9$) interpolation,
  - 7 sets $C^1$ Hermite ($p = 4$) interpolation.

- `MSMSplit` < select the splitting >
  **Acceptable Values:** $0, 1, \ldots, 6$
  **Default Value:** 0
  **Description:** Select the splitting function:
  - 0 sets $C^2$ Taylor splitting,
  - 1 sets $C^3$ Taylor splitting,
  - 2 sets $C^4$ Taylor splitting,
  - 3 sets $C^5$ Taylor splitting,
  - 4 sets $C^6$ Taylor splitting,
  - 5 sets $C^7$ Taylor splitting,
  - 6 sets $C^8$ Taylor splitting.

- `MSMLevels` < maximum number of levels >
  **Acceptable Values:** non-negative integer
  **Default Value:** 0
  **Description:** Set the maximum number of levels to use in the grid hierarchy. Although setting slightly lower than the default might (or might not) improve performance and/or accuracy for non-periodic simulation, it is generally best to leave this at the default value "0" which will then automatically adjust the levels to the size of the given system.

- `MSMPadding` < grid padding (Å) >
  **Acceptable Values:** non-negative real
  **Default Value:** 2.5
  **Description:** The grid padding applies only to non-periodic dimensions, for which the extent of the grid is automatically determined by the maximum and minimum of the initial coordinates plus the padding value.

- `MSMxmin, MSMymin, MSMzmin` < minimum x-, y-, z-coordinate (Å) >
  **Acceptable Values:** real
  **Description:** Set independently the minimum x-, y-, or z-coordinates of the simulation. This parameter is applicable only to non-periodic dimensions. It is useful in conjunction with setting a boundary restraining force with Tcl boundary forces in Sec. 8.11.

- `MSMxmax, MSMymax, MSMzmax` < maximum x-, y-, z-coordinate (Å) >
  **Acceptable Values:** real
  **Description:** Set independently the maximum x-, y-, or z-coordinates of the simulation. This parameter is applicable only to non-periodic dimensions. It is useful in conjunction with setting a boundary restraining force with Tcl boundary forces in Sec. 8.11.

- `MSMBlockSizeX, MSMBlockSizeY, MSMBlockSizeZ` < block size for grid decomposition >
  **Acceptable Values:** positive integer
  **Default Value:** 8
  **Description:** Tune parallel performance by adjusting the block size used for parallel domain decomposition of the grid. Recommended to keep the default.

- `MSMSerial` < Use serial long-range solver? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Enable instead the slow serial long-range solver. Intended to be used only for testing and diagnostic purposes.

### 5.2.6 Full direct parameters

The direct computation of electrostatics is not intended to be used during real calculations, but rather as a testing or comparison measure. Because of the $\mathcal{O}(N^2)$ computational complexity for performing direct calculations, this is *much* slower than using PME or MSM to compute full electrostatics for large systems. In the case of periodic boundary conditions, the nearest image convention is used rather than a full Ewald sum.

- `FullDirect` < calculate full electrostatics directly? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Specifies whether or not direct computation of full electrostatics should be performed.

### 5.2.7 Tabulated nonbonded interaction parameters

In order to support coarse grained models and semiconductor force fields, the tabulated energies feature replaces the normal van der Waals potential for specified pairs of atom types with one interpolated from user-supplied energy tables. The electrostatic potential is not altered.

Pairs of atom types to which the modified interactions apply are specified in a CHARMM parameter file by an `NBTABLE` section consisting of lines with two atom types and a corresponding interaction type name. For example, tabulated interactions for SI-O, O-O, and SI-SI pairs would be specified in a parameter file as:

```
NBTABLE
SI O SIO
O O OO
SI SI SISI
```

Each interaction type must correspond to an entry in the energy table file. The table file consists of a header formatted as:

```
# multiple comment lines
<number_of_tables> <table_spacing (A)> <maximum_distance (A)>
```

followed by *number_of_tables* energy tables formatted as:

```
TYPE <interaction type name>
0 <energy (kcal/mol)> <force (kcal/mol/A)>
<table_spacing> <energy (kcal/mol)> <force (kcal/mol/A)>
<2*table_spacing> <energy (kcal/mol)> <force (kcal/mol/A)>
<3*table_spacing> <energy (kcal/mol)> <force (kcal/mol/A)>
...
<maximum_distance - 3*table_spacing> <energy (kcal/mol)> <force (kcal/mol/A)>
<maximum_distance - 2*table_spacing> <energy (kcal/mol)> <force (kcal/mol/A)>
<maximum_distance - table_spacing> <energy (kcal/mol)> <force (kcal/mol/A)>
```

The table entry at *maximum_distance* will match the energy of the previous entry but have a force of zero. The maximum distance must be at least equal to the nonbonded cutoff distance and entries beyond the cutoff distance will be ignored. For the above example with a cutoff of 12 Å the table file could look like:

```
# parameters for silicon dioxide
3 0.01 14.0
TYPE SIO
0 5.092449e+26 3.055469e+31
0.01 5.092449e+14 3.055469e+17
0.02 7.956951e+12 2.387085e+15
0.03 6.985526e+11 1.397105e+14
...
13.98 0.000000e+00 -0.000000e+00
13.99 0.000000e+00 -0.000000e+00
TYPE OO
0 1.832907e+27 1.099744e+32
0.01 1.832907e+15 1.099744e+18
0.02 2.863917e+13 8.591751e+15
0.03 2.514276e+12 5.028551e+14
...
13.98 0.000000e+00 -0.000000e+00
13.99 0.000000e+00 -0.000000e+00
TYPE SISI
0 0.000000e+00 -0.000000e+00
0.01 0.000000e+00 -0.000000e+00
...
13.98 0.000000e+00 -0.000000e+00
13.99 0.000000e+00 -0.000000e+00
```

The following three parameters are required for tabulated energies.

- `tabulatedEnergies` < use tabulated energies >
  **Acceptable Values:** `yes` or `no`

**Default Value:** `no`
**Description:** Specifies whether or not tabulated energies will be used for van der Waals interactions between specified pairs of atom types.

- `tabulatedEnergiesFile` < file containing energy table >
  **Acceptable Values:** file name
  **Description:** Provides one energy table for each interaction type in parameter file. See format above.

- `tableInterpType` < cubic or linear interpolation >
  **Acceptable Values:** `cubic` or `linear`
  **Description:** Specifies the order for interpolating between energy table entries.

## 5.3 Water Models

NAMD currently supports the 3-site TIP3P water model, the 4-site TIP4P water model, and the 5-site SWM4-NDP water model (from the Drude force field) [38]. TIP3P is the current default water model. Usage of alternative water models is described below.

- `waterModel` < using which water model? >
  **Acceptable Values:** `tip3`, `tip4`, `swm4`
  **Default Value:** `tip3`
  **Description:** Specifies the water model to be used. When using the TIP3P water model, the ordering of atoms within each TIP3P water molecule must be oxygen, hydrogen, hydrogen. When using the TIP4P water model, the ordering of atoms within each TIP4P water molecule must be oxygen, hydrogen, hydrogen, lone pair. When using the SWM4-NDP water model, the ordering of atoms within each SWM4-NDP water molecule must be oxygen, Drude particle, lone pair, hydrogen, hydrogen. Alternative orderings will fail.

## 5.4 Drude polarizable force field

The Drude oscillator model represents induced electronic polarization by introducing an auxiliary particle attached to each polarizable atom via a zero-length harmonic spring. The advantage with the Drude model is that it preserves the simple particle-particle Coulomb electrostatic interaction employed in nonpolarizable force fields, thus its implementation in NAMD is more straightforward than alternative models for polarization. NAMD performs the integration of Drude oscillators by employing a novel dual Langevin thermostat to "freeze" the Drude oscillators while maintaining the normal "warm" degrees of freedom at the desired temperature [31]. Use of the Langevin thermostat enables better parallel scalability than the earlier reported implementation which made use of a dual Nosé-Hoover thermostat acting on, and within, each nucleus-Drude pair [39]. Performance results show that the NAMD implementation of the Drude model maintains good parallel scalability with an increase in computational cost by not more than twice that of using a nonpolarizable force field [31].

Excessive "hyperpolarization" of Drude oscillators can be prevented by two different schemes. The default "hard wall" option reflects elongated springs back towards the nucleus using a simple collision model. Alternatively, the Drude oscillators can be supplemented by a flat-bottom quartic restraining potential (usually with a large force constant).

The Drude polarizable force field requires some extensions to the CHARMM force field. An anisotropic spring term is added to account for out-of-plane forces from a polarized atom and its covalently bonded neighbor with two more covalently bonded neighbors (similar in structure to an improper bond). The screened Coulomb correction of Thole is calculated between pairs of Drude oscillators that would otherwise be excluded from nonbonded interaction and optionally between non-excluded, nonbonded pairs of Drude oscillators that are within a prescribed cutoff distance [79, 80]. Also included in the Drude force field are the use of off-centered massless interaction sites, so called "lone pairs" (LPs), to avoid the limitations of centrosymmetric-based Coulomb interactions [26]. The coordinate of each LP site is constructed based on three "host" atoms. The calculated forces on the massless LP must be transferred to the host atoms, preserving total force and torque. After an integration step of velocities and positions, the position of the LP is updated based on the three host atoms, along with additional geometry parameters that give displacement and in-plane and out-of-plane angles. See our research web page (`http://www.ks.uiuc.edu/Research/Drude/`) for additional details and parallel performance results. There are review papers that provide mathematical details of the Drude polarizable force field terms and the dual Langevin thermostat [40, 29].

### 5.4.1 Required input files

No additional files are required by NAMD to use the Drude polarizable force field. However, it is presently beyond the capability of the `psfgen` tool to generate the PSF file needed to perform a simulation using the Drude model. For now, CHARMM is needed to generate correct input files.

The CHARMM force field parameter files specific to the Drude model are required. The PDB file must also include the Drude particles (mass between 0.05 and 1.0) and the LPs (mass 0). The Drude particles always immediately follow their parent atom. The PSF file augments the "atom" section with additional columns that include the "Thole" and "alpha" parameters for the screened Coulomb interactions of Thole. The PSF file also requires additional sections that list the LPs, including their host atoms and geometry parameters, and list the anisotropic interaction terms, including their parameters. A Drude-compatible PSF file is denoted by the keyword "DRUDE" given along the top line.

### 5.4.2 Standard output

The NAMD logging to standard output is extended to provide additional temperature data on the cold and warm degrees of freedom. Four additional quantities are listed on the `ETITLE` and `ENERGY` lines:

`DRUDECOM` gives the temperature for the warm center-of-mass degrees of freedom,

`DRUDEBOND` gives the temperature for the cold Drude oscillator degrees of freedom,

`DRCOMAVG` gives the average temperature (averaged since the previously reported temperature) for the warm center-of-mass degrees of freedom,

`DRBONDAVG` gives the average temperature (averaged since the previously reported temperature) for the cold Drude oscillator degrees of freedom.

The energies resulting from the Drude oscillators and the anisotropic interactions are summed into the `BOND` energy. The energies resulting from the LPs and the screened Coulomb interactions of Thole are summed into the `ELECT` energy.

### 5.4.3 Drude force field parameters

The Drude model should be used with the Langevin thermostat enabled (`Langevin=on`). Doing so permits the use of normal sized time steps (e.g., 1 fs). The Drude model is also compatible with constant pressure simulation using the Langevin piston. Long-range electrostatics may be calculated using PME. The nonbonded exclusions should generally be set to use either the 1-3 exclusion policy (`exclude=1-3`) or the scaled 1-4 exclusion policy (`exclude=scaled1-4`).

The Drude water model (SWM4-NDP) is a 5-site model with four charge sites and a negatively charged Drude particle [38], with the particles ordered in the input files as oxygen, Drude particle, LP, hydrogen, hydrogen. The atoms in the water molecules should be constrained (`rigidBonds=water`), with use of the SETTLE algorithm recommended (`useSettle=on`). Explicitly setting the water model (`waterModel=swm4`) is optional.

- `drude` < Perform integration of Drude oscillators? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** The integration uses a dual Langevin thermostat to freeze the Drude oscillators while maintaining the warm degrees of freedom at the desired temperature. Must also enable the Langevin thermostat. If `drude` is set to `on`, then `drudeTemp` must also be defined.

- `drudeTemp` < temperature for freezing the Drude oscillators (K) >
  **Acceptable Values:** non-negative decimal
  **Description:** For stability, the Drude oscillators must be kept at a very cold termpature. Using a Langevin thermostat, it is possible to set this temperature to 0 K.

- `drudeDamping` < damping coefficient for Drude oscillators (1/ps) >
  **Acceptable Values:** positive decimal
  **Description:** The Langevin coupling coefficient to be applied to the Drude oscillators. If not given, `drudeDamping` is set to the value of `langevinDamping`, but values of as much as an order of magnitude greater may be appropriate.

- `drudeNbTholeCut` < nonbonded Thole interaction radius (Å) >
  **Acceptable Values:** positive decimal
  **Default Value:** 5.0
  **Description:** If `drudeNbTholeCut` is non-zero, the screened Coulomb correction of Thole is also calculated for non-excluded, nonbonded pairs of Drude oscillators that are within this radius of interaction.

- `drudeHardWall` < use collisions to correct hyperpolarization? >
  **Acceptable Values:** on or off
  **Default Value:** on
  **Description:** Excessively elongated Drude oscillator bonds are avoided by reflective collisions induced at a fixed cutoff, `drudeBondLen`. A large number of such events is usually indicative of unstable/unphysical dynamics and a simulation will stop if double the cutoff is exceeded.

- `drudeBondLen` < hyperpolarization cutoff (Å) >
  **Acceptable Values:** positive decimal

**Default Value:** 0.25
**Description:** If using `drudeHardWall on`, this is the distance at which collisions occur. Otherwise, this is the distance at which an additional quartic restraining potential is applied to each Drude oscillator. In this latter case, a value of 0.2 Å (slightly smaller than default) is recommended.

- `drudeBondConst` < Drude oscillator restraining force constant >
  **Acceptable Values:** positive decimal
  **Default Value:** 40000.0
  **Description:** If `drudeHardWall off` and `drudeBondConst` is non-zero, an additional quartic restraining potential is applied to a Drude oscillator if its length exceeds `drudeBondLen`.

## 5.5 MARTINI Residue-Based Coarse-Grain Forcefield

The MARTINI forcefield for residue-based coarse-grain models allows simulation of several tens of atoms as only several large coarse-grained particles [46, 47, 52]. In the MARTINI model, each protein residue is represented by a backbone bead and usually one or more sidechain beads.

When preparing MARTINI simulations it is important to include only those dihedrals specified by the forcefield. Using the "auto dihedrals" or "regenerate dihedrals" feature of psfgen will create dihedrals for all possible sets of four bonded atoms. This is incorrect for MARTINI and will result in energy jumps because the dihedral potential function is degenerate for the angles of 180 degrees allowed by cosine-based angles.

When using MARTINI the following configuration parameters should be set as indicated:

```
cosAngles on
martiniSwitching on
dielectric 15.0
PME off
```

- `cosAngles` < enable the MARTINI cosine-based angle potential function >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not the MARTINI forcefield is being used, specifically cosine-based angle potential function. The cosine-based potential will only be used for angles in CHARMM parameter files that specify the cos keyword.

- `martiniSwitching` < enable the MARTINI Lennard-Jones switching function? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not the MARTINI forcefield is being used, specifically the Lennard-Jones switching function.

- `martiniDielAllow` < Allow dielectrics != 15.0 for use with MARTINI >
  **Acceptable Values:** `on` or `off`
  **Description:** `off` Allows user to specify a `dielectric` not equal to 15.0, ie a non-standard dielectric for MARTINI.

## 5.6 Constraints and Restraints

### 5.6.1 Bond constraint parameters

- `rigidBonds` < controls if and how ShakeH is used >
  **Acceptable Values:** `none`, `water`, `all`
  **Default Value:** `none`
  **Description:** When `water` is selected, the hydrogen-oxygen and hydrogen-hydrogen distances in waters are constrained to the nominal length or angle given in the parameter file, making the molecules completely rigid. When `rigidBonds` is `all`, waters are made rigid as described above and the bond between each hydrogen and the (one) atom to which it is bonded is similarly constrained. For the default case `none`, no lengths are constrained.

- `rigidTolerance` < allowable bond-length error for ShakeH (Å) >
  **Acceptable Values:** positive decimal
  **Default Value:** 1.0e-8
  **Description:** The ShakeH algorithm is assumed to have converged when all constrained bonds differ from the nominal bond length by less than this amount.

- `rigidIterations` < maximum ShakeH iterations >
  **Acceptable Values:** positive integer
  **Default Value:** 100
  **Description:** The maximum number of iterations ShakeH will perform before giving up on constraining the bond lengths. If the bond lengths do not converge, a warning message is printed, and the atoms are left at the final value achieved by ShakeH. Although the default value is 100, convergence is usually reached after fewer than 10 iterations.

- `rigidDieOnError` < maximum ShakeH iterations >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `on`
  **Description:** Exit and report an error if rigidTolerance is not achieved after rigidIterations.

- `useSettle` < Use SETTLE for waters. >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `on`
  **Description:** If rigidBonds are enabled then use the non-iterative SETTLE algorithm to keep waters rigid rather than the slower SHAKE algorithm.

### 5.6.2 Position restraint parameters

The following describes the parameters for the position restraints feature of NAMD. For historical reasons the term "constraints" has been carried over from X-PLOR. This feature allows a restraining potential to each atom of an arbitrary set during the simulation.

- `constraints` < are position restraints active? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not position restraints are active. If it is set to `off`, then no position restraints are computed. If it is set to `on`, the potential $k \times (\mathbf{x} - \mathbf{x}_0)^p$ is applied

to each atom. Per-atom values for $k$ can be defined by either `conskfile` or `conskcol`, for $\mathbf{x}_0$ by `consref`, and for $p$ by `consexp`.

- `consexp` < exponent for position restraint energy function >
  **Acceptable Values:** positive, even integer
  **Default Value:** 2
  **Description:** Exponent to be use in the position restraint energy function. This value must be a positive integer, and only even values really make sense. This parameter is used only if `constraints` is set to `on`.

- `consref` < PDB file containing restraint reference positions >
  **Acceptable Values:** UNIX file name
  **Description:** PDB file to use for reference positions for position restraints. Each atom that has a positive force constant will be restrained about the position specified in this file.

- `conskfile` < PDB file containing force constant values >
  **Acceptable Values:** UNIX filename
  **Description:** PDB file to use for force constants for position restraints.

- `conskcol` < column of PDB file containing force constant >
  **Acceptable Values:** `X`, `Y`, `Z`, `O`, or `B`
  **Description:** Column of the PDB file to use for the position restraint force constant. This parameter may specify any of the floating point fields of the PDB file, either X, Y, Z, occupancy, or beta-coupling (temperature-coupling). Regardless of which column is used, a value of 0 indicates that the atom qshould not be restrained. Otherwise, the value specified is used as the force constant for that atom's restraining potential.

- `constraintScaling` < scaling factor for position restraint energy function >
  **Acceptable Values:** positive
  **Default Value:** 1.0
  **Description:** The position restraint energy function is multiplied by this parameter, making it possible to gradually turn off restraints during equilibration. This parameter is used only if `constraints` is set to `on`.

- `selectConstraints` < Restrain only selected Cartesian components of the coordinates? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** This option is useful to restrain the positions of atoms to a plane or a line in space. If active, this option will ensure that only selected Cartesian components of the coordinates are restrained. E.g.: Restraining the positions of atoms to their current z values with no restraints in x and y will allow the atoms to move in the x-y plane while retaining their original z-coordinate. Restraining the x and y values will lead to free motion only along the z coordinate.

- `selectConstrX` < Restrain X components of coordinates >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Restrain the Cartesian x components of the positions.

- `selectConstrY` < Restrain Y components of coordinates >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Restrain the Cartesian y components of the positions.

- `selectConstrZ` < Restrain Z components of coordinates >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Restrain the Cartesian z components of the positions.

### 5.6.3 Group position restraint parameters

The following describes the parameters for the group position restraints feature of NAMD. **Please note that this feature is only available in GPU-resident version of NAMD (NAMD3).** This feature allows restraining the center of mass (COM) of a group of atoms to a reference position or another COM of group of atoms, during the simulation.
The restraining potential is defined as:

$$E_{\text{restraint}} = k \times (\xi - \xi_{\mathbf{0}})^n$$
$$\xi = \begin{cases} \overrightarrow{r}_2^{\text{COM}} - \overrightarrow{r}_1^{\text{COM}} & \text{if using distance vector} \\ \|\overrightarrow{r}_2^{\text{COM}} - \overrightarrow{r}_1^{\text{COM}}\| & \text{if using distance magnitude} \end{cases}$$
$$\xi_{\mathbf{0}} = \begin{cases} \overrightarrow{r}_{\text{restraint}}^{\text{center}} & \text{if using distance vector} \\ \|\overrightarrow{r}_{\text{restraint}}^{\text{center}}\| & \text{if using distance magnitude} \end{cases}$$

where $k$ is restraint force constant defined by `groupResK` and $n$ is the restraint exponent defined by `groupResExp`.
$\xi$ is the distance vector or length that connects the COM of group 1 to the COM of group 2. In case of defining a reference position for COM of group 1 (`group1RefPos`), $\xi$ specifies the distance vector or length that connects the reference position to the COM of group 2.
$\xi_{\mathbf{0}}$ is the restraint center vector defined by `groupResCenter`. $\overrightarrow{r}_{\mathbf{2}}^{\text{COM}}$ is calculated COM for the atoms in group 2, where atom indices are defined in `group2File` or `group2List`. $\overrightarrow{r}_{\mathbf{1}}^{\text{COM}}$ is either the reference position defined by `group1RefPos` or calculated COM for the atoms in group 1, where atom indices are defined in `group1File` or `group1List`.

Once group position restraints are activated, NAMD outputs additional restraints information for each restraint group. With the frequency of energy outputs, NAMD prints the following fields: **RES_TITLE: TS GROUP_NAME DISTANCE.X DISTANCE.Y DISTANCE.Z FORCE.X FORCE.Y FORCE.Z ENERGY** where "TS", "GROUP_NAME", "DISTANCE", "FORCE", "ENERGY" is timestep, assigned tag for the group restraint, distance vector (Å) $\overrightarrow{r}_{\mathbf{2}}^{\text{COM}} - \overrightarrow{r}_{\mathbf{1}}^{\text{COM}}$, applied restraint force (kcal/mol × Å) along $\frac{\overrightarrow{r}_{\mathbf{2}}^{\text{COM}} - \overrightarrow{r}_{\mathbf{1}}^{\text{COM}}}{\|\overrightarrow{r}_2^{\text{COM}} - \overrightarrow{r}_1^{\text{COM}}\|}$, and restraint energy (kcal/mol), respectively.

- `groupRestraints` < are group position restraints active? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not group position restraints are active.

- `group1File` < tag > < group 1 atom indices file name >
  **Acceptable Values:** UNIX file name
  **Description:** File specifying the restrained atom indices in group 1. In this file, **zero-based** atom index (the first atom has index 0) is used, where each index is separated by a newline. The COM of atoms specified in this file will be restrained about the COM of atoms specified in `group2File` or `group2List`.

- `group1List` < tag > < group 1 list of atom indices >
  **Acceptable Values:** list of integer
  **Description:** List of restrained atom indices in group 1. In this list, **zero-based** atom index (the first atom has index 0) is used. The COM of atoms specified in this list will be restrained about the COM of atoms specified in `group2File` or `group2List`.

- `group1RefPos` < tag > < group 1 reference COM position >
  **Acceptable Values:** float triple (x y z)
  **Description:** Reference COM positions for group 1. The COM of atoms specified in `group2File` or `group2List` will be restrained about this position.

- `group2File` < tag > < group 2 atom indices file name >
  **Acceptable Values:** UNIX file name
  **Description:** File specifying the restrained atom indices in group 2. In this file, **zero-based** atom index (the first atom has index 0) is used, where each index is separated by a newline. The COM of atoms specified in this file will be restrained about `group1RefPos` or the COM of atoms specified in `group1File` or `group1List`.

- `group2List` < tag > < group 2 list of atom indices >
  **Acceptable Values:** list of integer
  **Description:** List of restrained atom indices in group 2. In this list, **zero-based** atom index (the first atom has index 0) is used. The COM of atoms specified in this list will be restrained about `group1RefPos` or the COM of atoms specified in `group1File` or `group1List`.

- `groupResExp` < tag > < restraint exponent >
  **Acceptable Values:** positive, even integer
  **Default Value:** 2
  **Description:** Exponent to be used in the group position restraint energy function. This value must be a positive even integer.

- `groupResK` < tag > < restraint force constant >
  **Acceptable Values:** positive float
  **Description:** Force constants to be used in group position restraint energy function. This value must be a positive float.

- `groupResCenter` < tag > < restraint center >
  **Acceptable Values:** float triple (x y z)
  **Description:** Equilibrium or center value in group position restraint energy function. This value must be a vector.

- `groupResUseMagnitude` < tag > < use distance magnitude >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`

**Description:** Specifies if distance magnitude or distance vector is used in group position restraint energy function. If `groupResUseMagnitude on`, the distance magnitude $\|\vec{r}_{\mathbf{2}}^{\mathrm{COM}} - \vec{r}_{\mathbf{1}}^{\mathrm{COM}}\|$ is used in the group position restraint potential.

- `groupResX` < tag > < Restrain X components of distance >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `on`
  **Description:** Restrain the Cartesian X components of the distance $\vec{r}_{\mathbf{2}}^{\mathrm{COM}} - \vec{r}_{\mathbf{1}}^{\mathrm{COM}}$.

- `groupResY` < tag > < Restrain Y components of distance >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `on`
  **Description:** Restrain the Cartesian Y components of the distance $\vec{r}_{\mathbf{2}}^{\mathrm{COM}} - \vec{r}_{\mathbf{1}}^{\mathrm{COM}}$.

- `groupResZ` < tag > < Restrain Z components of distance >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `on`
  **Description:** Restrain the Cartesian Z components of the $\vec{r}_{\mathbf{2}}^{\mathrm{COM}} - \vec{r}_{\mathbf{1}}^{\mathrm{COM}}$.

### 5.6.4 Fixed atoms parameters

Atoms may be held fixed during a simulation. NAMD avoids calculating most interactions in which all affected atoms are fixed unless `fixedAtomsForces` is specified.

- `fixedAtoms` < are there fixed atoms? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not fixed atoms are present.

- `fixedAtomsForces` < are forces between fixed atoms calculated? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not forces between fixed atoms are calculated. This option is required to turn fixed atoms off in the middle of a simulation. These forces will affect the pressure calculation, and you should leave this option off when using constant pressure if the coordinates of the fixed atoms have not been minimized. The use of constant pressure with significant numbers of fixed atoms is not recommended.

- `fixedAtomsFile` < PDB file containing fixed atom parameters >
  **Acceptable Values:** UNIX filename
  **Default Value:** `coordinates`
  **Description:** PDB file to use for the fixed atom flags for each atom. If this parameter is not specified, then the PDB file specified by `coordinates` is used.

- `fixedAtomsCol` < column of PDB containing fixed atom parameters >
  **Acceptable Values:** `X`, `Y`, `Z`, `O`, or `B`
  **Default Value:** `O`
  **Description:** Column of the PDB file to use for the containing fixed atom parameters for each atom. The coefficients can be read from any floating point column of the PDB file. A value of 0 indicates that the atom is not fixed.

### 5.6.5 Extra bond, angle, and dihedral restraints

Additional bond, angle, and dihedral energy terms may be applied to system, allowing secondary or tertiary structure to be restrained, for example. Extra bonded terms are not considered part of the molecular structure and hence do not alter nonbonded exclusions. The energies from extra bonded terms are included with the normal bond, angle, and dihedral energies in NAMD output.

All extra bonded terms are harmonic potentials of the form $U(x) = k(x - x_{ref})^2$ except dihedrals and impropers with a non-zero periodicity specified, which use $U(x) = k(1 + cos(nx - x_{ref}))$. The only difference between dihedrals and impropers is the output field that their potential energy is added to.

Due to a very old bug all NAMD releases prior to 2.13 have used the MARTINI cosine-based angle potential function for all extra angles. Since workflows may unknowingly depend on this undocumented behavior, cosine-based angles remain the default, but a warning is printed unless the desired behavior is specified via the new option extraBondsCosAngles (defaults to "on", set to "off" to use the normal harmonic angle potential function for all extra angles).

The extra bonded term implementation shares the parallel implementation of regular bonded terms in NAMD, allowing large numbers of extra terms to be specified with minimal impact on parallel scalability. Extra bonded terms do not have to duplicate normal bonds/angles/dihedrals, but each extra bond/angle/dihedral should only involve nearby atoms. If the atoms involved are too far apart a bad global bond count will be reported in parallel runs.

Extra bonded terms are enabled via the following options:

- `extraBonds` < enable extra bonded terms? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not extra bonded terms are present.

- `extraBondsCosAngles` < are extra angles cosine-based? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `on`
  **Description:** Specifies whether or not all extra angles are cosine-based for consistency with previous versions. Set to `off` to use normal harmonic angle potential for all extra angles.

- `extraBondsFile` < file containing extra bonded terms >
  **Acceptable Values:** file
  **Description:** File containing extra bonded terms. May be repeated for multiple files.

The extra bonds file(s) should contain lines of the following formats:

- `bond <atom> <atom> <k> <ref>`
- `angle <atom> <atom> <atom> <k> <ref>`
- `dihedral <atom> <atom> <atom> <atom> <k> <ref>`
- `dihedral <atom> <atom> <atom> <atom> <k> <n> <ref>`
- `improper <atom> <atom> <atom> <atom> <k> <ref>`
- `improper <atom> <atom> <atom> <atom> <k> <n> <ref>`

- `wall <atom> <atom> <k> <lower> <upper>`
- `# <comment ...>`

In all cases `<atom>` is a **zero-based** atom index (the first atom has index 0), `<ref>` is a reference distance in Å (bond) or angle in degrees (others), and `<k>` is a spring constant in the potential energy function $U(x) = k(x - x_{ref})^2$ or, for dihedrals and impropers with periodicity `<n>` specified and not 0, $U(x) = k(1 + cos(nx - x_{ref}))$. Note that $x_{ref}$ is only a minimum for the harmonic potential; the sinusoidal potential has minima at $(x_{ref} + 180)/n + i \times 360/n$.

Use of `wall` implements a harmonic wall potential similar to the Colvars harmonic wall restraint. The potential function is

$$U(x) = \begin{cases} k(x - x_{\text{upper}})^2, & \text{if } x > x_{\text{upper}} \\ 0, & \text{if } x_{\text{lower}} \leq x \leq x_{\text{upper}} \\ k(x - x_{\text{lower}})^2, & \text{if } x < x_{\text{lower}} \end{cases} .$$

## 6 Generalized Born Implicit Solvent

Generalized Born implicit solvent (GBIS) is a fast but approximate method for calculating molecular electrostatics in solvent as described by the Poisson Boltzmann equation which models water as a dielectric continuum. GBIS enables the simulation of atomic structures without including explicit solvent water. The elimination of explicit solvent greatly accelerates simulations; this speedup is lessed by the increased computational complexity of the implicit solvent electrostatic calculation and a longer interaction cutoff. These are discussed in greater detail below.

### 6.1 Theoretical Background

Water has many biologically necessary properties, one of which is as a dielectric. As a dielectric, water screens (lessens) electrostatic interactions between charged particles. Water can therefore be crudely modeled as a dielectric continuum. In this manner, the electrostatic forces of a biological system can be expressed as a system of differential equations which can be solved for the electric field caused by a collection of charges.

#### 6.1.1 Poisson Boltzmann Equation

The Poisson Boltzmann equation (PBE),

$$\vec{\nabla} \cdot \left[\epsilon(\vec{r})\vec{\nabla}\Psi(\vec{r})\right] = -4\pi\rho^f(\vec{r}) - 4\pi\sum_i c_i^\infty q_i \lambda(\vec{r}) \cdot \exp\left[\frac{-q_i\Psi(\vec{r})}{k_B T}\right]$$

is a nonlinear equation which solves for the electrostatic field, $\Psi(\vec{r})$, based on the position dependent dielectric, $\epsilon(\vec{r})$, the position-dependent accessibility of position $\vec{r}$ to the ions in solution, $\lambda(\vec{r})$, the solute charge distribution, $\rho^f(\vec{r})$, and the bulk charge density, $c_i^\infty$, of ion $q_i$. While this equation does exactly solve for the electrostic field of a charge distribution in a dielectric, it is very expensive to solve, and therefore not suitable for molecular dynamics.

#### 6.1.2 Generalized Born

The Generalized Born (GB) equation is an approximation of the PBE. It models atoms as charged spheres whose internal dielectric is lower than that of the environment. The screening which each atom, $i$, experiences is determined by the local environment; the more atom $i$ is surrounded by other atoms, the less it's electrostatics will be screened since it is more surrounded by low dielectric; this property is called one atom descreening another. Different GB models calculate atomic descreening differently. Descreening is used to calculate the Born radius, $\alpha_i$, of each atom. The Born radius of an atom measures the degree of descreening. A large Born radius represents small screening (strong electric field) as if the atom were in vacuum. A small Born radius represents large screening (weak electric field) as if the atom were in bulk water. The next section describes how the Born radius is calculated and how it is used to calculate electrostatics.

#### 6.1.3 Generalized Born Equations

In a GB simulation, the total electrostatic force on an atom, $i$, is the net Coulomb force on atom $i$ (from nearby atoms) minus the GB force on atom $i$ (also caused by nearby atoms):

$$\vec{F}_i = \vec{F}_i^{Coulomb} - \vec{F}_i^{GB}.$$

Forces are contributed by other nearby atoms within a cutoff. The GB force on atom $i$ is the derivative of the total GB energy with respect to relative atom distances $r_{ij}$,

$$\begin{aligned}
\vec{F}_i^{GB} &= -\sum_j \left[\frac{dE_T^{GB}}{dr_{ij}}\right]\hat{r}_{ji} &(8)\\
&= -\sum_j \left[\sum_k \frac{\partial E_T^{GB}}{\partial \alpha_k}\frac{d\alpha_k}{dr_{ij}} + \frac{\partial E_{ij}^{GB}}{\partial r_{ij}}\right]\hat{r}_{ji} &(9)\\
&= -\sum_j \left[\frac{\partial E_T^{GB}}{\partial \alpha_i}\frac{d\alpha_i}{dr_{ij}} + \frac{\partial E_T^{GB}}{\partial \alpha_j}\frac{d\alpha_j}{dr_{ij}} + \frac{\partial E_{ij}^{GB}}{\partial r_{ij}}\right]\hat{r}_{ji}\ . &(10)
\end{aligned}$$

where the partial derivatives are included since the Born radius, $\alpha$, is a function of all relative atom distances. The total GB energy of the system is

$$E_T^{GB} = \sum_i \sum_{j>i} E_{ij}^{GB} + \sum_i E_{ii}^{GB}\ , \tag{11}$$

where $E_{ii}^{GB}$ is the Born radius dependent self energy of atom $i$, and the GB energy between atoms $i$ and $j$ is given by

$$E_{ij}^{GB} = -k_e D_{ij}\frac{q_i q_j}{f_{ij}}\ . \tag{12}$$

The dielectric term [73] is

$$D_{ij} = \left(\frac{1}{\epsilon_p} - \frac{\exp\left(-\kappa f_{ij}\right)}{\epsilon_s}\right)\ , \tag{13}$$

and the GB function [76] is

$$f_{ij} = \sqrt{r_{ij}^2 + \alpha_i\alpha_j \exp\left(\frac{-r_{ij}^2}{4\alpha_i\alpha_j}\right)}\ . \tag{14}$$

As the Born radii of atoms $i$ and $j$ decrease (increasing screening), the effective distance between the atoms ($f_{ij}$) increases. The implicit solvent implemented in NAMD is the model of Onufriev, Bashford and Case [56, 57] which calculates the Born radius as

$$\alpha_k = \left[\frac{1}{\rho_{k0}} - \frac{1}{\rho_k}\tanh\left(\delta\psi_k - \beta\psi_k^2 + \gamma\psi_k^3\right)\right]^{-1} \tag{15}$$

where

$$\psi_k = \rho_{k0}\sum_l H_{kl}\ . \tag{16}$$

$H_{ij}$ is the piecewise descreening function [57, 28, 65]; the seven piecewise regimes are

$$\text{Regimes} = \left\{\begin{array}{rll}
0 & r_{ij} > r_c + \rho_{js} & \text{(sphere } j \text{ beyond cutoff)}\\
\text{I} & r_{ij} > r_c - \rho_{js} & \text{(sphere } j \text{ partially within cutoff)}\\
\text{II} & r_{ij} > 4\rho_{js} & \text{(artificial regime for smoothing)}\\
\text{III} & r_{ij} > \rho_{i0} + \rho_{js} & \text{(spheres not overlapping)}\\
\text{IV} & r_{ij} > |\rho_{i0} - \rho_{js}| & \text{(spheres overlapping)}\\
\text{V} & \rho_{i0} < \rho_{js} & \text{(sphere } i \text{ inside sphere } j\text{)}\\
\text{VI} & \text{otherwise} & \text{(sphere } j \text{ inside sphere } j\text{)}
\end{array}\right. \tag{17}$$

and the values of $H_{ij}$ are

$$H_{ij} = \begin{cases} 0 & 0 \\ \text{I} & \frac{1}{8r_{ij}}\left[1+\frac{2r_{ij}}{r_{ij}-\rho_{js}}+\frac{1}{r_c^2}\left(r_{ij}^2-4r_cr_{ij}-\rho_{js}^2\right)+2\ln\frac{r_{ij}-\rho_{js}}{r_c}\right] \\ \text{II} & \frac{\rho_{js}^2}{r_{ij}^2}\frac{\rho_{js}}{r_{ij}^2}\left[a+\frac{\rho_{js}^2}{r_{ij}^2}\left(b+\frac{\rho_{js}^2}{r_{ij}^2}\left(c+\frac{\rho_{js}^2}{r_{ij}^2}\left(d+\frac{\rho_{js}^2}{r_{ij}^2}e\right)\right)\right)\right] \\ \text{III} & \frac{1}{2}\left[\frac{\rho_{js}}{r_{ij}^2-\rho_{js}^2}+\frac{1}{2r_{ij}}\ln\frac{r_{ij}-\rho_{js}}{r_{ij}+\rho_{js}}\right] \\ \text{IV} & \frac{1}{4}\left[\frac{1}{\rho_{i0}}\left(2-\frac{1}{2r_{ij}\rho_{i0}}\left(r_{ij}^2+\rho_{i0}^2-\rho_{js}^2\right)\right)-\frac{1}{r_{ij}+\rho_{js}}+\frac{1}{r_{ij}}\ln\frac{\rho_{i0}}{r_{ij}+\rho_{js}}\right] \\ \text{V} & \frac{1}{2}\left[\frac{\rho_{js}}{r_{ij}^2-\rho_{js}^2}+\frac{2}{\rho_{i0}}+\frac{1}{2r_{ij}}\ln\frac{\rho_{js}-r_{ij}}{r_{ij}+\rho_{js}}\right] \\ \text{VI} & 0 \end{cases} \tag{18}$$

Below are defined the derivatives of the above functions which are required for force calculations.

$$\frac{\partial E_{ij}}{\partial r_{ij}} = -k_e\left[\frac{q_iq_j}{f_{ij}}\frac{\partial D_{ij}}{\partial r_{ij}} - \frac{q_iq_jD_{ij}}{f_{ij}^2}\frac{\partial f_{ij}}{\partial r_{ij}}\right] \tag{19}$$

$$\frac{\partial D_{ij}}{\partial r_{ij}} = \frac{\kappa}{\epsilon_s}\exp\left(-\kappa f_{ij}\right)\frac{\partial f_{ij}}{\partial r_{ij}} \tag{20}$$

$$\frac{\partial f_{ij}}{\partial r_{ij}} = \frac{r_{ij}}{f_{ij}}\left[1-\frac{1}{4}\exp\left(\frac{-r_{ij}^2}{4\alpha_i\alpha_j}\right)\right] \tag{21}$$

$$\frac{d\alpha_k}{dr_{ij}} = \frac{\alpha_k^2}{\rho_k}\left(1-\tanh^2\left(\delta\psi_k-\beta\psi_k^2+\gamma\psi_k^3\right)\right)\left(\delta-2\beta\psi_k+3\beta\psi_k^2\right)\frac{d\psi_k}{dr_{ij}} \tag{22}$$

$$\begin{aligned} \frac{d\psi_k}{dr_{ij}} &= \rho_{k0}\sum_l \frac{dH_{kl}}{dr_{ij}} && (23) \\ &= \rho_{k0}\sum_l \frac{\partial H_{kl}}{\partial r_{kl}}\frac{dr_{kl}}{dr_{ij}} && (24) \\ &= \rho_{k0}\left[\frac{\partial H_{kj}}{\partial r_{kj}}\delta_{ki}+\frac{\partial H_{ki}}{\partial r_{ki}}\delta_{kj}\right] && (25) \end{aligned}$$

$$\begin{aligned} \frac{d\alpha_k}{dr_{ij}} = &\ \frac{\alpha_i^2\rho_{i0}}{\rho_i}\left(1-\tanh^2\left(\delta\psi_i-\beta\psi_i^2+\gamma\psi_i^3\right)\right)\left(\delta-2\beta\psi_i+3\beta\psi_i^2\right)\frac{\partial H_{ij}}{\partial r_{ij}}\delta_{ki} \\ + &\ \frac{\alpha_j^2\rho_{j0}}{\rho_j}\left(1-\tanh^2\left(\delta\psi_j-\beta\psi_j^2+\gamma\psi_j^3\right)\right)\left(\delta-2\beta\psi_j+3\beta\psi_j^2\right)\frac{\partial H_{ji}}{\partial r_{ij}}\delta_{kj} \end{aligned} \tag{26}$$

$$\frac{\partial E_{ij}}{\partial \alpha_i} = -\frac{1}{\alpha_i}\frac{k_eq_iq_j}{2f_{ij}^2}\left(\frac{\kappa}{\epsilon_s}\exp\left(-\kappa f_{ij}\right)-\frac{D_{ij}}{f_{ij}}\right)\left(\alpha_i\alpha_j+\frac{r_{ij}^2}{4}\right)\exp\left(\frac{-r_{ij}^2}{4\alpha_i\alpha_j}\right) \tag{27}$$

$$\frac{\partial E_{ij}}{\partial \alpha_j} = -\frac{1}{\alpha_j}\frac{k_eq_iq_j}{2f_{ij}^2}\left(\frac{\kappa}{\epsilon_s}\exp\left(-\kappa f_{ij}\right)-\frac{D_{ij}}{f_{ij}}\right)\left(\alpha_i\alpha_j+\frac{r_{ij}^2}{4}\right)\exp\left(\frac{-r_{ij}^2}{4\alpha_i\alpha_j}\right) \tag{28}$$

$$\frac{\partial H_{ij}}{\partial r_{ij}} = \begin{cases} 0 & 0 \\ \mathrm{I} & \left[-\frac{(r_c+\rho_{js}-r_{ij})(r_c-\rho_{js}+r_{ij})\left(\rho_{js}^2+r_{ij}^2\right)}{8r_c^2 r_{ij}^2(\rho_{js}-r_{ij})^2} - \frac{1}{4r_{ij}^2}\ln\frac{r_{ij}-\rho_{js}}{r_c}\right] \\ \mathrm{II} & \left[-4a\frac{\rho_{js}^3}{r_{ij}^5} - 6b\frac{\rho_{js}^5}{r_{ij}^7} - 8c\frac{\rho_{js}^7}{r_{ij}^9} - 10d\frac{\rho_{js}^9}{r_{ij}^{11}} - 12e\frac{\rho_{js}^{11}}{r_{ij}^{13}}\right] \\ \mathrm{III} & \frac{1}{2}\left[-\frac{\rho_{js}\left(r_{ij}^2+\rho_{js}^2\right)}{r_{ij}\left(r_{ij}^2-\rho_{js}^2\right)^2} - \frac{1}{2r_{ij}^2}\ln\frac{r_{ij}-\rho_{js}}{r_{ij}+\rho_{js}}\right] \\ \mathrm{IV} & \frac{1}{4}\left[-\frac{1}{2\rho_{i0}^2} + \frac{r_{ij}^2\left(\rho_{i0}^2-\rho_{js}^2\right)-2r_{ij}\rho_{js}^3+\rho_{js}^2\left(\rho_{i0}^2-\rho_{js}^2\right)}{2r_{ij}^2\rho_{i0}^2(r_{ij}+\rho_{js})^2} - \frac{1}{r_{ij}^2}\ln\frac{\rho_{i0}}{r_{ij}+\rho_{js}}\right] \\ \mathrm{V} & \frac{1}{2}\left[-\frac{\rho_{js}\left(r_{ij}^2+\rho_{js}^2\right)}{r_{ij}\left(r_{ij}^2-\rho_{js}^2\right)^2} - \frac{1}{2r_{ij}^2}\ln\frac{\rho_{js}-r_{ij}}{r_{ij}+\rho_{js}}\right] \\ \mathrm{VI} & 0 \end{cases} \tag{29}$$

Other variables referenced in the above GB equations are

- $r_{ij}$ - distance between atoms i and j; calculated from atom coordinates.
- $\kappa$ - debye screening length; calculated from ion concentration, $\kappa^{-1} = \sqrt{\frac{\epsilon_0 \epsilon_p kT}{2N_A e^2 I}}$; $\kappa^{-1} = 10$ Å for 0.1 M monovalent salt.
- $\epsilon_s$ - dielectric constant of solvent.
- $\epsilon_p$ - dielectric constant of protein.
- $\alpha_i$ - Born radius of atom $i$.
- $\rho_i$ - intrinsic radius of atom $i$ taken from Bondi [9].
- $\rho_0$ - intrinsic radius offset; $\rho_0 = 0.09$ Å by default [57].
- $\rho_{i0} = \rho_i - \rho_0$
- $\rho_{is} = \rho_{i0} S_{ij}$
- $S_{ij}$ - atom radius scaling factor [28, 73].
- $k_e$ - Coulomb's constant, $\frac{1}{4\pi\epsilon_0}$, 332.063711 kcal Å / e$^2$.
- $\{\delta, \beta, \gamma\} = \{0.8, 0, 2.91\}$ or $\{1.0, 0.8, 4.85\}$ [57]

## 6.2 3-Phase Calculation

The GBIS algorithm requires three phases of calculation, with each phase containing an iteration over all atom pairs with the cutoff. In phase 1, the screening of atom pairs is summed; at the conclusion of phase 1, the Born radii are calculated. In phase 2, the $\frac{\partial E_{ij}^{GB}}{\partial r_{ij}}$ force contribution (hereafter called the dEdr force) is calculated as well as the partial derivative of the Born radii with respect to the atom positions, $\frac{d\alpha_i}{dr_{ij}}$. In phase 3, the $\frac{\partial E_T^{GB}}{\partial \alpha_i}\frac{d\alpha_i}{dr_{ij}}$ force contribution (hereafter called the dEda force) is calculated.

## 6.3 Configuration Parameters

When using GBIS, user's should not use `PME` (because it is not compatible with GBIS). Periodic boundary conditions are supported but are optional. User's will need to increase `cutoff`; 16-18 Å is a good place to start but user's will have to check their system's behavior and increase `cutoff` accordingly. GBIS interactions are never excluded regardless of the type of force field used, thus user's can choose any value for `exclude` without affecting GBIS; user's should still choose `exclude` based on the force field as if using explicit solvent. When using GBIS, multiple timestepping behaves as follows: the dEdr force is calculated every `nonbondedFreq` steps (as with explicit solvent, 2 is a reasonable frequency) and the dEda force is calculated every `fullElectFrequency` steps (because dEda varies more slowly than dEdr, 4 is a reasonable frequency).

- `GBIS` < Use Generalized Born Implicit Solvent? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Turns on GBIS method in NAMD.

- `solventDielectric` < dielectric of water >
  **Acceptable Values:** positive decimal
  **Default Value:** 78.5
  **Description:** Defines the dielectric of the solvent, usually 78.5 or 80.

- `intrinsicRadiusOffset` < shrink the intrinsic radius of atoms (Å) >
  **Acceptable Values:** positive decimal
  **Default Value:** 0.09
  **Description:** This offset shrinks the intrinsic radius of atoms (used only for calculating Born radius) to give better agreement with Poisson Boltzmann calculations. Most users should not change this parameter.

- `ionConcentration` < concentration of ions in solvent (Molar) >
  **Acceptable Values:** positive decimal
  **Default Value:** 0.2
  **Description:** An ion concentration of 0 M represents distilled water. Increasing the ion concentration increases the electrostatic screening.

- `GBISDelta` < GB$^{OBC}$ parameter for calculating Born radii >
  **Acceptable Values:** decimal
  **Default Value:** 1.0
  **Description:** Use $\{\texttt{GBISDelta}, \texttt{GBISBeta}, \texttt{GBISGamma}\} = \{1.0, 0.8, 4.85\}$ for GB$^{OBC}$II and $\{0.8, 0.0, 2.90912\}$ for GB$^{OBC}$I. See $\{\alpha, \beta, \gamma\}$ in [57] for more information.

- `GBISBeta` < GB$^{OBC}$ parameter for calculating Born radii >
  **Acceptable Values:** decimal
  **Default Value:** 0.8
  **Description:** See `GBISDelta`.

- `GBISGamma` < GB$^{OBC}$ parameter for calculating Born radii >
  **Acceptable Values:** decimal
  **Default Value:** 4.85
  **Description:** See `GBISDelta`.

- `alphaCutoff` < cutoff used in calculating Born radius and derivatives (phases 1 and 3) (Å) >
  **Acceptable Values:** positive decimal
  **Default Value:** 15
  **Description:** Cutoff used for calculating Born radius. Only atoms within this cutoff de-screen an atom. Though `alphaCutoff` can bet set to be larger or shorter than `cutoff`, since atom forces are more sensitive to changes in position than changes in Born radius, user's should generally set `alphaCutoff` to be shorter than `cutoff`.

- `SASA` < whether or not to calculate SASA >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** The nonpolar / hydrophobic energy contribution from implicit solvent is calculated; it is proportional to the solvent-accessible surface area (SASA) which is calculated by the Linear Combination of Pairwise Overlap (LCPO) method [85]. It evaluated every `nonbondedFreq` steps and its energy is added into the reported `ELECT` energy.

- `surfaceTension` < surface tension of SASA energy >
  **Acceptable Values:** positive decimal
  **Default Value:** 0.005 kcal/mol/Å$^2$
  **Description:** Surface tension used when calculating hydrophobic `SASA` energy; $E_{\text{nonpolar}} = \text{surfaceTension} \times \text{surfaceArea}$.

Below is a sample excerpt from a NAMD config file for nonbonded and multistep parameters when using `GBIS` and `SASA`:

```
#GBIS parameters
GBIS on
ionConcentration 0.3
alphaCutoff 14
#nonbonded parameters
switching on
switchdist 15
cutoff 16
pairlistdist 18
#hydrophobic energy
sasa on
surfaceTension 0.006
#multistep parameters
timestep 1
nonbondedFreq 2
fullElectFrequency 4
```

# 7 Standard Minimization and Dynamics Parameters

## 7.1 Boundary Conditions

In addition to periodic boundary conditions, NAMD provides spherical and cylindrical boundary potentials to contain atoms in a given volume. To apply more general boundary potentials written in Tcl, use `tclBC` as described in Sec. 8.11.

### 7.1.1 Periodic boundary conditions

NAMD provides periodic boundary conditions in 1, 2 or 3 dimensions. The following parameters are used to define these boundary conditions.

- `cellBasisVector1` < basis vector for periodic boundaries (Å) >
  **Acceptable Values:** vector
  **Default Value:** 0 0 0
  **Description:** Specifies a basis vector for periodic boundary conditions.

- `cellBasisVector2` < basis vector for periodic boundaries (Å) >
  **Acceptable Values:** vector
  **Default Value:** 0 0 0
  **Description:** Specifies a basis vector for periodic boundary conditions.

- `cellBasisVector3` < basis vector for periodic boundaries (Å) >
  **Acceptable Values:** vector
  **Default Value:** 0 0 0
  **Description:** Specifies a basis vector for periodic boundary conditions.

- `cellOrigin` < center of periodic cell (Å) >
  **Acceptable Values:** position
  **Default Value:** 0 0 0
  **Description:** When position rescaling is used to control pressure, this location will remain constant. Also used as the center of the cell for wrapped output coordinates.

- `extendedSystem` < XSC file to read cell parameters from >
  **Acceptable Values:** file name
  **Description:** In addition to .coor and .vel output files, NAMD generates a .xsc (eXtended System Configuration) file which contains the periodic cell parameters and extended system variables, such as the strain rate in constant pressure simulations. Periodic cell parameters will be read from this file if this option is present, ignoring the above parameters.

- `XSTfile` < XST file to write cell trajectory to >
  **Acceptable Values:** file name
  **Default Value:** *outputname*`.xst`
  **Description:** NAMD can also generate a .xst (eXtended System Trajectory) file which contains a record of the periodic cell parameters and extended system variables during the simulation. If `XSTfile` is defined, then `XSTfreq` must also be defined.

- `XSTfreq` < how often to append state to XST file >
  **Acceptable Values:** positive integer

**Description:** Like the `DCDfreq` option, controls how often the extended system configuration will be appended to the XST file.

- `wrapAll` < wrap all coordinates around periodic boundaries? >
  **Acceptable Values:** on or off
  **Default Value:** off
  **Description:** Coordinates are normally output relative to the way they were read in. Hence, if part of a molecule crosses a periodic boundary it is not translated to the other side of the cell on output. This option applies a translation to the center-of-mass of each molecule or contiguous cluster of bonded atoms to keep it within the periodic unit cell. The translation has usually *no effect on the physical trajectory*, because the force field potentials used in NAMD follow the minimum-image convention for interatomic distances. However, some complex quantities, for example the center of mass of a multimeric protein, will be undefined as a result of this option. If you plan on applying external forces (`SMD`, `tclForces` or `Colvars`) to such quantities, it is recommended to keep this option off, and to possibly replace it with a custom restraint.

- `wrapWater` < wrap water coordinates around periodic boundaries? >
  **Acceptable Values:** on or off
  **Default Value:** off
  **Description:** This option is similar to the `wrapAll` option, but its effect is restricted to water molecules only.

- `wrapNearest` < use nearest image to cell origin when wrapping coordinates? >
  **Acceptable Values:** on or off
  **Default Value:** off
  **Description:** Coordinates are normally wrapped to the diagonal unit cell centered on the origin. This option, combined with `wrapWater` or `wrapAll`, wraps coordinates to the nearest image to the origin, providing hexagonal or other cell shapes.

### 7.1.2 Spherical harmonic boundary conditions

NAMD provides spherical harmonic boundary conditions. These boundary conditions can consist of a single potential or a combination of two potentials. The following parameters are used to define these boundary conditions.

- `sphericalBC` < use spherical boundary conditions? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not spherical boundary conditions are to be applied to the system. If set to `on`, then `sphericalBCCenter`, `sphericalBCr1` and `sphericalBCk1` must be defined, and `sphericalBCexp1`, `sphericalBCr2`, `sphericalBCk2`, and `sphericalBCexp2` can optionally be defined.

- `sphericalBCCenter` < center of sphere (Å) >
  **Acceptable Values:** position
  **Description:** Location around which sphere is centered.

- `sphericalBCr1` < radius for first boundary condition (Å) >
  **Acceptable Values:** positive decimal
  **Description:** Distance at which the first potential of the boundary conditions takes effect. This distance is a radius from the center.

- `sphericalBCk1` < force constant for first potential >
  **Acceptable Values:** non-zero decimal
  **Description:** Force constant for the first harmonic potential. A positive value will push atoms toward the center, and a negative value will pull atoms away from the center.

- `sphericalBCexp1` < exponent for first potential >
  **Acceptable Values:** positive, even integer
  **Default Value:** 2
  **Description:** Exponent for first boundary potential. The only likely values to use are 2 and 4.

- `sphericalBCr2` < radius for second boundary condition (Å) >
  **Acceptable Values:** positive decimal
  **Description:** Distance at which the second potential of the boundary conditions takes effect. This distance is a radius from the center. If this parameter is defined, then `spericalBCk2` must also be defined.

- `sphericalBCk2` < force constant for second potential >
  **Acceptable Values:** non-zero decimal
  **Description:** Force constant for the second harmonic potential. A positive value will push atoms toward the center, and a negative value will pull atoms away from the center.

- `sphericalBCexp2` < exponent for second potential >
  **Acceptable Values:** positive, even integer
  **Default Value:** 2
  **Description:** Exponent for second boundary potential. The only likely values to use are 2 and 4.

### 7.1.3 Cylindrical harmonic boundary conditions

NAMD provides cylindrical harmonic boundary conditions. These boundary conditions can consist of a single potential or a combination of two potentials. The following parameters are used to define these boundary conditions.

- `cylindricalBC` < use cylindrical boundary conditions? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not cylindrical boundary conditions are to be applied to the system. If set to `on`, then `cylindricalBCCenter`, `cylindricalBCr1`, `cylindricalBCl1` and `cylindricalBCk1` must be defined, and `cylindricalBCAxis`, `cylindricalBCexp1`, `cylindricalBCr2`, `cylindricalBCl2`, `cylindricalBCk2`, and `cylindricalBCexp2` can optionally be defined.

- `cylindricalBCCenter` < center of cylinder (Å) >
  **Acceptable Values:** position
  **Description:** Location around which cylinder is centered.

- `cylindricalBCAxis` < axis of cylinder (Å) >
  **Acceptable Values:** `x`, `y`, or `z`
  **Description:** Axis along which cylinder is aligned.

- `cylindricalBCr1` < radius for first boundary condition (Å) >
  **Acceptable Values:** positive decimal
  **Description:** Distance at which the first potential of the boundary conditions takes effect along the non-axis plane of the cylinder.

- `cylindricalBCl1` < distance along cylinder axis for first boundary condition (Å) >
  **Acceptable Values:** positive decimal
  **Description:** Distance at which the first potential of the boundary conditions takes effect along the cylinder axis.

- `cylindricalBCk1` < force constant for first potential >
  **Acceptable Values:** non-zero decimal
  **Description:** Force constant for the first harmonic potential. A positive value will push atoms toward the center, and a negative value will pull atoms away from the center.

- `cylindricalBCexp1` < exponent for first potential >
  **Acceptable Values:** positive, even integer
  **Default Value:** 2
  **Description:** Exponent for first boundary potential. The only likely values to use are 2 and 4.

- `cylindricalBCr2` < radius for second boundary condition (Å) >
  **Acceptable Values:** positive decimal
  **Description:** Distance at which the second potential of the boundary conditions takes effect along the non-axis plane of the cylinder. If this parameter is defined, then `cylindricalBCl2` and `spericalBCk2` must also be defined.

- `cylindricalBCl2` < radius for second boundary condition (Å) >
  **Acceptable Values:** positive decimal
  **Description:** Distance at which the second potential of the boundary conditions takes effect along the cylinder axis. If this parameter is defined, then `cylindricalBCr2` and `spericalBCk2` must also be defined.

- `cylindricalBCk2` < force constant for second potential >
  **Acceptable Values:** non-zero decimal
  **Description:** Force constant for the second harmonic potential. A positive value will push atoms toward the center, and a negative value will pull atoms away from the center.

- `cylindricalBCexp2` < exponent for second potential >
  **Acceptable Values:** positive, even integer
  **Default Value:** 2
  **Description:** Exponent for second boundary potential. The only likely values to use are 2 and 4.

## 7.2 Energy Minimization

### 7.2.1 Conjugate gradient parameters

The default minimizer uses a sophisticated conjugate gradient and line search algorithm with much better performance than the older velocity quenching method. The method of conjugate gradients is used to select successive search directions (starting with the initial gradient) which eliminate repeated minimization along the same directions. Along each direction, a minimum is first bracketed (rigorously bounded) and then converged upon by either a golden section search, or, when possible, a quadratically convergent method using gradient information.

For most systems, it just works.

- `minimization` < Perform conjugate gradient energy minimization? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Turns efficient energy minimization `on` or `off`.

- `minTinyStep` < first initial step for line minimizer >
  **Acceptable Values:** positive decimal
  **Default Value:** 1.0e-6
  **Description:** If your minimization is immediately unstable, make this smaller.

- `minBabyStep` < max initial step for line minimizer >
  **Acceptable Values:** positive decimal
  **Default Value:** 1.0e-2
  **Description:** If your minimization becomes unstable later, make this smaller.

- `minLineGoal` < gradient reduction factor for line minimizer >
  **Acceptable Values:** positive decimal
  **Default Value:** 1.0e-4
  **Description:** Varying this might improve conjugate gradient performance.

### 7.2.2 Velocity quenching parameters

You can perform energy minimization using a simple quenching scheme. While this algorithm is not the most rapidly convergent, it is sufficient for most applications. There are only two parameters for minimization: one to activate minimization and another to specify the maximum movement of any atom.

- `velocityQuenching` < Perform old-style energy minimization? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Turns slow energy minimization `on` or `off`.

- `maximumMove` < maximum distance an atom can move during each step (Å) >
  **Acceptable Values:** positive decimal
  **Default Value:** $0.75 \times$ `cutoff/stepsPerCycle`
  **Description:** Maximum distance that an atom can move during any single timestep of minimization. This is to insure that atoms do not go flying off into space during the first few timesteps when the largest energy conflicts are resolved.

## 7.3 Dynamics

### 7.3.1 Timestep parameters

- `numsteps` < number of timesteps >
  **Acceptable Values:** positive integer
  **Description:** The number of simulation timesteps to be performed. An integer greater than 0 is acceptable. The total amount of simulation time is `numsteps` × `timestep`.

- `timestep` < timestep size (fs) >
  **Acceptable Values:** non-negative decimal
  **Default Value:** 1.0
  **Description:** The timestep size to use when integrating each step of the simulation. The value is specified in femtoseconds.

- `firsttimestep` < starting timestep value >
  **Acceptable Values:** non-negative integer
  **Default Value:** 0
  **Description:** The number of the first timestep. This value is typically used only when a simulation is a continuation of a previous simulation. In this case, rather than having the timestep restart at 0, a specific timestep number can be specified.

### 7.3.2 Initialization

- `temperature` < initial temperature (K) >
  **Acceptable Values:** positive decimal
  **Description:** Initial temperature value for the system. Using this option will generate a random velocity distribution for the initial velocities for all the atoms such that the system is at the desired temperature. Either the `temperature` or the `velocities/binvelocities` option must be defined to determine an initial set of velocities. Both options cannot be used together.

- `COMmotion` < allow initial center of mass motion? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Specifies whether or not motion of the center of mass of the entire system is allowed. If this option is set to `no`, the initial velocities of the system will be adjusted to remove center of mass motion of the system. Note that this does not preclude later center-of-mass motion due to external forces such as random noise in Langevin dynamics, boundary potentials, and harmonic restraints.

- `seed` < random number seed >
  **Acceptable Values:** positive integer
  **Default Value:** pseudo-random value based on current UNIX clock time
  **Description:** Number used to seed the random number generator if `temperature` or `langevin` is selected. This can be used so that consecutive simulations produce the same results. If no value is specified, NAMD will choose a pseudo-random value based on the current UNIX clock time. The random number seed will be output during the simulation startup so that its value is known and can be reused for subsequent simulations. Note that if

Langevin dynamics are used in a parallel simulation (i.e., a simulation using more than one processor) even using the same seed will *not* guarantee reproducible results.

### 7.3.3 Conserving momentum

- `zeroMomentum` < remove center of mass drift due to PME >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** If enabled, the net momentum of the simulation and any resultant drift is removed before every full electrostatics step. This correction should conserve energy and have minimal impact on parallel scaling. This feature should only be used for simulations that would conserve momentum except for the slight errors in PME. (Features such as fixed atoms, harmonic restraints, steering forces, and Langevin dynamics do not conserve momentum; use in combination with these features should be considered experimental.) Since the momentum correction is delayed, enabling outputMomenta will show a slight nonzero linear momentum but there should be no center of mass drift.

### 7.3.4 Multiple timestep parameters

To further reduce the cost of computing full electrostatics, NAMD uses a multiple timestepping integration scheme. In this scheme, the total force acting on each atom is broken into two pieces, a quickly varying local component and a slower long range component. The local force component is defined in terms of a *splitting function*. The local force component consists of all bonded and van der Waals interactions as well as that portion of electrostatic interactions for pairs that are separated by less than the local interaction distance determined by the splitting function. The long range component consists only of electrostatic interactions outside of the local interaction distance. Since the long range forces are slowly varying, they are not evaluated every timestep. Instead, they are evaluated every $k$ timesteps, specified by the NAMD parameter `fullElectFrequency`. An impulse of $k$ times the long range force is applied to the system every $k$ timesteps (i.e., the r-RESPA integrator is used). For appropriate values of $k$, it is believed that the error introduced by this infrequent evaluation is modest compared to the error already incurred by the use of the numerical (Verlet) integrator. Improved methods for incorporating these long range forces are currently being investigated, with the intention of improving accuracy as well as reducing the frequency of long range force evaluations.

In the scheme described above, the van der Waals forces are still truncated at the local interaction distance. Thus, the van der Waals cutoff distance forms a lower limit to the local interaction distance. While this is believed to be sufficient, there are investigations underway to remove this limitation and provide full van der Waals calculations in $\mathcal{O}(N)$ time as well.

One of the areas of current research being studied using NAMD is the exploration of better methods for performing multiple timestep integration. Currently the only available method is the impulse-based Verlet-I or r-RESPA method which is stable for timesteps up to 4 fs for long-range electrostatic forces, 2 fs for short-range nonbonded forces, and 1 fs for bonded forces Setting `rigid all` (i.e., using SHAKE) increases these timesteps to 6 fs, 2 fs, and 2 fs respectively but eliminates bond motion for hydrogen. The mollified impulse method (MOLLY) reduces the resonance which limits the timesteps and thus increases these timesteps to 6 fs, 2 fs, and 1 fs while retaining all bond motion.

- `fullElectFrequency` < number of timesteps between full electrostatic evaluations >
  **Acceptable Values:** positive integer factor of `stepspercycle`
  **Default Value:** `nonbondedFreq`
  **Description:** This parameter specifies the number of timesteps between each full electrostatics evaluation. It is recommended that `fullElectFrequency` be chosen so that the product of `fullElectFrequency` and `timestep` does not exceed 4.0 unless `rigidBonds all` or `molly on` is specified, in which case the upper limit is perhaps doubled.

- `nonbondedFreq` < timesteps between nonbonded evaluation >
  **Acceptable Values:** positive integer factor of `fullElectFrequency`
  **Default Value:** 1
  **Description:** This parameter specifies how often short-range nonbonded interactions should be calculated. Setting `nonbondedFreq` between 1 and `fullElectFrequency` allows triple timestepping where, for example, one could evaluate bonded forces every 1 fs, short-range nonbonded forces every 2 fs, and long-range electrostatics every 4 fs.

- `MTSAlgorithm` < MTS algorithm to be used >
  **Acceptable Values:** `impulse/verletI` or `constant/naive`
  **Default Value:** `impulse`
  **Description:** Specifies the multiple timestep algorithm used to integrate the long and short range forces. `impulse/verletI` is the same as r-RESPA. `constant/naive` is the stale force extrapolation method.

- `longSplitting` < how should long and short range forces be split? >
  **Acceptable Values:** `c1`, `c2`
  **Default Value:** `c1`
  **Description:** Specifies the method used to split electrostatic forces between long and short range potentials. The `c1` option uses a cubic polynomial splitting function,

$$S_3(r) = 1 - \frac{3}{2}\left(\frac{r}{r_{\text{cut}}}\right) + \frac{1}{2}\left(\frac{r}{r_{\text{cut}}}\right)^3,$$

  to affect $C^1$ continuity in the splitting of the electrostatic potential [72]. The `c2` option uses a quintic polynomial splitting function,

$$S_5(r) = 1 - 10\left(\frac{r}{r_{\text{cut}}}\right)^3 + 15\left(\frac{r}{r_{\text{cut}}}\right)^4 - 6\left(\frac{r}{r_{\text{cut}}}\right)^5,$$

  to affect $C^2$ continuity in the splitting of the electrostatic potential. The $S_5$ splitting function, contributed by Bruce Berne, Ruhong Zhou, and Joe Morrone, produces demonstrably better long time stability than $S_3$ without requiring any additional computational cost during simulation, since the nonbonded forces are calculated via a lookup table. Note that earlier options `xplor` and `sharp` are no longer supported.

- `molly` < use mollified impulse method (MOLLY)? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** This method eliminates the components of the long range electrostatic forces which contribute to resonance along bonds to hydrogen atoms, allowing a fullElectFrequency

of 6 (vs. 4) with a 1 fs timestep without using `rigidBonds all`. You may use `rigidBonds water` but using `rigidBonds all` with MOLLY makes no sense since the degrees of freedom which MOLLY protects from resonance are already frozen.

- `mollyTolerance` < allowable error for MOLLY >
  **Acceptable Values:** positive decimal
  **Default Value:** 0.00001
  **Description:** Convergence criterion for MOLLY algorithm.

- `mollyIterations` < maximum MOLLY iterations >
  **Acceptable Values:** positive integer
  **Default Value:** 100
  **Description:** Maximum number of iterations for MOLLY algorithm.

## 7.4 Temperature Control and Equilibration

### 7.4.1 Langevin dynamics parameters

NAMD is capable of performing Langevin dynamics, where additional damping and random forces are introduced to the system. This capability is based on that implemented in X-PLOR which is detailed in the X-PLOR *User's Manual* [12], although a different integrator is used.

- `langevin` < use Langevin dynamics? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not Langevin dynamics active. If set to `on`, then the parameter `langevinTemp` must be set and the parameters `langevinFile` and `langevinCol` can optionally be set to control the behavior of this feature.

- `langevinTemp` < temperature for Langevin calculations (K) >
  **Acceptable Values:** positive decimal
  **Description:** Temperature to which atoms affected by Langevin dynamics will be adjusted. This temperature will be roughly maintained across the affected atoms through the addition of friction and random forces.

- `langevinDamping` < damping coefficient for Langevin dynamics (1/ps) >
  **Acceptable Values:** positive decimal
  **Default Value:** per-atom values from PDB file
  **Description:** Langevin coupling coefficient to be applied to all atoms (unless `langevinHydrogen` is `off`, in which case only non-hydrogen atoms are affected). If not given, a PDB file is used to obtain coefficients for each atom (see `langevinFile` and `langevinCol` below).

- `langevinHydrogen` < Apply Langevin dynamics to hydrogen atoms? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `on`
  **Description:** If `langevinDamping` is set then setting `langevinHydrogen` to `off` will turn off Langevin dynamics for hydrogen atoms. This parameter has no effect if Langevin coupling coefficients are read from a PDB file.

- `langevinFile` < PDB file containing Langevin parameters >
  **Acceptable Values:** UNIX filename
  **Default Value:** `coordinates`
  **Description:** PDB file to use for the Langevin coupling coefficients for each atom. If this parameter is not specified, then the PDB file specified by `coordinates` is used.

- `langevinCol` < column of PDB from which to read coefficients >
  **Acceptable Values:** `X`, `Y`, `Z`, `O`, or `B`
  **Default Value:** `O`
  **Description:** Column of the PDB file to use for the Langevin coupling coefficients for each atom. The coefficients can be read from any floating point column of the PDB file. A value of 0 indicates that the atom will remain unaffected.

### 7.4.2 Temperature coupling parameters

NAMD is capable of performing temperature coupling, in which forces are added or reduced to simulate the coupling of the system to a heat bath of a specified temperature. This capability is based on that implemented in X-PLOR which is detailed in the X-PLOR *User's Manual* [12].

- `tCouple` < perform temperature coupling? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not temperature coupling is active. If set to `on`, then the parameter `tCoupleTemp` must be set and the parameters `tCoupleFile` and `tCoupleCol` can optionally be set to control the behavior of this feature.

- `tCoupleTemp` < temperature for heat bath (K) >
  **Acceptable Values:** positive decimal
  **Description:** Temperature to which atoms affected by temperature coupling will be adjusted. This temperature will be roughly maintained across the affected atoms through the addition of forces.

- `tCoupleFile` < PDB file with tCouple parameters >
  **Acceptable Values:** UNIX filename
  **Default Value:** `coordinates`
  **Description:** PDB file to use for the temperature coupling coefficient for each atom. If this parameter is not specified, then the PDB file specified by `coordinates` is used.

- `tCoupleCol` < column of PDB from which to read coefficients >
  **Acceptable Values:** `X`, `Y`, `Z`, `O`, or `B`
  **Default Value:** `O`
  **Description:** Column of the PDB file to use for the temperature coupling coefficient for each atom. This value can be read from any floating point column of the PDB file. A value of 0 indicates that the atom will remain unaffected.

### 7.4.3 Stochastic velocity rescaling parameters

The stochastic velocity rescaling method originated by [13] can be viewed as an extension (and correction) of the Berendsen method. The implementation in NAMD is based on that from GROMACS, with some slight performance modifications during random number generation.

- `stochRescale` < perform stochastic rescaling? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not stochastic rescaling is active. If set to `on`, then the parameters `stochRescaleTemp` and `stochRescalePeriod` must be set.

- `stochRescaleTemp` < temperature for heat bath (K) >
  **Acceptable Values:** positive decimal
  **Description:** Temperature to which all atoms will be periodically readjusted toward. This temperature will be correctly maintained (in the canonical sense) over all atoms by rescaling the velocities with both deterministic (using the instantaneous temperature) and stochastic components.

- `stochRescalePeriod` < time parameter (ps) for temperature coupling >
  **Acceptable Values:** positive decimal
  **Description:** The stochastic rescaling algorithm holds for an arbitrary time parameter introduced when solving the Fokker-Planck equation. For systems predominantly composed of liquid water a value near 2 ps is appropriate and values between 0.5 and 2 ps are common in the literature for simulations of biomolecules. Larger values will generally result in weaker coupling and thus more NVE-like dynamics, but may also lead to slow (i.e. incorrect) convergence to the correct ensemble.

- `stochRescaleFreq` < number of timesteps between rescalings >
  **Acceptable Values:** positive integer
  **Default Value:** `stepsPerCycle`
  **Description:** The stochastic rescaling algorithm is invoked at fixed intervals. The effective time parameter is technically the ratio `stochRescaleFreq/stochRescalePeriod` (after converting into proper units using the value of `timestep`). The default should be adequate for most applications, but a smaller value closer to the frequency at which the nonbonded list is rebuilt would also be appropriate. When using multiple time stepping, it is important that rescaling occurs at timesteps that are integer multiples of the slowest interaction type (usually `fullElectFrequency`).

- `stochRescaleHeat` < Should heat transfer and work be computed? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** When active, the *cumulative* heat transfer with the thermostat will be reported as `HEAT`. The work due to the thermostat and integrator can then be computed as the change in total energy less the heat transfer and is reported as `WORK`. Note that the work includes all sources, including non-conservative elements of the Hamiltonian, but should otherwise approach zero for simulations at or near equilibrium. The accumulation starts at `firstTimestep` and can be reset from `Tcl` by re-setting this to zero. **This is an experimental option and not yet guaranteed for any specific purpose.**

### 7.4.4 Temperature rescaling parameters

NAMD allows equilibration of a system by means of temperature rescaling. Using this method, all of the velocities in the system are periodically rescaled so that the entire system is set to the desired temperature. The following parameters specify how often and to what temperature this rescaling is performed.

- `rescaleFreq` < number of timesteps between temperature rescaling >
  **Acceptable Values:** positive integer
  **Description:** The equilibration feature of NAMD is activated by specifying the number of timesteps between each temperature rescaling. If this value is given, then the `rescaleTemp` parameter must also be given to specify the target temperature.

- `rescaleTemp` < temperature for equilibration (K) >
  **Acceptable Values:** positive decimal
  **Description:** The temperature to which all velocities will be rescaled every `rescaleFreq` timesteps. This parameter is valid only if `rescaleFreq` has been set.

### 7.4.5 Temperature reassignment parameters

NAMD allows equilibration of a system by means of temperature reassignment. Using this method, all of the velocities in the system are periodically reassigned so that the entire system is set to the desired temperature. The following parameters specify how often and to what temperature this reassignment is performed.

- `reassignFreq` < number of timesteps between temperature reassignment >
  **Acceptable Values:** positive integer
  **Description:** The equilibration feature of NAMD is activated by specifying the number of timesteps between each temperature reassignment. If this value is given, then the `reassignTemp` parameter must also be given to specify the target temperature.

- `reassignTemp` < temperature for equilibration (K) >
  **Acceptable Values:** positive decimal
  **Default Value:** `temperature` if set, otherwise none
  **Description:** The temperature to which all velocities will be reassigned every `reassignFreq` timesteps. This parameter is valid only if `reassignFreq` has been set.

- `reassignIncr` < temperature increment for equilibration (K) >
  **Acceptable Values:** decimal
  **Default Value:** 0
  **Description:** In order to allow simulated annealing or other slow heating/cooling protocols, `reassignIncr` will be added to `reassignTemp` after each reassignment. (Reassignment is carried out at the first timestep.) The `reassignHold` parameter may be set to limit the final temperature. This parameter is valid only if `reassignFreq` has been set.

- `reassignHold` < holding temperature for equilibration (K) >
  **Acceptable Values:** positive decimal
  **Description:** The final temperature for reassignment when `reassignIncr` is set; `reassignTemp` will be held at this value once it has been reached. This parameter is valid only if `reassignIncr` has been set.

### 7.4.6 Lowe-Andersen dynamics parameters

NAMD can perform Lowe-Andersen dynamics, a variation of Andersen dynamics whereby the radial relative velocities of atom pairs are randomly modified based on a thermal distribution. The Lowe-Andersen thermostat is Galilean invariant, therefore conserving momentum, and is thus independent of absolute atom velocities. Forces are applied only between non-bonded, non-hydrogen pairs of atoms. When using rigid bonds, forces are applied to the center of mass of hydrogen groups. The implementation is based on Koopman and Lowe [37].

- `loweAndersen` < use Lowe-Andersen dynamics? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not Lowe-Andersen dynamics are active. If set to `on`, then the parameter `loweAndersenTemp` must be set and the parameters `loweAndersenCutoff` and `loweAndersenRate` can optionally be set.

- `loweAndersenTemp` < temperature for Lowe-Andersen calculations (K) >
  **Acceptable Values:** positive decimal
  **Description:** Temperature of the distribution used to set radial relative velocities. This determines the target temperature of the system.

- `loweAndersenCutoff` < cutoff radius for Lowe-Andersen collisions (Å) >
  **Acceptable Values:** positive decimal
  **Default Value:** 2.7
  **Description:** Forces are only applied to atoms within this distance of one another.

- `loweAndersenRate` < rate for Lowe-Andersen collisions (1/ps) >
  **Acceptable Values:** positive decimal
  **Default Value:** 50
  **Description:** Determines the probability of a collision between atoms within the cutoff radius. The probability is the rate specified by this keyword times the non-bonded timestep.

## 7.5 Pressure Control

Constant pressure simulation (and pressure calculation) require periodic boundary conditions. Pressure is controlled by dynamically adjusting the size of the unit cell and rescaling all atomic coordinates (other than those of fixed atoms) during the simulation.

Pressure values in NAMD output are in bar. PRESSURE is the pressure calculated based on individual atoms, while GPRESSURE incorporates hydrogen atoms into the heavier atoms to which they are bonded, producing smaller fluctuations. The TEMPAVG, PRESSAVG, and GPRESSAVG are the average of temperature and pressure values since the previous ENERGY output; for the first step in the simulation they will be identical to TEMP, PRESSURE, and GPRESSURE.

The phenomenological pressure of bulk matter reflects averaging in both space and time of the sum of a large positive term (the kinetic pressure, $nRT/V$), and a large cancelling negative term (the static pressure). The instantaneous pressure of a simulation cell as simulated by NAMD will have mean square fluctuations (according to David Case quoting Section 114 of *Statistical Physics* by Landau and Lifshitz) of $kT/(V\beta)$, where $\beta$ is the compressibility, which is RMS of roughly 100 bar for a 10,000 atom biomolecular system. Much larger fluctuations are regularly observed in practice.

The instantaneous pressure for a biomolecular system is well defined for “internal” forces that are based on particular periodic images of the interacting atoms, conserve momentum, and are translationally invariant. When dealing with externally applied forces such as harmonic constraints, fixed atoms, and various steering forces, NAMD bases its pressure calculation on the relative positions of the affected atoms in the input coordinates and assumes that the net force will average to zero over time. For time periods during with the net force is non-zero, the calculated pressure fluctuations will include a term proportional to the distance to the affected from the user-defined cell origin. A good way to observe these effects and to confirm that pressure for external forces is handled reasonably is to run a constant volume cutoff simulation in a cell that is larger than the molecular system by at least the cutoff distance; the pressure for this isolated system should average to zero over time.

Because NAMD’s impluse-basd multiple timestepping system alters the balance between bonded and non-bonded forces from every timestep to an average balance over two steps, the calculated pressure on even and odd steps will be different. The PRESSAVG and GPRESSAVG fields provide the average over the non-printed intermediate steps. If you print energies on every timestep you will see the effect clearly in the PRESSURE field.

The following options affect all pressure control methods.

- `useGroupPressure` < group or atomic quantities >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Pressure can be calculated using either the atomic virial and kinetic energy (the default) or a hydrogen-group based pseudo-molecular virial and kinetic energy. The latter fluctuates less and is required in conjunction with rigidBonds (SHAKE).

- `useFlexibleCell` < anisotropic cell fluctuations >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** NAMD allows the three orthogonal dimensions of the periodic cell to fluctuate independently when this option is enabled.

- `useConstantRatio` < constant shape in first two cell dimensions >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** When enabled, NAMD keeps the ratio of the unit cell in the x-y plane constant while allowing fluctuations along all axes. The `useFlexibleCell` option is required for this option.

- `useConstantArea` < constant area and normal pressure conditions >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** When enabled, NAMD keeps the dimension of the unit cell in the x-y plane constant while allowing fluctuations along the z axis. This is not currently implemented in Berendsen’s method.

### 7.5.1 Berendsen pressure bath coupling

NAMD provides constant pressure simulation using Berendsen’s method. The following parameters are used to define the algorithm.

- `BerendsenPressure` < use Berendsen pressure bath coupling? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not Berendsen pressure bath coupling is active. If set to `on`, then the parameters `BerendsenPressureTarget`, `BerendsenPressureCompressibility` and `BerendsenPressureRelaxationTime` must be set and the parameter `BerendsenPressureFreq` can optionally be set to control the behavior of this feature.

- `BerendsenPressureTarget` < target pressure (bar) >
  **Acceptable Values:** positive decimal
  **Description:** Specifies target pressure for Berendsen's method. A typical value would be 1.01325 bar, atmospheric pressure at sea level.

- `BerendsenPressureCompressibility` < compressibility (bar$^{-1}$) >
  **Acceptable Values:** positive decimal
  **Description:** Specifies compressibility for Berendsen's method. A typical value would be 4.57E-5 bar$^{-1}$, corresponding to liquid water. The higher the compressibility, the more volume will be adjusted for a given pressure difference. The compressibility and the relaxation time appear only as a ratio in the dynamics, so a larger compressibility is equivalent to a smaller relaxation time.

- `BerendsenPressureRelaxationTime` < relaxation time (fs) >
  **Acceptable Values:** positive decimal
  **Description:** Specifies relaxation time for Berendsen's method. If the instantaneous pressure did not fluctuate randomly during a simulation and the compressibility estimate was exact then the inital pressure would decay exponentially to the target pressure with this time constant. Having a longer relaxation time results in more averaging over pressure measurements and hence smaller fluctuations in the cell volume. A reasonable choice for relaxation time would be 100 fs. The compressibility and the relaxation time appear only as a ratio in the dynamics, so a larger compressibility is equivalent to a smaller relaxation time.

- `BerendsenPressureFreq` < how often to rescale positions >
  **Acceptable Values:** positive multiple of `nonbondedFrequency` and `fullElectFrequency`
  **Default Value:** `nonbondedFrequency` or `fullElectFrequency` if used
  **Description:** Specifies number of timesteps between position rescalings for Berendsen's method. Primarily to deal with multiple timestepping integrators, but also to reduce cell volume fluctuations, cell rescalings can occur on a longer interval. This could reasonably be between 1 and 20 timesteps, but the relaxation time should be at least ten times larger.

### 7.5.2 Nosé-Hoover Langevin piston pressure control

NAMD provides constant pressure simulation using a modified Nosé-Hoover method in which Langevin dynamics is used to control fluctuations in the barostat. This method should be combined with a method of temperature control, such as Langevin dynamics, in order to simulate the NPT ensemble.

The Langevin piston Nose-Hoover method in NAMD is a combination of the Nose-Hoover constant pressure method as described in GJ Martyna, DJ Tobias and ML Klein, "Constant pressure molecular dynamics algorithms", J. Chem. Phys 101(5), 1994, with piston fluctuation control implemented using Langevin dynamics as in SE Feller, Y Zhang, RW Pastor and BR Brooks, "Constant pressure molecular dynamics simulation: The Langevin piston method", J. Chem. Phys. 103(11), 1995.

The equations of motion are:

$$\begin{aligned}
r' &= p/m + e'r \\
p' &= F - e'p - gp + R \\
V' &= 3Ve' \\
e'' &= 3V/W(P - P_0) - g_e e' + R_e/W \\
W &= 3N\tau^2 kT \\
< R^2 > &= 2mgkT/h \\
\tau &= \text{oscillationperiod} \\
< R_e^2 > &= 2Wg_e kT/h
\end{aligned}$$

Here, $W$ is the mass of piston, $R$ is noise on atoms, and $R_e$ is the noise on the piston.

The user specifies the desired pressure, oscillation and decay times of the piston, and temperature of the piston. The compressibility of the system is not required. In addition, the user specifies the damping coefficients and temperature of the atoms for Langevin dynamics.

The following parameters are used to define the algorithm.

- `LangevinPiston` < use Langevin piston pressure control? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not Langevin piston pressure control is active. If set to `on`, then the parameters `LangevinPistonTarget`, `LangevinPistonPeriod`, `LangevinPistonDecay` and `LangevinPistonTemp` must be set.

- `LangevinPistonTarget` < target pressure (bar) >
  **Acceptable Values:** positive decimal
  **Description:** Specifies target pressure for Langevin piston method. A typical value would be 1.01325 bar, atmospheric pressure at sea level.

- `LangevinPistonPeriod` < oscillation period (fs) >
  **Acceptable Values:** positive decimal
  **Description:** Specifies barostat oscillation time scale for Langevin piston method. If the instantaneous pressure did not fluctuate randomly during a simulation and the decay time was infinite (no friction) then the cell volume would oscillate with this angular period. Having a longer period results in more averaging over pressure measurements and hence slower fluctuations in the cell volume. A reasonable choice for the piston period would be 200 fs.

- `LangevinPistonDecay` < damping time scale (fs) >
  **Acceptable Values:** positive decimal
  **Description:** Specifies barostat damping time scale for Langevin piston method. A value larger than the piston period would result in underdamped dynamics (decaying ringing in the cell volume) while a smaller value approaches exponential decay as in Berendsen's method above. A smaller value also corresponds to larger random forces with increased coupling to

the Langevin temperature bath. Typically this would be chosen equal to or smaller than the piston period, such as 100 fs.

- `LangevinPistonTemp` < noise temperature (K) >
  **Acceptable Values:** positive decimal
  **Description:** Specifies barostat noise temperature for Langevin piston method. This should be set equal to the target temperature for the chosen method of temperature control.

- `SurfaceTensionTarget` < Surface tension target (dyn/cm) >
  **Acceptable Values:** decimal
  **Default Value:** 0.0
  **Description:** Specifies surface tension target. Must be used with `useFlexibleCell` and periodic boundary conditions. The pressure specified in `LangevinPistonTarget` becomes the pressure along the z axis, and surface tension is applied in the x-y plane.

- `StrainRate` < initial strain rate >
  **Acceptable Values:** decimal triple (x y z)
  **Default Value:** 0. 0. 0.
  **Description:** Optionally specifies the initial strain rate for pressure control. Is overridden by value read from file specified with `extendedSystem`. There is typically no reason to set this parameter.

- `ExcludeFromPressure` < Should some atoms be excluded from pressure rescaling? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not to exclude some atoms from pressure rescaling. The coordinates and velocites of such atoms are not rescaled during constant pressure simulations, though they do contribute to the virial calculation. May be useful for membrane protein simulation. EXPERIMENTAL.

- `ExcludeFromPressureFile` < File specifying excluded atoms >
  **Acceptable Values:** PDB file
  **Default Value:** coordinates file
  **Description:** PDB file with one column specifying which atoms to exclude from pressure rescaling. Specify 1 for excluded and 0 for not excluded.

- `ExcludeFromPressureCol` < Column in PDB file for specifying excluded atoms >
  **Acceptable Values:** O, B, X, Y, or Z
  **Default Value:** O
  **Description:** Specifies which column of the pdb file to check for excluded atoms.

### 7.5.3 Monte Carlo barostat

NAMD provides constant pressure simulation using Monte Carlo method to control volume fluctuations in the barostat. This method should be combined with a method of temperature control, in order to simulate the NPT ensemble. This feature is only supported in *GPU Resident mode*.

Please note that before switching to constant pressure simulations, you will need to equilibrate your system using NVT ensemble simulation.

In this method, a trial volume-change, $\Delta V$, is generated uniformly, within the range of $[-\Delta V_{\max}, +\Delta V_{\max}]$, where $\Delta V_{\max}$ is the maximum volume-change, adjusted to obtain the target acceptance ratio.

The probability of accepting volume-change from $V$ to $V' = V + \Delta V$ is:

$$\begin{aligned}
P^{acc} &= \min\left[1, \left(\frac{V'}{V}\right)^{N} e^{-\beta(\Delta U + P\Delta V - \gamma\Delta A)}\right] \\
\Delta U &= U\left(\mathbf{s}^N, V'\right) - U\left(\mathbf{s}^N, V\right) \\
\Delta V &= V' - V \\
\Delta A &= \text{change in x-y area} \\
\beta &= \frac{1}{kT} \\
\mathbf{s}^N &= \text{reduced coordinates}
\end{aligned}$$

where, $\gamma$, $P$, $T$, $U\left(\mathbf{s}^N, V'\right)$, and $U\left(\mathbf{s}^N, V\right)$ are the target surface tension, target pressure, system temperature, and total potential energy of the system at *new* and *old* configuration, respectively. $N$ denotes the total number of molecule in the system.

The following parameters are used to define the algorithm.

- `MonteCarloPressure` < use Monte Carlo pressure control? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not Monte Carlo pressure control is active. If set to `on`, then the parameters `MonteCarloPressureTarget` and `MonteCarloTemp` must be set. The parameters `MonteCarloPressureFreq`, `MonteCarloMaxVolume`, `MonteCarloAcceptanceRate`, `MonteCarloAdjustmentFreq`, and `SurfaceTensionTarget` can optionally be set to control the behavior of this feature.

- `MonteCarloPressureTarget` < target pressure (bar) >
  **Acceptable Values:** positive decimal
  **Description:** Specifies target pressure for Monte Carlo barostat. A typical value would be 1.01325 bar, atmospheric pressure at sea level.

- `MonteCarloTemp` < system temperature (K) >
  **Acceptable Values:** positive decimal
  **Description:** Specifies temperature for Monte Carlo barostat. This should be set equal to the target temperature for the chosen method of temperature control.

- `MonteCarloPressureFreq` < how often attempt to change volume? >
  **Acceptable Values:** positive multiple of `nonbondedFrequency` and `fullElectFrequency`
  **Default Value:** 50 * `nonbondedFrequency` or 50 * `fullElectFrequency` if used
  **Description:** Specifies number of timesteps between attempt to change the volume and rescale the coordinates. A typical value for this parameter could be 50.

- `MonteCarloMaxVolume` < initial maximum volume-change for each axis dimension ($\text{Å}^3$) >
  **Acceptable Values:** vector of positive decimal and less than $0.3V$

**Default Value:** $0.01V$ $0.01V$ $0.01V$
**Description:** Optionally specifies the initial maximum volume-change for each axis dimension ($\Delta V_{\max}$) in Monte Carlo pressure control. If `extendedSystem` is also specified, value is overridden. There is typically no reason to set this parameter.

- `MonteCarloAcceptanceRate` < target volume-change acceptance rate >
**Acceptable Values:** decimal between 0.2 and 0.8
**Default Value:** 0.5
**Description:** Specifies the desired acceptance rate for volume-change attempt. The $\Delta V_{\max}$ is dynamically adjusted to reach the target acceptance rate. Higher acceptance rate results in small volume fluctuations, while lower acceptance rate results in large volume fluctuations.

- `MonteCarloAdjustmentFreq` < how often adjust maximum volume-change? >
**Acceptable Values:** positive integer
**Default Value:** 30
**Description:** Specifies the number of volume-change attempts, to collect acceptance rate statistics, in order to adjust maximum volume-change ($\Delta V_{\max}$). Higher value would provide more accurate statistics, but might require longer simulation to reach equilibration. A typical value of 30 for isotropic and 50 for anisotropic cell fluctuation could be used for this parameter.

- `SurfaceTensionTarget` < Surface tension target (dyn/cm) >
**Acceptable Values:** decimal
**Default Value:** 0.0
**Description:** Specifies surface tension target. Must be used with `useFlexibleCell` and periodic boundary conditions. The pressure specified in `MonteCarloPressureTarget` becomes the pressure along the z axis, and surface tension is applied in the x-y plane.

# 8 User Defined Forces

There are several ways to apply external forces to simulations with NAMD. These are described below.

## 8.1 Constant Forces

NAMD provides the ability to apply constant forces to some atoms. There are three parameters that control this feature.

- `constantForce` < Apply constant forces? >
  **Acceptable Values:** yes or no
  **Default Value:** no
  **Description:** Specifies whether or not constant forces are applied.

- `consForceFile` < PDB file containing forces to be applied >
  **Acceptable Values:** UNIX filename
  **Description:** The X, Y, Z and occupancy (O) fields of this file are read to determine the constant force vector of each atom, which is (X,Y,Z)*O, in unit of kcal/(mol*Å). The occupancy (O) serves as a scaling factor, which could expand the range of the force applied. (One may be unable to record very large or very small numbers in the data fields of a PDB file due to limited space). Zero forces are ignored.

  Specifying `consforcefile` is optional; constant forces may be specified or updated between runs by using the consForceConfig command.

- `consForceScaling` < Scaling factor for constant forces >
  **Acceptable Values:** decimal
  **Default Value:** 1.0
  **Description:** Scaling factor by which constant forces are multiplied. May be changed between run commands.

## 8.2 External Electric Field

NAMD provides the ability to apply a constant electric field to the molecular system being simulated. Energy due to the external field will be reported in the MISC column and will be continuous even in simulations using periodic boundary conditions as unwrapped coordinates are used to calculate energy and pressure, resulting in linearly increasing pressure over time for systems with free ions. To avoid this effect, for periodic simulations the new `eFieldNormalized` option should be used with the electric field vector multiplied by the cell dimension. There are three parameters that control this feature.

- `eFieldOn` < apply electric field? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Specifies whether or not an electric field is applied.

- `eField` < electric field vector >
  **Acceptable Values:** vector of decimals (x y z)
  **Description:** Vector which describes the electric field to be applied. Units are

kcal/(mol Å $e$), which is natural for simulations. This parameter may be changed between `run` commands, allowing a square wave or other approximate wave form to be applied.

- `eFieldNormalized` < electric field vector scaled by cell basis vectors? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Specifies whether or not that eField vector has been scaled by the cell basis vectors, thus indicating the voltage drop across the cell in units of kcal/(mol $e$). The eField vector is then scaled by the reciprocal lattice vectors at each timestep. When eFieldNormalized is true the eField forces do not contribute to the pressure calculation.

## 8.3 Grid Forces

NAMD provides the ability to specify grids describing a potential in the simulation space. Each atom is affected by the potential based on its charge and its position, using the potential function interpolated from the specified grid(s). Energy due to the grid-defined field will be reported in the MISC column of the output, unless a scaling factor not proportional to (1,1,1) is used.

NAMD allows the definition of multiple grids, each with a separate set of defining parameters. This is specified using a tag field in each of the mgridforceXXX commands. The tag is an alphanumeric string without spaces which identifies to which grid the specified field applies.

The grid file format is a subset of the DataExplorer DX file format, as shown below:

```
# Lines at the beginning of the file starting with a # symbol
# are ignored as comments
# Variables (replaced by numbers in an actual file):
#   xn, yn, and zn are the number of data points along each dimension;
#   xorg, yorg, and zorg is the origin of the grid, in angstroms;
#   x[1-3]del, y[1-3]del, and z[1-3]del are the basis vectors which transform
#   grid indices to coordinates in angstroms:
#      x(i,j,k) = xorg + i * x1del + j * y1del + k * z1del
#      y(i,j,k) = yorg + i * x2del + j * y2del + k * z2del
#      z(i,j,k) = zorg + i * x3del + j * y3del + k * z3del
#
#   Grid data follows, with three values per line, ordered z fast, y medium,
#   and x slow. Exactly xn*yn*zn values should be given. Grid data is then
#   terminated with a field object.
#
#  Note: Other features of the DX file format are not handled by this code
#
object 1 class gridpositions counts xn yn zn
origin xorg yorg zorg
delta x1del y1del z1del
delta x2del y2del z2del
delta x3del y3del z3del
object 2 class gridconnections counts xn yn zn
object 3 class array type double rank 0 items [ xn*yn*zn ] data follows
f1 f2 f3
f4 f5 f6
```

```
.
.
.
object 4 class field
component "positions" value 1
component "connections" value 2
component "data" value 3
```

Each dimension of the grid may be specified as continuous or not. If the grid is not continuous in a particular dimension, the potential grid is padded with one border slices on each non-continuous face of the grid, and border grid values are computed so that the force felt by an atom outside the grid goes to zero. If the grid is continuous along a particular dimension, atoms outside the grid are affected by a potential that is interpolated from the grid and its corresponding periodic image along that dimension.

To calculate the force on an atom due to the grid, the atom's coordinates are transformed according to the current basis vectors of the simulation box to a coordinate frame that is centered at the center of the specified grid. Note that the size and spatial coordinates of the grid remain fixed, and are not scaled as the size of the simulation box fluctuates. For atoms within the grid, the force is computed by analytically determining the gradient of the tricubic polynomial used to interpolate the potential from surrounding grid values. For atoms outside the grid, the state of the `mgridforcecont[1,2,3]` determine whether the force is zero, or computed from the images of the grid as described above. Note that if the grid is ever larger than the periodic box, it is truncated at the edge of that box. The consequence of this is that the computed potential will not vary smoothly at the edges, introducing numerical instability.

NAMD also supports non-uniform grids, allowing regions of a grid to be defined at higher resolution. Non-uniform grids are structured hierarchically, with a single *maingrid* which has one or more *subgrid*s. Each subgrid spans a number of maingrid cells in each of the three dimensions, and effectively redefines the data in that region. The subgrids are usually defined at higher resolution, with the restriction that the number of cells along each dimension is an integral number of the original number in the maingrid. Note that the maingrid still has data points in regions where subgrids are defined, and that, on the boundary of a subgrid, *they must agree with the values in the subgrid.* Subgrids, in turn, may have subgrids of their own, which may have subgrids of their own, etc.

A non-uniform grid file takes the form of a special comment block followed by multiple normal grid definitions. The special comment block defines the grid hierarchy, and consists of comments beginning with `# namdnugrid`. An example follows:

```
# namdnugrid version 1.0
# namdnugrid maingrid subgrids count 2
# namdnugrid subgrid 1 generation 1 min x1 y1 z1 max x2 y2 z2 subgrids count 2
# namdnugrid subgrid 2 generation 2 min x3 y3 z3 max x4 y4 z4 subgrids count 0
# namdnugrid subgrid 3 generation 2 min x5 y5 z5 max x6 y6 z6 subgrids count 0
# namdnugrid subgrid 4 generation 1 min x7 y7 z7 max x8 y8 z8 subgrids count 0
```

The maingrid is described by the number of subgrids. Subgrids are additionally described by a subgrid number; a generation number, which should be one higher than the generation of its supergrid; and `min` and `max` attributes, which describe the location of the subgrid within its supergrid. In this example, the maingrid has two subgrids, subgrid 1 and subgrid 4, labeled `generation 1`.

The first of these subgrids has two subgrids of its own (`generation 2`). Notice that subgrids are described immediately after their supergrid. The `min` and `max` attributes are given in units of grid *cells* of the supergrid. For example, a subgrid with `min 0 0 0 max 1 1 1` would redefine 8 cells of its supergrid, the space between gridpoints (0, 0, 0) and (2, 2, 2) in grid coordinates. Following the comment block, the maingrid and subgrids are defined in the format described above, in the same order as the comment block.

The following parameters describe the grid-based potentials.

- `mgridforce` < apply grid forces? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Specifies whether or not any grid forces are being applied.

- `mgridforcefile` < tag > < PDB file specifying force multipliers and charges for each atomd >
  **Acceptable Values:** UNIX file name
  **Description:** The force on each atom is scaled by the corresponding value in this PDB file. By setting the force multiplier to zero for an atom, it will not be affected by the grid force.

- `mgridforcecol` < tag > < column of PDB from which to read force multipliers >
  **Acceptable Values:** X, Y, Z, O, or B
  **Default Value:** B
  **Description:** Which column in the PDB file specified by `mgridforcefile` contains the scaling factor

- `mgridforcechargecol` < tag > < column of PDB from which to read atom charges >
  **Acceptable Values:** X, Y, Z, O, or B
  **Default Value:** Atom charge used for electrostatics.
  **Description:** Which column in the PDB file specified by `mgridforcefile` contains the atom charge. By default, the charge value specified for the short-range Columb interactions are also used for the grid force. Both `mgridforcecol` and `mgridforceqcol` can be specified, in which case the apparent charge of the atom will be the product of the two values.

- `mgridforcepotfile` < tag > < grid potential file name >
  **Acceptable Values:** UNIX file name
  **Description:** File specifying the grid size, coordinates, and potential values.

- `mgridforcevolts` < tag > < grid potential units in eV/charge >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** If set, the grid potential values are expressed in eV. Otherwise, values are in kcal/(mol *charge*)

- `mgridforcescale` < tag > < scale factor for grid potential >
  **Acceptable Values:** Vector of decimals $scale_x$ $scale_y$ $scale_z$
  **Default Value:** 1 1 1
  **Description:** Defines the scale factors that modulate the amplitude of the grid potential forces in each dimension. When the three values are the same number, the grid potential's value is also included in the MISC column of the energy output. After initialization, the

three scale factors may be updated between “run” commands by the `updategridforcescale` command. In the special case when “0 0 0” is given for this option, the corresponding grid potential can be used a collective variable in the Colvars module (Sec. 9), allowing the use of restraint potentials and fully time-dependent forces.

- `updategridforcescale` < tag > < scale factor for grid potential >
  **Acceptable Values:** Vector of decimals $scale_x$ $scale_y$ $scale_z$
  **Default Value:** 1 1 1
  **Description:** Provides new scale factors to be applied to the grid potential values. This comand can be issued between “run” commands to modify the amplitude of the grid potential. The values provided remain constant for the duration of each “run” command.

- `mgridforcecont1` < tag > < Is grid continuous in the direction of the first basis vector >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** By specifying that the grid is continuous in a direction, atoms outside of the grid will be affected by a force determined by interpolating based on the values at the edge of the grid with the values of the corresponding edge of the periodic image of the grid. The current size of the simulation box is taken into account, so that as the simulation box size fluctuates, the force on an atom outside of the grid varies continuously until it re-enters the opposite edge of the grid. If the grid is not continuous in this direction, the interpolated force on atoms near the edge of the grid is calculated so that it continuously approaches zero as an atom approaches the edge of the grid.

- `mgridforcecont2` < tag > < Is grid continuous in the direction of the second basis vector >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Operates the same as `mgridforcecont1` except applies in the direction of the second basis vector

- `mgridforcecont3` < tag > < Is grid continuous in the direction of the third basis vector >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Operates the same as `mgridforcecont1` except applies in the direction of the third basis vector

- `mgridforcevoff` < tag > < Offset periodic images of the grid by specified amounts >
  **Acceptable Values:** vector of decimals (x y z)
  **Default Value:** (0 0 0)
  **Description:** If a continuous grid is used along a particular basis vector, it may be desirable to shift the potentials in the image to manipulate the potential outside the grid. For example, consider the case where the potential is a ramp in the $x$ direction and the grid is defined for points $[0, N)$, with a potential $f(i, j, k)$ given by $f(i, j, k) = f_0 + i(f_1 - f_0)/N$. By shifting the images of the grid, the potential can be transformed as illustrated in Fig. 4.

- `mgridforcelite` < tag > < Is grid to use Gridforce Lite interpolation? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`

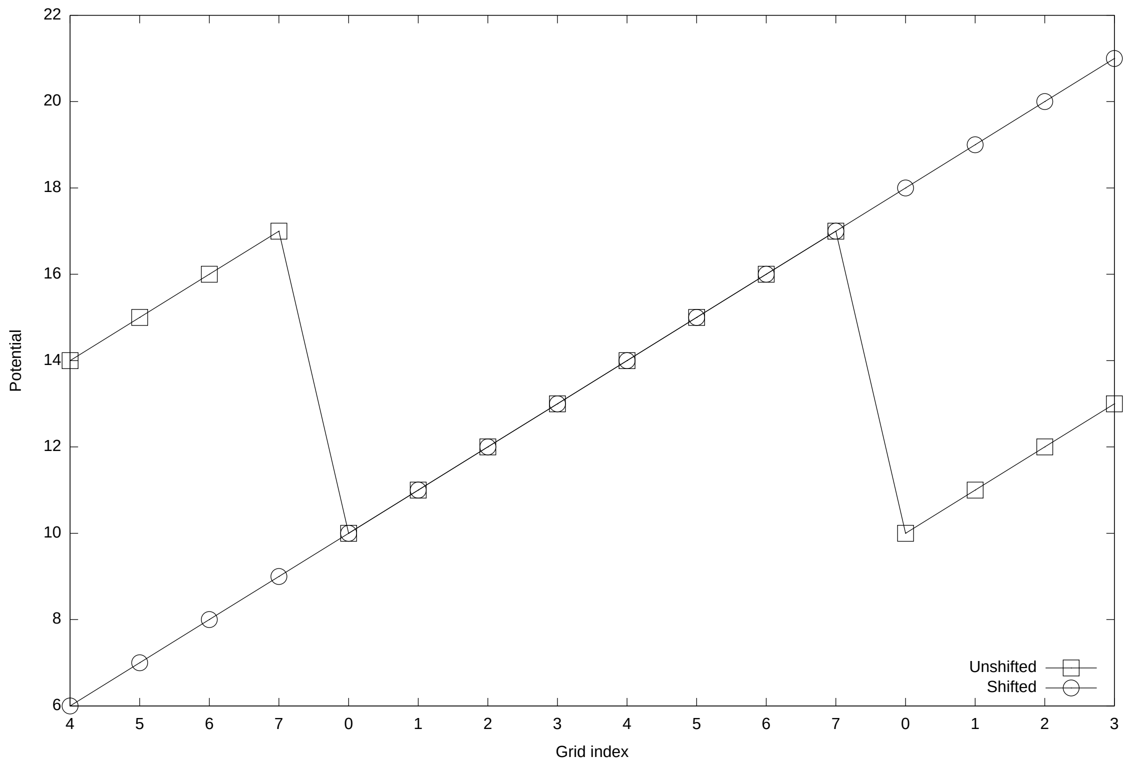


Figure 4: Graph showing a slice of a ramp potential, with eight grid points along the axis, and a periodic cell size which just contains the grid. The Unshifted case shows how the pontential is not smooth when `mgridforcevoff` is not specified, or set to zero. The Shifted potential shows the grid that results when `mgridfocevoff` is set so that the wrapped potential is offset so that the potential has constant slope at the periodic boundaries.

**Description:** When Gridforce Lite is enabled, a faster but less accurate interpolation method is used to compute forces. Specifically, rather than computing a tri-cubic interpolation of the potential, from which the force is then computed analytically, Gridforce Lite computes force as a linear interpolation. This method also increases the memory required by Gridforce. Note that Gridforce Lite is incompatible with use of the `mgridforcecont[123]` keywords and with non-uniform grids.

## 8.4 Moving Constraints

Moving constraints feature works in conjunction with the Harmonic Constraints (see an appropriate section of the User's guide). The reference positions of all constraints will move according to

$$\vec{r}(t) \;=\; \vec{r}_0 \,+\, \vec{v}t\,. \tag{30}$$

A velocity vector $\vec{v}$ (`movingConsVel`) needs to be specified.

The way the moving constraints work is that the moving reference position is calculated every integration time step using Eq. 30, where $\vec{v}$ is in Å/timestep, and $t$ is the current timestep (i.e., `firstTimestep` plus however many timesteps have passed since the beginning of NAMD run). Therefore, one should be careful when restarting simulations to appropriately update the

`firstTimestep` parameter in the NAMD configuration file or the reference position specified in the reference PDB file.
**NOTE:** NAMD actually calculates the constraints potential with $U = k(x - x_0)^d$ and the force with $F = dk(x - x_0)$, where $d$ is the exponent `consexp`. The result is that if one specifies some value for the force constant $k$ in the PDB file, effectively, the force constant is $2k$ in calculations. This caveat was removed in SMD feature.

The following parameters describe the parameters for the moving harmonic constraint feature of NAMD.

- `movingConstraints` < Are moving constraints active >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Should moving restraints be applied to the system. If set to `on`, then `movingConsVel` must be defined. May not be used with `rotConstraints`.

- `movingConsVel` < Velocity of the reference position movement >
  **Acceptable Values:** vector in Å/timestep
  **Description:** The velocity of the reference position movement. Gives both absolute value and direction

## 8.5 Rotating Constraints

The constraints parameters are specified in the same manner as for usual (static) harmonic constraints. The reference positions of all constrained atoms are then rotated with a given angular velocity about a given axis. If the force constant of the constraints is sufficiently large, the constrained atoms will follow their reference positions.

A rotation matrix $M$ about the axis unit vector $v$ is calculated every timestep for the angle of rotation corresponding to the current timestep. angle $= \Omega t$, where $\Omega$ is the angular velocity of rotation.

From now on, all quantities are 3D vectors, except the matrix $M$ and the force constant $K$.

The current reference position $R$ is calculated from the initial reference position $R_0$ (at $t = 0$), $R = M(R_0 - P) + P$, where $P$ is the pivot point.

Coordinates of point N can be found as $N = P + ((R - P) \cdot v)v$. Normal from the atom pos to the axis is, similarly, normal $= (P + ((X - P) \cdot v)v) - X$ The force is, as usual, $F = K(R - X)$; This is the force applied to the atom in NAMD (see below). NAMD does not know anything about the torque applied. However, the torque applied to the atom can be calculated as a vector product torque $= F \times normal$ Finally, the torque applied to the atom with respect to the axis is the projection of the torque on the axis, i.e., $torque_{proj} = torque \cdot v$

If there are atoms that have to be constrained, but not moved, this implementation is not suitable, because it will move *all* reference positions.

Only one of the moving and rotating constraints can be used at a time.

Using very soft springs for rotating constraints leads to the system lagging behind the reference positions, and then the force is applied along a direction different from the "ideal" direction along the circular path.

Pulling on N atoms at the same time with a spring of stiffness K amounts to pulling on the whole system by a spring of stiffness NK, so the overall behavior of the system is as if you are pulling with a very stiff spring if N is large.

In both moving and rotating constraints the force constant that you specify in the constraints pdb file is multiplied by 2 for the force calculation, i.e., if you specified $K = 0.5$ kcal/mol/Å$^2$ in the pdb file, the force actually calculated is $F = 2K(R - X) = 1$ kcal/mol/Å$^2$ $(R - X)$. SMD feature of NAMD does the calculation without multiplication of the force constant specified in the config file by 2.

- `rotConstraints` < Are rotating constraints active >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Should rotating restraints be applied to the system. If set to `on`, then `rotConsAxis`, `rotConsPivot` and `rotConsVel` must be defined. May not be used with `movingConstraints`.

- `rotConsAxis` < Axis of rotation >
  **Acceptable Values:** vector (may be unnormalized)
  **Description:** Axis of rotation. Can be any vector. It gets normalized before use. If the vector is 0, no rotation will be performed, but the calculations will still be done.

- `rotConsPivot` < Pivot point of rotation >
  **Acceptable Values:** position in Å
  **Description:** Pivot point of rotation. The rotation axis vector only gives the direction of the axis. Pivot point places the axis in space, so that the axis goes through the pivot point.

- `rotConsVel` < Angular velocity of rotation >
  **Acceptable Values:** rate in degrees per timestep
  **Description:** Angular velocity of rotation, degrees/timestep.

## 8.6 Symmetry Restraints

Symmetry restraints are based on symmetrical relationships between monomers. Defined monomers are transformed to overlap and an average position for each atom is calculated. After the average structure is transformed back, a harmonic force is calculated which drives each monomer to the average.

- `symmetryRestraints` < Are symmetry restraints active? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Should Symmetry constraining forces be applied to the system. If symmetry restraints are enabled, `symmetryk`* and `symmetryFile` must be defined in the input file as well. *See `symmetryk` entry for details.

- `symmetryFirstFullStep` < First step to apply full harmonic force >
  **Acceptable Values:** Non-negative integer
  **Default Value:** `symmetryFirstStep`
  **Description:** Force constant `symmetryk` linearly increased from `symmetryFirstStep` to `symmetryFirstFullStep`

- `symmetryLastFullStep` < Last step to apply full harmonic force >
  **Acceptable Values:** Non-negative integer

**Default Value:** `symmetryLastStep`
**Description:** Force constant `symmetryk` linearly decreased from `symmetryLastFullStep` to `symmetryLastStep`

- `symmetryk` < Constant for harmonic restraining forces >
  **Acceptable Values:** Positive value
  **Description:** Harmonic force constant. Scaled down by number of atoms in the monomer. If this setting is omitted, the value in the occupancy column of the pdb file specified by `symmetrykFile` will be used as the constant for that atom. This allows the user to specify the constant on a per-atom basis.

- `symmetrykFile` < pdb containing per atom force constants >
  **Acceptable Values:** Path to pdb file
  **Description:** pdb where the occupancy column specifies the per atom force constants. If using overlapping symmetry groups, you must include one additional `symmetrykfile` per `symmetryFile`

- `symmetryScaleForces` < Scale symmetry restraints over time >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** If turned on, the harmonic force applied by the symmetry restraints will linearly evolve with each time step based on `symmetryFirstFullStep` and `symmetryLastFullStep`.

- `symmetryFile` < File for symmetry information >
  **Acceptable Values:** Path to PDB file
  **Description:** Restrained atoms are those whose occupancy (O) is nonzero in the symmetry pdb file. The file must contain no more atoms than the structure file and those atoms present must have the exact same index as the structure file (i.e., the file may contain a truncated atom selection "index $< N$" but not an arbitrary selection). The value in the occupancy column represent the "symmetry group" the atom belongs to. These symmetry groups are used for denoting monomers of the same type. These groups will be transformed by the matrices in their own `symmetryMatrixFile` and averaged separetely from other groups. The designation in the occupancy column should be an integer value starting at 1 and proceeding in ascending order, mirroring the order of the corresponding matrix file within the configuration file (e.g. the first symmetryMatrixFile contains the matrices for symmetry group 1). The value in the atom's beta column represents its monomer designation. This should be an integer value starting at 1 and proceeding in ascending order, relative to the order of the corresponding transformation matrix found in the symmetry group's `symmetryMatrixFile`. If an atom is contained in more than one symmetry group, additional pdb files can be listed. These pdb files should follow the same rules as the first one (unique group and monomer identifiers in increasing order).

- `symmetryMatrixFile` < File for transformation matrices >
  **Acceptable Values:** Path to matrix file
  **Description:** The `symmetryMatrixFile` is a path to a file that contains a list of transformation matrices to make the monomers overlap. The file should contain one (and only one) matrix for each monomer in the order of monomer ID designated in the symmetryFile.

Each symmetry group should have its own symmetryMatrixFile file containing only the matrices used by the monomers in that group. These should be formatted with spaces between columns and NO spaces between rows as follows:

1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1

with different matrices separated by a single blank line (and no line before the first or after the last matrix). This file is OPTIONAL. Leave this line out to have namd generate the transformations for you.

- `symmetryFirstStep` < first symmetry restraint timestep >
  **Acceptable Values:** Non-negative integer
  **Default Value:** 0
  **Description:**

- `symmetryLastStep` < last symmetry restraint timestep >
  **Acceptable Values:** Positive integer
  **Default Value:** infinity
  **Description:** Symmetry restraints are applied only between `symmetryFirstStep` and `symmetryLastStep`. Use these settings with caution and ensure restraints are only being applied when necessary (e.g. not during equilibration).

## 8.7 Targeted Molecular Dynamics (TMD)

In TMD, subset of atoms in the simulation is guided towards a final 'target' structure by means of steering forces. At each timestep, the RMS distance between the current coordinates and the target structure is computed (after first aligning the target structure to the current coordinates). The force on each atom is given by the gradient of the potential

$$U_{TMD} = \frac{1}{2}\frac{k}{N}\left[RMS(t) - RMS^*(t)\right]^2 \tag{31}$$

where $RMS(t)$ is the instantaneous best-fit RMS distance of the current coordinates from the target coordinates, and $RMS^*(t)$ evolves linearly from the initial RMSD at the first TMD step to the final RMSD at the last TMD step. The spring constant $k$ is scaled down by the number $N$ of targeted atoms.

Atoms can be separated into non-overlapping constraint domains by assigning integer values in the beta column of the `TMDFile`. Forces on the atoms will be calculated for each domain independently of the other domains.

Within each domain, the set of atoms used to fit the target structure can be different from the set of atoms that are biased towards the target structure. If the altloc field in the `TMDFile` is not ' ' or '0' then the atom is fitted. If the occupancy is non-zero then the atom is biased. If none of the atoms in a domain have altloc set then all biased atoms are fitted.

Note that using different atoms for fitting and biasing or not using the same spring constant for all target atoms within a domain will result in forces conserving neither energy nor momentum. In this case harmonic restraints and Langevin dynamics are likely needed.

- `TMD` < Is TMD active >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Should TMD steering forces be applied to the system. If TMD is enabled, `TMDk`, `TMDFile`, and `TMDLastStep` must be defined in the input file as well.

- `TMDk` < Elastic constant for TMD forces >
  **Acceptable Values:** Positive value in $kcal/mol/\text{Å}^2$.
  **Description:** The value of $k$ in Eq. 31. A value of 200 seems to work well in many cases. If this setting is omitted, the value in the occupancy column of the pdb file specified by `TMDFile` will be used as the constant for that atom. This allows the user to specify the constant on a per-atom basis.

- `TMDOutputFreq` < How often to print TMD output >
  **Acceptable Values:** Positive integer
  **Default Value:** 1
  **Description:** TMD output consists of lines of the form `TMD ts targetRMS currentRMS` where `ts` is the timestep, `targetRMS` is the target RMSD at that timestep, and `currentRMS` is the actual RMSD.

- `TMDFile` < File for TMD information >
  **Acceptable Values:** Path to PDB file
  **Description:** Biased atoms are those whose occupancy (O) is nonzero in the TMD PDB file. Fitted atoms are those whose altloc field is not ‘ ’ or ‘0’, if present, otherwise all biased atoms are fitted. The file must contain no more atoms than the structure file and those atoms present must have the exact same index as the structure file (i.e., the file may contain a truncated atom selection “index $< N$” but not an arbitrary selection). The coordinates for the target structure are also taken from the targeted atoms in this file. Non-targeted atoms are ignored. The beta column of targetted atoms is used to designate non-overlapping constraint domains. Forces will be calculated for atoms within a domain separately from atoms of other domains.

- `TMDFirstStep` < first TMD timestep >
  **Acceptable Values:** Positive integer
  **Default Value:** 0
  **Description:**

- `TMDLastStep` < last TMD timestep >
  **Acceptable Values:** Positive integer
  **Description:** TMD forces are applied only between `TMDFirstStep` and `TMDLastStep`. The target RMSD evolves linearly in time from the initial to the final target value.

- `TMDInitialRMSD` < target RMSD at first TMD step >
  **Acceptable Values:** Non-negative value in Å
  **Default Value:** from coordinates
  **Description:** In order to perform TMD calculations that involve restarting a previous NAMD run, be sure to specify `TMDInitialRMSD` with the same value in each NAMD input file, and use the NAMD parameter `firstTimestep` in the continuation runs so that the target RMSD continues from where the last run left off.

- `TMDFinalRMSD` < target RMSD at last TMD step >
  **Acceptable Values:** Non-negative value in Å
  **Default Value:** 0
  **Description:** If no `TMDInitialRMSD` is given, the initial RMSD will be calculated at the first TMD step. `TMDFinalRMSD` may be less than or greater than `TMDInitialRMSD`, depending on whether the system is to be steered towards or away from a target structure, respectively. Forces are applied only if $RMS(t)$ is betwween `TMDInitialRMSD` and $RMS*(t)$; in other words, only if the current RMSD fails to keep pace with the target value.

- `TMDDiffRMSD` < Is double-sided TMD active? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Turns on the double-sided TMD feature which targets the transition between two structures. This is accomplished by modifying the TMD force such that the potential is based on the difference of RMSD's from the two structures:

$$U_{TMD} = \frac{1}{2}\frac{k}{N}\left[DRMS(t) - DRMS^{*}(t)\right]^{2} \tag{32}$$

  where $DRMS(t)$ is RMS1(t) - RMS2(2) (RMS1 being the RMSD from structure 1 and RMS2 the RMSD from structure 2). The first structure is specified as normal in `TMDFile` and the second structure should be specified in `TMDFile2`, preserving any domain designations found in `TMDFile`.

- `TMDFile2` < Second structure file for double-sided TMD >
  **Acceptable Values:** Path to PDB file
  **Description:** PDB file defining the second structure of a double sided TMD. This file should contain the same number of atoms as `TMDFile` along with the same domain designations if any are specified.

## 8.8 Steered Molecular Dynamics (SMD)

The SMD feature is independent from the harmonic constraints, although it follows the same ideas. In both SMD and harmonic constraints, one specifies a PDB file which indicates which atoms are 'tagged' as constrained. The PDB file also gives initial coordinates for the constraint positions. One also specifies such parameters as the force constant(s) for the constraints, and the velocity with which the constraints move.

There are two major differences between SMD and harmonic constraints:

- In harmonic constraints, each tagged atom is harmonically constrained to a reference point which moves with constant velocity. In SMD, it is the *center of mass* of the tagged atoms which is constrained to move with constant velocity.

- In harmonic constraints, each tagged atom is constrained in all three spatial dimensions. In SMD, tagged atoms are constrained *only along the constraint direction* (unless the optional `SMDk2` keyword is used.)

The center of mass of the SMD atoms will be harmonically constrained with force constant $k$ (`SMDk`) to move with velocity $v$ (`SMDVel`) in the direction $\vec{n}$ (`SMDDir`). SMD thus results in the following potential being applied to the system:

$$U(\vec{r}_1, \vec{r}_2, ..., t) \;=\; \frac{1}{2}k\left[vt - (\vec{R}(t) - \vec{R}_0)\cdot\vec{n}\right]^2. \tag{33}$$

Here, $t \equiv N_{ts}dt$ where $N_{ts}$ is the number of elapsed timesteps in the simulation and $dt$ is the size of the timestep in femtoseconds. Also, $\vec{R}(t)$ is the current center of mass of the SMD atoms and $R_0$ is the initial center of mass as defined by the coordinates in `SMDFile`. Vector $\vec{n}$ is normalized by NAMD before being used.

Optionally, one may also specify a transverse force constant $k_2$ (`SMDk2`). The potential then becomes

$$U(\vec{r}_1, \vec{r}_2, ..., t) \;=\; \frac{1}{2}k\left[vt - (\vec{R}(t) - \vec{R}_0)\cdot\vec{n}\right]^2 + \frac{1}{2}k_2\left[\left(\vec{R}(t) - \vec{R}_0\right)^2 - \left((\vec{R}(t) - \vec{R}_0)\cdot\vec{n}\right)^2\right]. \tag{34}$$

In this case, the force constant $k$ controls the potential *parallel* to the pulling direction $\vec{n}$, while the transverse force constant $k_2$ controls the potential *perpendicular* to $\vec{n}$.

**Output** NAMD provides output of the current SMD data. The frequency of output is specified by the `SMDOutputFreq` parameter in the configuration file. Every `SMDOutputFreq` timesteps NAMD will print the current timestep, current position of the center of mass of the restrained atoms, and the current force applied to the center of mass (in piconewtons, pN). The output line starts with word `SMD`. The energy associated with SMD is reported under the MISC title of NAMD's energy output.

**Parameters** The following parameters describe the parameters for the SMD feature of NAMD.

- `SMD` < Are SMD features active >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Should SMD harmonic constraint be applied to the system. If set to `on`, then `SMDk`, `SMDFile`, `SMDVel`, and `SMDDir` must be defined. Specifying `SMDOutputFreq` is optional.

- `SMDFile` < SMD constraint reference position >
  **Acceptable Values:** UNIX filename
  **Description:** File to use for the initial reference position for the SMD harmonic constraints. All atoms in this PDB file with a nonzero value in the *occupancy* column will be tagged as SMD atoms. The coordinates of the tagged SMD atoms will be used to calculate the initial center of mass. During the simulation, this center of mass will move with velocity `SMDVel` in the direction `SMDDir`. The actual atom order in this PDB file must match that in the structure or coordinate file, since the atom number field in this PDB file will be ignored.

- `SMDk` < force constant to use in SMD simulation >
  **Acceptable Values:** positive real
  **Description:** SMD harmonic constraint force constant. Must be specified in kcal/mol/Å$^2$. The conversion factor is 1 kcal/mol = 69.479 pN Å.

- `SMDk2` < force constant for transverse direction to use in SMD simulation >
  **Acceptable Values:** positive real
  **Default Value:** 0

**Description:** SMD transverse harmonic constraint force constant. Must be specified in kcal/mol/Å$^2$. The conversion factor is 1 kcal/mol = 69.479 pN Å.

- `SMDVel` < Velocity of the SMD reference position movement >
  **Acceptable Values:** nonzero real, Å/timestep
  **Description:** The velocity of the SMD center of mass movement. Gives the absolute value.

- `SMDDir` < Direction of the SMD center of mass movement >
  **Acceptable Values:** non-zero vector
  **Description:** The direction of the SMD reference position movement. The vector does not have to be normalized, it is normalized by NAMD before being used.

- `SMDOutputFreq` < frequency of SMD output >
  **Acceptable Values:** positive integer
  **Default Value:** 1
  **Description:** The frequency in timesteps with which the current SMD data values are printed out.

## 8.9 Interactive Molecular Dynamics (IMD)

NAMD now works directly with IMD-compatible clients, such as VMD and MDAnalysis [51, 63], to allow you to view and interactively steer your simulation. With IMD enabled, you can connect to NAMD at any time during the simulation to view the current state of the system or perform interactive steering. For more information about using IMD version 3 (IMDv3) with a python package IMDClient that utilizes MDAnalysis, please refer to `https://imdclient.readthedocs.io/en/latest/usage.html#namd`. The IMDv3 implementation in NAMD was contributed by Amruthesh Thirumalaiswamy and Lawson Woods from the MDAnalysis community.

The following options are available as a part of NAMD's IMD implementation:

- `IMDon` < is IMD active? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not to listen for an IMD connection.

- `IMDversion` < IMD version number >
  **Acceptable Values:** `2` or `3`
  **Default Value:** `2`
  **Description:** Specifies the version of IMD protocol used. Currently VMD uses `2` and IMDClient (MDAnalysis) uses `3`. The detailed version 3 protocol can be found here.

- `IMDport` < port number to expect a connection on >
  **Acceptable Values:** positive integer
  **Description:** This is a free port number on the machine that node 0 is running on. This number will have to be entered into VMD.

- `IMDfreq` < timesteps between sending coordinates >
  **Acceptable Values:** positive integer
  **Description:** This allows coordinates to be sent less often, which may increase NAMD performance or be necessary due to a slow network.

- `IMDwait` < wait for an IMD connection? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** If `no`, NAMD will proceed with calculations whether a connection is present or not. If `yes`, NAMD will pause at startup until a connection is made, and pause when the connection is lost.

- `IMDignore` < ignore interactive steering forces >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** If `yes`, NAMD will ignore any steering forces generated by VMD to allow a simulation to be monitored without the possibility of perturbing it.

The following options are only available when using `IMDversion 3`:

- `IMDsendPositions` < Send positions via IMD? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `yes`
  **Description:** Specifies whether or not to send atom positions in angstroms via IMD.

- `IMDsendEnergies` < Send energies via IMD? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `yes`
  **Description:** Specifies whether or not to send energies (total, potential, Van der Waals et. al) in kilojoules/mol via IMD.

- `IMDsendTime` < Send simulation time via IMD? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Specifies whether or not to send simulation time ( timestep in picoseconds, simulation time in picoseconds and step number) via IMD.

- `IMDsendBoxDimensions` < Send box dimensions via IMD? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Specifies whether or not to send box dimensions (as a 3x3 matrix) via IMD.

- `IMDsendVelocities` < Send velocities via IMD? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Specifies whether or not to send atom velocities in angstrom/picosecond via IMD.

- `IMDsendForces` < Send forces via IMD? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `no`
  **Description:** Specifies whether or not to send atom forces kilojoules/(mol*angstrom) via IMD.

- `IMDwrapPositions` < Wrap positions sent to IMD? >
  **Acceptable Values:** `yes` or `no`
  **Default Value:** `yes`
  **Description:** Specifies whether or not to wrap atom positions sent to the IMD client.

## 8.10 Tcl Forces and Analysis

NAMD provides a limited Tcl scripting interface designed for applying forces and performing on-the-fly analysis. This interface is efficient if only a few coordinates, either of individual atoms or centers of mass of groups of atoms, are needed. In addition, information must be requested one timestep in advance. To apply forces individually to a potentially large number of atoms, use `tclBC` instead as described in Sec. 8.11. The following configuration parameters are used to enable the Tcl interface:

- `tclForces` < is Tcl interface active? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not Tcl interface is active. If it is set to `off`, then no Tcl code is executed. If it is set to `on`, then Tcl code specified in `tclForcesScript` parameters is executed.

- `tclForcesScript` < input for Tcl interface >
  **Acceptable Values:** file or {script}
  **Description:** Must contain either the name of a Tcl script file or the script itself between { and } (may include multiple lines). This parameter may occur multiple times and scripts will be executed in order of appearance. The script(s) should perform any required initialization on the Tcl interpreter, including requesting data needed during the first timestep, and define a procedure `calcforces` { } to be called every timestep.

At this point only low-level commands are defined. In the future this list will be expanded. Current commands are:

- `print <anything>`
  This command should be used instead of `puts` to display output. For example, "`print Hello World`".

- `atomid <segname> <resid> <atomname>`
  Determines atomid of an atom from its segment, residue, and name. For example, "`atomid br 2 N`".

- `addatom <atomid>`
  Request coordinates of this atom for next force evaluation, and the calculated total force on this atom for current force evaluation. Request remains in effect until `clearconfig` is called. For example, "`addatom 4`" or "`addatom [atomid br 2 N]`".

- `addgroup <atomid list>`
  Request center of mass coordinates of this group for next force evaluation. Returns a group ID which is of the form `gN` where `N` is a small integer. This group ID may then be used to find coordinates and apply forces just like a regular atom ID. Aggregate forces may then be

applied to the group as whole. Request remains in effect until `clearconfig` is called. For example, "`set groupid [addgroup { 14 10 12 }]`".

- `clearconfig`
  Clears the current list of requested atoms. After `clearconfig`, calls to `addatom` and `addgroup` can be used to build a new configuration.

- `getstep`
  Returns the current step number.

- `loadcoords <varname>`
  Loads requested atom and group coordinates (in Å) into a local array. `loadcoords` should only be called from within the `calcforces` procedure. For example, "`loadcoords p`" and "`print $p(4)`".

- `loadforces <varname>`
  Loads the forces applied in the previous timestep (in kcal mol$^{-1}$ Å$^{-1}$) into a local array. `loadforces` should only be called from within the `calcforces` procedure. For example, "`loadforces f`" and "`print $f(4)`".

- `enabletotalforces/disabletotalforces`
  Enables/disables the "`loadtotalforces`" command, described below, which is disabled by default to avoid unneeded work and communication.

- `loadtotalforces <varname>`
  Loads the total forces on each requested atom and group in the previous time step (in kcal mol$^{-1}$Å$^{-1}$) into a local array. The total force also includes external forces. Note that the "`loadforces`" command returns external forces applied by the user. Therefore, one can subtract the external force on an atom from the total force on this atom to get the pure force arising from the simulation system. Note that "`enabletotalforces`" must be called first.

- `loadmasses <varname>`
  Loads requested atom and group masses (in amu) into a local array. `loadmasses` should only be called from within the `calcforces` procedure. For example, "`loadcoords m`" and "`print $m(4)`".

- `addforce <atomid|groupid> <force vector>`
  Applies force (in kcal mol$^{-1}$ Å$^{-1}$) to atom or group. `addforce` should only be called from within the `calcforces` procedure. For example, "`addforce $groupid { 1. 0. 2. }`".

- `addenergy <energy (kcal/mol)>`
  This command adds the specified energy to the MISC column (and hence the total energy) in the energy output. For normal runs, the command does not affect the simulation trajectory at all, and only has an artificial effect on its energy output. However, it can indeed affect minimizations.

With the commands above and the functionality of the Tcl language, one should be able to perform any on-the-fly analysis and manipulation. To make it easier to perform certain tasks, some Tcl routines are provided below.

Several vector routines (`vecadd`, `vecsub`, `vecscale`) from the VMD Tcl interface are defined. Please refer to VMD manual for their usage.

The following routines take atom coordinates as input, and return some geometry parameters (bond, angle, dihedral).

- `getbond <coor1> <coor2>`
  Returns the length of the bond between the two atoms. Actually the return value is simply the distance between the two coordinates. "coor1" and "coor2" are coordinates of the atoms.

- `getangle <coor1> <coor2> <coor3>`
  Returns the angle (from 0 to 180) defined by the three atoms. "coor1", "coor2" and "coor3" are coordinates of the atoms.

- `getdihedral <coor1> <coor2> <coor3> <coor4>`
  Returns the dihedral (from -180 to 180) defined by the four atoms. "coor1", "coor2", "coor3" and "coor4" are coordinates of the atoms.

The following routines calculate the derivatives (gradients) of some geometry parameters (angle, dihedral).

- `anglegrad <coor1> <coor2> <coor3>`
  An angle defined by three atoms is a function of their coordinates: $\theta\left(\vec{r_1}, \vec{r_2}, \vec{r_3}\right)$ (in radian). This command takes the coordinates of the three atoms as input, and returns a list of $\{\frac{\partial\theta}{\partial\vec{r_1}}\ \frac{\partial\theta}{\partial\vec{r_2}}\ \frac{\partial\theta}{\partial\vec{r_3}}\}$. Each element of the list is a 3-D vector in the form of a Tcl list.

- `dihedralgrad <coor1> <coor2> <coor3> <coor4>`
  A dihedral defined by four atoms is a function of their coordinates: $\phi\left(\vec{r_1}, \vec{r_2}, \vec{r_3}, \vec{r_4}\right)$ (in radian). This command takes the coordinates of the four atoms as input, and returns a list of $\{\frac{\partial\phi}{\partial\vec{r_1}}\ \frac{\partial\phi}{\partial\vec{r_2}}\ \frac{\partial\phi}{\partial\vec{r_3}}\ \frac{\partial\phi}{\partial\vec{r_4}}\}$. Each element of the list is a 3-D vector in the form of a Tcl list.

As an example, here's a script which applies a harmonic constraint (reference position being 0) to a dihedral. Note that the "addenergy" line is not really necessary – it simply adds the calculated constraining energy to the MISC column, which is displayed in the energy output.

```
tclForcesScript {
  # The IDs of the four atoms defining the dihedral
  set aid1 112
  set aid2 123
  set aid3 117
  set aid4 115

  # The "spring constant" for the harmonic constraint
  set k 3.0

  addatom $aid1
  addatom $aid2
  addatom $aid3
  addatom $aid4
  set PI 3.1416

  proc calcforces {} {

    global aid1 aid2 aid3 aid4 k PI

    loadcoords p

    # Calculate the current dihedral
    set phi [getdihedral $p($aid1) $p($aid2) $p($aid3) $p($aid4)]
    # Change to radian
    set phi [expr $phi*$PI/180]

    # (optional) Add this constraining energy to "MISC" in the energy output
    addenergy [expr $k*$phi*$phi/2.0]

    # Calculate the "force" along the dihedral according to the harmonic constraint
    set force [expr -$k*$phi]

    # Calculate the gradients
    foreach {g1 g2 g3 g4} [dihedralgrad $p($aid1) $p($aid2) $p($aid3) $p($aid4)] {}

    # The force to be applied on each atom is proportional to its
    # corresponding gradient
    addforce $aid1 [vecscale $g1 $force]
    addforce $aid2 [vecscale $g2 $force]
    addforce $aid3 [vecscale $g3 $force]
    addforce $aid4 [vecscale $g4 $force]
  }
}
```

## 8.11 Tcl Boundary Forces

While the `tclForces` interface described above is very flexible, it is only efficient for applying forces to a small number of pre-selected atoms. Applying forces individually to a potentially large number of atoms, such as applying boundary conditions, is much more efficient with the `tclBC` facility described below.

- `tclBC` < are Tcl boundary forces active? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not Tcl interface is active. If it is set to `off`, then no Tcl code is executed. If it is set to `on`, then Tcl code specified in the `tclBCScript` parameter is executed.

- `tclBCScript` < input for Tcl interface >
  **Acceptable Values:** {script}

**Description:** Must contain the script itself between { and } (may include multiple lines). This parameter may occur only once. The script(s) should perform any required initialization on the Tcl interpreter and define a procedure `calcforces <step> <unique> [args...]` to be called every timestep.

- `tclBCArgs` < extra args for tclBC calcforces command >
  **Acceptable Values:** {args...}
  **Description:** The string (or Tcl list) provided by this option is appended to the tclBC calcforces command arguments. This parameter may appear multiple times during a run in order to alter the parameters of the boundary potential function.

The script provided in `tclBCScript` and the `calcforces` procedure it defines are executed in multiple Tcl interpreters, one for every processor that owns patches. These `tclBC` interpreters do not share state with the Tcl interpreter used for `tclForces` or config file parsing. The `calcforces` procedure is passed as arguments the current timestep, a "unique" flag which is non-zero for exactly one Tcl interpreter in the simulation (that on the processor of patch zero), and any arguments provided to the most recent `tclBCArgs` option. The "unique" flag is useful to limit printing of messages, since the command is invoked on multiple processors.

The `print`, `vecadd`, `vecsub`, `vecscale`, `getbond`, `getangle`, `getdihedral`, `anglegrad`, and `dihedralgrad` commands described under `tclForces` are available at all times.

The `wrapmode <mode>` command, available in the `tclBCScript` or the `calcforces` procedure, determines how coordinates obtained in the calcforces procedure are wrapped around periodic boundaries. The options are:

- `patch`, (default) the position in NAMD's internal patch data structure, requires no extra calculation and is almost the same as `cell`
- `input`, the position corresponding to the input files of the simulation
- `cell`, the equivalent position in the unit cell centered on the `cellOrigin`
- `nearest`, the equivalent position nearest to the `cellOrigin`

The following commands are available from within the `calcforces` procedure:

- `nextatom`
  Sets the internal counter to a new atom and return 1, or return 0 if all atoms have been processed (this may even happen the first call). This should be called as the condition of a while loop, i.e., `while {[nextatom]} { ... }` to iterate over all atoms. One one atom may be accessed at a time.
- `dropatom`
  Excludes the current atom from future iterations on this processor until `cleardrops` is called. Use this to eliminate extra work when an atom will not be needed for future force calculations. If the atom migrates to another processor it may reappear, so this call should be used only as an optimization.
- `cleardrops`
  All available atoms will be iterated over by `nextatom` as if `dropatom` had never been called.

- `getcoord`
  Returns a list {x y z} of the position of the current atom wrapped in the periodic cell (if there is one) in the current wrapping mode as specified by `wrapmode`.

- `getcell`
  Returns a list of 1–4 vectors containing the cell origin (center) and as many basis vectors as exist, i.e., {{`ox oy oz`} {`ax ay az`} {`bx by bz`} {`cx cy cz`}}. It is more efficient to set the wrapping mode than to do periodic image calculations in Tcl.

- `getmass`
  Returns the mass of the current atom.

- `getcharge`
  Returns the charge of the current atom.

- `getid`
  Returns the 1-based ID of the current atom.

- `addforce` {`<fx> <fy> <fz>`}
  Adds the specified force to the current atom for this step.

- `addenergy <energy>`
  Adds potential energy to the BOUNDARY column of NAMD output.

As an example, these spherical boundary condition forces:

```
sphericalBC        on
sphericalBCcenter  0.0,0.0,0.0
sphericalBCr1      48
sphericalBCk1      10
sphericalBCexp1    2
```

Are replicated in the following script:

```
tclBC on
tclBCScript {
  proc veclen2 {v1} {
    foreach {x1 y1 z1} $v1 { break }
    return [expr $x1*$x1 + $y1*$y1 + $z1*$z1]
  }

  # wrapmode input
  # wrapmode cell
  # wrapmode nearest
  # wrapmode patch ;# the default

  proc calcforces {step unique R K} {
    if { $step % 20 == 0 } {
      cleardrops
      # if $unique { print "clearing dropped atom list at step $step" }
    }
    set R [expr 1.*$R]
    set R2 [expr $R*$R]
    set tol 2.0
    set cut2 [expr ($R-$tol)*($R-$tol)]

    while {[nextatom]} {
      # addenergy 1 ; # monitor how many atoms are checked
      set rvec [getcoord]
      set r2 [veclen2 $rvec]
      if { $r2 < $cut2 } {
        dropatom
        continue
      }
      if { $r2 > $R2 } {
        # addenergy 1 ; # monitor how many atoms are affected
        set r [expr sqrt($r2)]
        addenergy [expr $K*($r - $R)*($r - $R)]
        addforce [vecscale $rvec [expr -2.*$K*($r-$R)/$r]]
      }
    }
  }
}

tclBCArgs {48.0 10.0}
```

## 8.12 External Program Forces

This feature allows an external program to be called to calculate forces at every force evaluation, taking all atom coordinates as input.

- `extForces` < Apply external program forces? >
  **Acceptable Values:** yes or no
  **Default Value:** no
  **Description:** Specifies whether or not external program forces are applied.

- `extForcesCommand` < Force calculation command >
  **Acceptable Values:** UNIX shell command
  **Description:** This string is the argument to the system() function at every forces evaluation and should read coordinates from the file specified by extCoordFilename and write forces to the file specified by extForceFilename.

- `extCoordFilename` < Temporary coordinate file >
  **Acceptable Values:** UNIX filename
  **Description:** Atom coordinates are written to this file, which should be read by the extForcesCommand. The format is one line of “atomid charge x y z” for every atom followed by three lines with the periodic cell basis vectors “a.x a.y a.z”, “b.x b.y b.z”, and “c.x c.y

c.z". The atomid starts at 1 (not 0). For best performance the file should be in /tmp and not on a network-mounted filesystem.

- `extForceFilename` < Temporary force file >
  **Acceptable Values:** UNIX filename
  **Description:** Atom forces are read from this file after extForcesCommand in run. The format is one line of "atomid replace fx fy fz" for every atom followed by the energy on a line by itself and then, optionally, three lines of the virial "v.xx v.xy v.xz", "v.yx v.yy v.yz", "v.zx v.zy v.zz" where, e.g., v.xy = - fx * y for a non-periodic force. The atomid starts at 1 (not 0) and all atoms must be present and in order. The energy is added to the MISC output field. The replace flag should be 1 if the external program force should replace the forces calculated by NAMD for that atom and 0 if the forces should be added. For best performance the file should be in /tmp and not on a network-mounted filesystem.

## 8.13 Multiple Time Stepping User Defined Forces

User defined forces are applied at every time step, unless the user specifies otherwise. The following options provide a variety of ways to control how frequently all user defined forces are applied. These apply to : colvars, tclforces, and any other forces controlled by the GlobalMaster infrastructure.

These apply a Multiple Time Stepping approach to globalmaster forces, where the user controls the length of the new outer step and how the forces are integrated (scaled, or stale) using the parameters defined below.

**Please note that this feature is only available in GPU-resident version of NAMD (NAMD3).**

- `globalMasterFrequency` < time steps between invoking global master >
  **Acceptable Values:** positive integer
  **Default Value:** 1
  **Description:** Controls the number of steps between invocations of globalmaster clients (i.e., colvars) therefore globalmaster clients will be called on step % `globalMasterFrequency` == 0. The step number reported to the clients will be standard NAMD time step. `globalMasterStep` will contain step / `globalMasterFrequency` so that clients can distinguish between the two. **Note:** values larger than 1 are supported only when `GPUresident` is `on`, and will raise an error otherwise.

- `globalMasterScaleByFrequency` < scale globalmaster forces by frequency? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Will apply the `globalMasterFrequency` value as a factor on the force contribution from the globalmaster clients when the new computation is triggered.

- `globalMasterStaleForces` < are stale forces on or off? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Will use the last computed value from the globalmaster clients for the force integration on the off steps, instead of the RESPA approach of applying the force once, as a single impulse. This should not be used if `globalMasterScaleByFrequency` is in use.

# 9 Collective Variable-based Calculations (Colvars)

## 9.1 Overview of Colvars module and links to documentation

The collective variables module (Colvars) is an independently-developed library for enhanced sampling simulations, licensed under the GNU lesser general public license (https://www.gnu.org/licenses/lgpl-3.0.en.html) and distributed together with NAMD. Full documentation and examples for this software, including any syntax and features specific to this version of NAMD, can be accessed at the following links:

- (HTML) https://colvars.github.io/namd-3.0/colvars-refman-namd.html
- (PDF) https://colvars.github.io/namd-3.0/colvars-refman-namd.pdf

See in particular the section regarding compatibility between versions of NAMD:
https://colvars.github.io/namd-3.0/colvars-refman-namd.html#sec:colvars_config_changes

When using the Colvars module please cite the following publication, alongside other publications for specific features as listed in the code's documentation and in the usage summary printed when running a Colvars-enabled NAMD simulation.

G. Fiorin, M. L. Klein, and J. Hénin, *Molecular Physics* **111** (22-23), 3345-3362 (2013).
https://doi.org/10.1080/00268976.2013.813594

Please ask any usage questions through the NAMD mailing list, and any development questions through the Colvars GitHub repository at https://github.com/Colvars/colvars.

# 10 Alchemical Free Energy Methods[1]

Alchemical free energy calculations model the physically impossible but computationally realizable process of gradually mutating a subset of atoms of a system from one state to another, through a series of intermediate steps. Two alternative methods for alchemical calculation of free energies from molecular dynamics simulation are available in NAMD: Free energy perturbation (FEP) and thermodynamic integration (TI).

## 10.1 Theoretical Background

Free energy differences can be obtained through four different routes: (i) probability densities, (ii) free energy perturbation, (iii) thermodynamic integration, or (iv) nonequilibrium work approaches [18]. Within NAMD, alchemical transformations are modeled following the second and the third routes, both of which rely upon the use of a general extent parameter often referred to as the coupling parameter [6, 45, 33, 34] for the description of chemical changes in the molecular systems between the reference and the target states.

### 10.1.1 The dual–topology paradigm

In a typical alchemical transformation setup involving the alteration of one chemical species into an alternate one in the course of the simulation, the atoms in the molecular topology can be classified into three groups, (i) a group of atoms that do not change during the simulation — *e.g.* the environment, (ii) the atoms describing the reference state, $a$, of the system, and (iii) the atoms that correspond to the target state, $b$, at the end of the alchemical transformation. The atoms representative of state $a$ should *never* interact with those of state $b$ throughout the MD simulation. Such a setup, in which atoms of both the initial and the final states of the system are present in the molecular topology file — *i.e.* the `psf` file — is characteristic of the so–called "dual topology" paradigm [22, 59, 2]. The hybrid Hamiltonian of the system is a function of the general extent parameter, $\lambda$, which connects smoothly state $a$ to state $b$. In the simplest case, such a connection may be achieved by linear combination of the corresponding Hamiltonians:

$$\mathcal{H}(\mathbf{x}, \mathbf{p}_x; \lambda) = \mathcal{H}_0(\mathbf{x}, \mathbf{p}_x) + \lambda \mathcal{H}_b(\mathbf{x}, \mathbf{p}_x) + (1 - \lambda)\mathcal{H}_a(\mathbf{x}, \mathbf{p}_x) \quad (35)$$

where $\mathcal{H}_a(\mathbf{x}, \mathbf{p}_x)$ describes the interaction of the group of atoms representative of the reference state, $a$, with the rest of the system. $\mathcal{H}_b(\mathbf{x}, \mathbf{p}_x)$ characterizes the interaction of the target topology, $b$, with the rest of the system. $\mathcal{H}_0(\mathbf{x}, \mathbf{p}_x)$ is the Hamiltonian describing those atoms that do not undergo any transformation during the MD simulation.

For instance, in the point mutation of an alanine side chain into that of glycine, by means of a free energy calculation — either free energy perturbation or thermodynamic integration, the topology of both the methyl group of alanine and the hydrogen borne by the $C_\alpha$ in glycine co–exist throughout the simulation (see Figure 5), yet without actually seeing each other.

The energy and forces are defined as a function of $\lambda$, in such a fashion that the interaction of the methyl group of alanine with the rest of the protein is effective at the beginning of the simulation, *i.e.* $\lambda = 0$, while the glycine $C_\alpha$ hydrogen atom does not interact with the rest of the protein, and *vice versa* at the end of the simulation, *i.e.* $\lambda = 1$. For intermediate values of $\lambda$, both the alanine

[1]The features described in this section were contributed by Surjit B. Dixit, Christophe Chipot (Nancy Université, Université Henri Poincaré, France), Floris Buelens (Institute of Structural and Molecular Biology, University of London, Birkbeck, UK), and Christopher Harrison (University of Illinois, Urbana, IL USA).

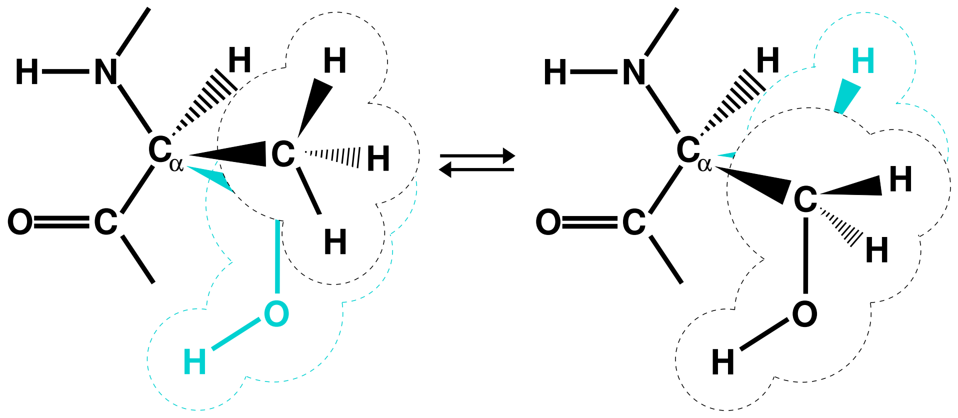


Figure 5: Dual topology description for an alchemical simulation. Case example of the mutation of alanine into serine. The lighter color denotes the non–interacting, alternate state.

and the glycine side chains participate in nonbonded interactions with the rest of the protein, scaled on the basis of the current value of $\lambda$. It should be clearly understood that these side chains never interact with each other.

It is noteworthy that end points of alchemical transformations carried out in the framework of the dual–topology paradigm have been shown to be conducive to numerical instabilities from molecular dynamics simulations, often coined as "end–point catastrophes". These scenarios are prone to occur when $\lambda$ becomes close to 0 or 1, and incoming atoms instantly appear where other particles are already present, which results in a virtually infinite potential as the interatomic distance tends towards 0. Such "end–point catastrophes" can be profitably circumvented by introducing a so–called soft–core potential [5, 43], aimed at a gradual scaling of the short–range nonbonded interactions of incoming atoms with their environment, as shown in Equation 36. What is really being modified is the value of a coupling parameter ($\lambda_{\rm LJ}$ or $\lambda_{\rm elec}$) that scales the interactions — *i.e.*, if set to 0, the latter are off; if set to 1, they are on — in lieu of the actual value of $\lambda$ provided by the user.

$$V_{\rm NB}(r_{ij}) = \lambda_{\rm LJ}\varepsilon_{ij}\left[\left(\frac{R_{ij}^{\rm min\ 2}}{r_{ij}^2+\delta(1-\lambda_{\rm LJ})}\right)^6 - \left(\frac{R_{ij}^{\rm min\ 2}}{r_{ij}^2+\delta(1-\lambda_{\rm LJ})}\right)^3\right] + \lambda_{\rm elec}\frac{q_iq_j}{\varepsilon_1 r_{ij}} \qquad (36)$$

It is also worth noting that the free energy calculation does not alter intermolecular bonded potentials, *e.g.* bond stretch, valence angle deformation and torsions, in the course of the simulation. In calculations targeted at the estimation of free energy differences between two states characterized by distinct environments — *e.g.* a ligand, bound to a protein in the first simulation, and solvated in water, in the second — as is the case for most free energy calculations that make use of a thermodynamic cycle, perturbation of intramolecular terms may, by and large, be safely avoided [10]. This property is controlled by the `alchDecouple` keyword described in

### 10.1.2 Free Energy Perturbation

Within the FEP framework [6, 17, 18, 23, 36, 45, 78, 81, 89], the free energy difference between two alternate states, $a$ and $b$, is expressed by:

$$\Delta A_{a\to b} = -\frac{1}{\beta}\,\ln \left\langle \exp\left\{-\beta\left[\mathcal{H}_b(\mathbf{x},\mathbf{p}_x) - \mathcal{H}_a(\mathbf{x},\mathbf{p}_x)\right]\right\}\right\rangle_a \tag{37}$$

Here, $\beta^{-1} \equiv k_B T$, where $k_B$ is the Boltzmann constant, $T$ is the temperature. $\mathcal{H}_a(\mathbf{x},\mathbf{p}_x)$ and $\mathcal{H}_b(\mathbf{x},\mathbf{p}_x)$ are the Hamiltonians describing states $a$ and $b$, respectively. $\langle\cdots\rangle_a$ denotes an ensemble average over configurations representative of the initial, reference state, $a$.

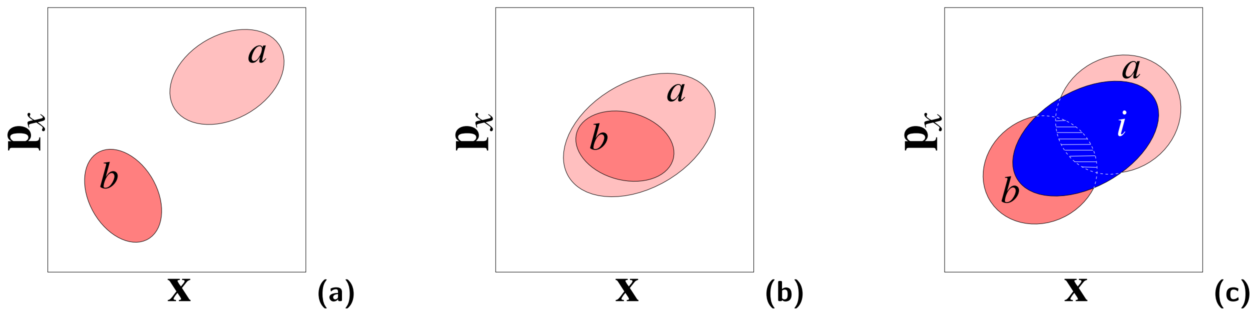


Figure 6: Convergence of an FEP calculation. If the ensembles representative of states $a$ and $b$ are too disparate, equation (37) will not converge **(a)**. If, in sharp contrast, the configurations of state $b$ form a subset of the ensemble of configurations characteristic of state $a$, the simulation is expected to converge **(b)**. The difficulties reflected in case **(a)** may be alleviated by the introduction of mutually overlapping intermediate states that connect $a$ to $b$ **(c)**. It should be mentioned that in practice, the kinetic contribution, $\mathcal{T}(\mathbf{p}_x)$, is assumed to be identical for state $a$ and state $b$.

Convergence of equation (37) implies that low–energy configurations of the target state, $b$, are also configurations of the reference state, $a$, thus resulting in an appropriate overlap of the corresponding ensembles — see Figure 6. Transformation between the two thermodynamic states is replaced by a series of transformations between non–physical, intermediate states along a well–delineated pathway that connects $a$ to $b$. This pathway is characterized by the general extent parameter [6, 33, 34, 45], $\lambda$, that makes the Hamiltonian and, hence, the free energy, a continuous function of this parameter between $a$ and $b$:

$$\Delta A_{a\to b} = -\frac{1}{\beta}\sum_{i=1}^{N} \ln \left\langle \exp\left\{-\beta\left[\mathcal{H}(\mathbf{x},\mathbf{p}_x;\lambda_{i+1}) - \mathcal{H}(\mathbf{x},\mathbf{p}_x;\lambda_i)\right]\right\}\right\rangle_i \tag{38}$$

Here, $N$ stands for the number of intermediate stages, or "windows" between the initial and the final states — see Figure 6.

### 10.1.3 Thermodynamic Integration

An alternative to the perturbation formula for free energy calculation is Thermodynamic Integration (TI). With the TI method, the free energy is given as [34, 77, 21]:

$$\Delta A = \int_0^1 \left\langle \frac{\partial \mathcal{H}(\mathbf{x},\mathbf{p}_x;\lambda)}{\partial\lambda}\right\rangle_\lambda d\lambda \tag{39}$$

In the multi-configuration thermodynamic integration approach [77] implemented in NAMD, $\langle \partial\mathcal{H}(\mathbf{x}, \mathbf{p}_x; \lambda)/\partial\lambda \rangle_\lambda$, the ensemble average of the derivative of the internal energy with respect to $\lambda$, is collected for a series of discrete $\lambda$ values and written to `tiOutFile`. These values are analyzed by the separately distributed script `NAMD_ti.pl`, which performs the integration of individual energy components and reports back the total $\Delta A$ values for the transformation.

## 10.2 Implementation of the free energy methods in NAMD

The procedures implemented in NAMD are particularly adapted for performing free energy calculations that split the $\lambda$ reaction path into a number of non–physical, intermediate states, or "windows". Separate simulations can be started for each window. Alternatively, the TCL scripting ability of NAMD can be employed advantageously to perform the complete simulation in a single run. An example, making use of such a script, is supplied at the end of this section. However, **the setup of sequential alchemical trsnaformations can be simplified** by calling the script library `fep.tcl`, found in the `lib/alch` directory of the NAMD distribution. This library provides two helper procedures, `runFEP` to run a series of evenly spaced windows, and `runFEPlist` to specify a list of $\lambda$ values to be sampled.

The following keywords can be used to run alchemical free energy calculations, whether FEP or TI.

- `alch` < Is an alchemical transformation to be performed? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Turns on alchemical transformation methods in NAMD.

- `alchType` < Which method is to be employed for the alchemical transformation? >
  **Acceptable Values:** `fep` or `ti`
  **Default Value:** `ti`
  **Description:** Turns on Hamiltonian scaling and ensemble averaging for alchemical FEP or TI.

- `alchWCA` < Turn on/off Weeks-Chandler-Andersen (WCA) decomposition. >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** When active, WCA decomposition changes the lambda dependence of the van der Waals perturbation following the repulsion/dispersion scheme proposed by Deng and Roux [20]. For example, for appearing atoms, all interactions are still fully coupled at `alchLambda` = `alchVdwLambdaEnd`, but repulsive components are instead fully coupled according to the new `alchRepLambdaEnd` keyword. No dispersive interactions (including terms from `LJcorrection`) are coupled until the repulsive interactions are fully coupled. By virtue of the formulation, `alchVdwShiftCoeff` does not have any effect in this scheme and any non-zero values are ignored. Note that this scheduling is completely separate from electrostatic coupling and the two may overlap in any way desired (this may not be stable!). In order to achieve the exact decoupling scheme proposed by Deng and Roux, one ought to set `alchRepLambdaEnd` < `alchVdwLambdaEnd` = `alchElecLambdaStart` < 1. **This scheme has only been widely tested when a single alchemical group is being used.** Due to current limitations, this scheme is not available when `alchType` is set to `ti`.

- `alchLambda` < Current value of the coupling parameter >
  **Acceptable Values:** positive decimal between 0.0 and 1.0
  **Description:** The coupling parameter value determining the progress of the perturbation for FEP or TI. This parameter is unnecessary when using the `runFEP` procedure of `fep.tcl`.

- `alchLambda2` < Forward projected value of the coupling parameter >
  **Acceptable Values:** positive decimal between 0.0 and 1.0
  **Description:** The `lambda2` value corresponds to the coupling parameter to be used for sampling in the next window. The free energy difference between `alchLambda2` and `alchLambda` is calculated. Through simulations at progressive values of `alchLambda` and `alchLambda2` the total free energy difference may be determined. This parameter is unnecessary when using the `runFEP` procedure of `fep.tcl`.

- `alchLambdaIDWS` < Backward value of the coupling parameter for Interleaved Double-Wide Sampling >
  **Acceptable Values:** decimal between 0.0 and 1.0, negative to disable
  **Description:** Setting this parameter between 0 and 1 activates Interleaved Double-Wide Sampling (IDWS), whereby the target lambda value alternates between `alchLambda2` and `alchLambdaIDWS`. The switch occurs every `fullElectFrequency` steps if defined, or `nonbondedFrequency` otherwise. Setting this parameter to a negative value (including between run statements) disables IDWS. When IDWS is active, the alchemy output file contains `FepEnergy` line headers for the forward energy differences, and `FepE_Back` for backward energy differences. FEP free energy estimates given in output are based on forward data only. The fepout file can be parsed using the Python module alchemlyb. Alternately, it can be postprocessed with the python script `deinterleave_idws.py`, found in the `lib/alch` directory of the NAMD distribution. This tool produces separate fepout files for the forward and backwards samples, suitable for computing e.g. a Bennett's Acceptance Ratio (BAR) estimate of the free energy difference. When using the `runFEP` or `runFEPlist` procedure of `fep.tcl`, IDWS can be enabled simply by adding a `true` flag to the argument list. Note that when IDWS is enabled, `alchOutFreq` must be a multiple of `fullElectFrequency`.

- `alchEquilSteps` < Number of equilibration steps in a window, prior to data collection >
  **Acceptable Values:** positive integer less than `numSteps` or `run`
  **Default Value:** 0
  **Description:** In each window `alchEquilSteps` steps of equilibration can be performed before ensemble averaging is initiated. The output also contains the data gathered during equilibration and is meant for analysis of convergence properties of the alchemical free energy calculation.

- `alchFile` < `pdb` file with perturbation flags >
  **Acceptable Values:** filename
  **Default Value:** coordinates
  **Description:** `pdb` file to be used for indicating the status of all atoms pertaining to the system, with respect to the alchemical transformation. If this parameter is not declared specifically, then the `pdb` file specified by `coordinates` is utilized for this information.

- `alchCol` < Column in the `alchFile` that carries the perturbation flag >
  **Acceptable Values:** X, Y, Z, O or B

**Default Value:** B
**Description:** Column of the `pdb` file to use for retrieving the status of each atom, *i.e.* a flag that indicates which atom will be perturbed in the course of the alchemical transformation. A value of `-1` in the specified column indicates that the atom will vanish as $\lambda$ moves from 0 to 1, whereas a value of `1` indicates that it will grow.

- `alchOutFreq` < Frequency of free energy output in time–steps >
**Acceptable Values:** positive integer
**Default Value:** 5
**Description:** Every `alchOutFreq` number of MD steps, the output file `alchOutFile` is updated by dumping energies that are used for ensemble averaging. This variable could be set to `1` to include all the configurations for ensemble averaging. Yet, it is recommended to update `alchOutFile` energies at longer intervals to avoid large files containing highly correlated data, unless a post–treatment, *e.g.* Bennett's acceptance ratio (BAR) [3] or simple overlap sampling (SOS) [41], is to be performed.

- `alchOutFile` < Alchemical free energy output filename >
**Acceptable Values:** filename
**Default Value:** `outfilename`
**Description:** An output file named `alchOutFile`, containing the FEP energies, or `tiOutFile`, containing the TI derivatives, dumped every `alchOutFreq` steps.

- `alchVdwShiftCoeff` < Soft-core van der Waals radius-shifting coefficient >
**Acceptable Values:** positive decimal
**Default Value:** 5
**Description:** This is a radius-shifting coefficient of $\lambda$ that is used to construct the modified vdW interactions during alchemical free energy calculations. Providing a positive value for `alchVdWShiftCoeff` ensures that the vdW potential is finite everywhere for small values of $\lambda$, which significantly improves the accuracy and convergence of FEP and TI calculations, and also prevents overlapping particles from making the simulation unstable. During FEP and TI, assuming $\lambda = 0$ denotes an absence of interaction, the interatomic distances used in the Lennard-Jones potential are shifted according to [5, 43]: $r^2 \rightarrow r^2 + \texttt{alchVdWShiftCoeff} \times (1 - \lambda)$

- `alchElecLambdaStart` < Value of $\lambda$ to introduce electrostatic interactions >
**Acceptable Values:** positive decimal
**Default Value:** 0.5
**Description:** In order to avoid the so–called "end-point catastrophes", it is crucial to avoid situations where growing particles overlap with existing particles with an unbounded interaction potential, which would approach infinity as the interaction distance approaches zero [5, 18]. One possible route for avoiding overlap of unbounded electrostatic potentials consists of allowing a bounded (soft-core) vdW potential, using a positive value of `alchVdWShiftCoeff`, to repel first all overlapping particles at low values of $\lambda$. As $\lambda$ increases, once the particles are repelled, it becomes safe to turn on FEP or TI electrostatics.

In the current implementation, the electrostatic interactions of an exnihilated, or appearing, particle are linearly coupled to the simulation over the $\lambda$ value range of `alchElecLambdaStart` – 1.0. At $\lambda$ values less than or equal to the user-defined value of `alchElecLambdaStart`, electrostatic interactions of the exnihilated particle are fully decoupled from the simulation.

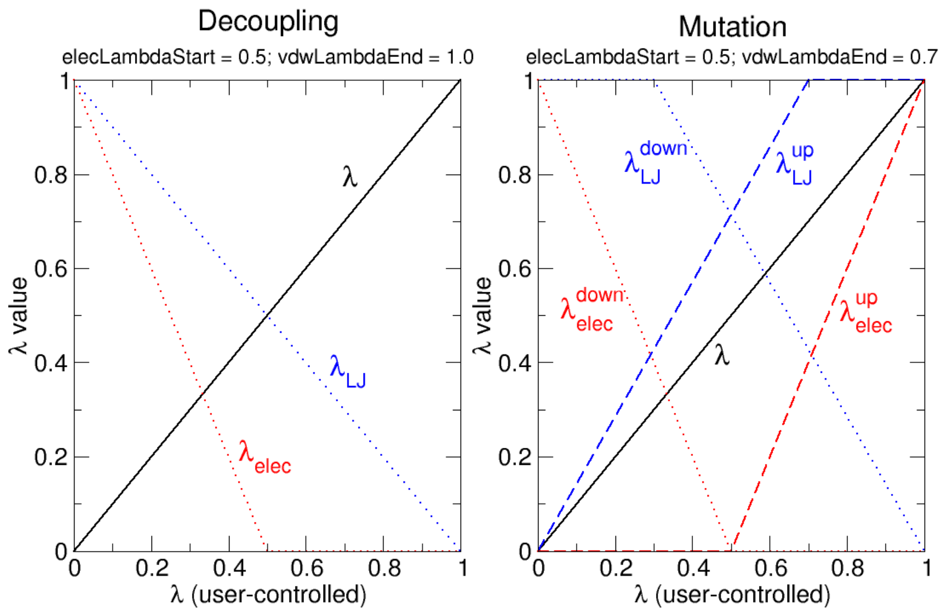


Figure 7: Relationship of user-defined $\lambda$ to coupling of electrostatic or vdW interactions to a simulation, given specific values of `alchElecLambdaStart` or `alchVdwLambdaEnd`.

Coupling of electrostatic interactions then increases linearly for increasing values of $\lambda$ until $\lambda$=1.0, at which point electrostatic interactions of the exnihilated particle are fully coupled to the simulation.

For annihilated, or vanishing, particles the electrostatic interactions are linearly decoupled from the simulation over the $\lambda$ value range of 0 – (1.0 - `alchElecLambdaStart`). At $\lambda$=0 electrostatic interactions are fully coupled to the simulation, and then linearly decreased with increasing $\lambda$ such that at $\lambda$ values greater than or equal to (1.0 - `alchElecLambdaStart`) electrostatic interactions are completely decoupled from the simulation. Two examples, shown in Figure 7, describe the relationship between the user-defined value of $\lambda$ and the coupling of electrostatic or vdW interactions to the simulation.

- `alchVdwLambdaEnd` < Value of $\lambda$ to cancel van der Waals interactions >
  **Acceptable Values:** positive decimal
  **Default Value:** 1.0
  **Description:** If the `alchElecLambdaStart` option is used, it may also be desirable to separate the scaling of van der Waals and electrostatic interactions. `alchVdwLambdaEnd` sets the value of $\lambda$ above which all vdW interactions are fully enabled for exnihilated particles.

  For an exnihilated particle, vdW interactions are fully decoupled at $\lambda$=0. The coupling of vdW interactions to the simulation is then increased with increasing values of $\lambda$ such that at values of $\lambda$ greater than or equal to `alchVdwLambdaEnd` the vdW interactions of the exnihilated particle are fully coupled to the simulation.

  For an annihilated particle, vdW interactions are completely coupled to the simulation for $\lambda$ values between 0 and (1 - `alchVdwLambdaEnd`). Then, vdW interactions of the annihilated particle are linearly decoupled over the range of $\lambda$ values between (1 - `alchVdwLambdaEnd`) and 1.0. VdW interactions are only fully decoupled when $\lambda$ reaches 1.0.

**New as of version 2.12:** The energy and virial terms added by `LJcorrection on` or `LJcorrectionAlt on` are now also controlled by the vdW $\lambda$ schedule. The average Lennard-Jones $A$ and $B$ coefficients are computed separately at both endpoints and then coupled linearly. In most practical situations the energy difference is extremely negligible, but this is more theoretically sound than the old behavior of averaging both endpoints together. However, the kinetic energy component of the virial *does* still count the endpoints together, as if annihilated alchemical atoms were an ideal gas. Again, this is likely quite negligible, nor is it clear that this should be treated specially.

- `alchRepLambdaEnd` < Value of $\lambda$ to cancel van der Waals repulsive interactions >
  **Acceptable Values:** positive decimal
  **Default Value:** 0.5
  **Description:** This parameter is only used when `alchWCA` is `on`, in which case it MUST be less than or equal to `alchVdwLambdaEnd`. For appearing atoms, this marks both the value at which repulsive interactions are completely coupled and at which dispersive interactions beging to become coupled (but are still zero).

- `alchBondLambdaEnd` < Value of $\lambda$ to cancel bonded interactions >
  **Acceptable Values:** positive decimal
  **Default Value:** 0.0
  **Description:** **New as of version 2.12** Bonded terms involving alchemical atoms may now also be scaled on a schedule similar to vdW interactions. Although this is more theoretically sound in many situations, this behavior is off by default.

- `alchBondDecouple` < Enable scaling of bonded terms within alchemical groups >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** This is essentially a bonded term analogue of the `alchDecouple` keyword. Setting `alchBondDecouple on`, causes bonded terms between alchemical atoms *in the same group* to also be scaled. This means that alchemical atoms are annihilated into ideal gas atoms instead of ideal gas molecules. In this case it is recommended to use the approach of Axelsen and Li [2] by way of the `extraBonds` keyword. Using `alchBondDecouple on` is strictly necessary if it is desired to have the endpoint (potential) energies of a dual-topology PSF match those of a non-alchemical PSF.

- `alchDecouple` < Disable scaling of nonbonded interactions within alchemical partitions >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** With `alchDecouple` set to `on`, only nonbonded interactions of perturbed, incoming and outgoing atoms with their environment are scaled, while interactions within the subset of perturbed atoms are preserved. On the contrary, if `alchDecouple` is set to `off`, interactions within the perturbed subset of atoms are also scaled and contribute to the cumulative free energy. In most calculations, intramolecular annihilation free energies are not particularly informative, and decoupling ought to be preferred. Under certain circumstances, it may, however, be desirable to scale intramolecular interactions, provided that the latter are appropriately accounted for in the thermodynamic cycle [18].

## 10.3 Examples of input files for running alchemical free energy calculations

**Note:** In this section the lambda values are specified manually. For sequential sampling of lambda values, it is simpler to call the `runFEP` or `runFEPlist` procedure of `fep.tcl`. See the comments in that file for instructions.

The first example illustrates the use of TCL scripting for running an alchemical transformation with the FEP feature of NAMD. In this calculation, $\lambda$ is changed continuously from 0 to 1 by increments of $\delta\lambda = 0.1$.

```
alch           On
alchType       fep
alchFile       ion.fep
alchCol        X
alchOutfile    ion.fepout
alchOutFreq    5
alchEquilSteps 5000
```

Enable alchemical simulation module
Set alchemical method to FEP
File containing the information about growing/shrinking atoms described in column `X`.
Output file containing the free energy.
Frequency at which `fepOutFreq` is updated.
Number of equilibration steps per $\lambda$–state.

```
set Lambda0    0.0
set dLambda    0.1
```

Starting value of $\lambda$.
Increment of $\lambda$, *i.e.* $\delta\lambda$.

```
while {$Lambda0 < 1.0} {
 alchLambda $Lambda0
 set Lambda0 [expr $Lambda0 + $dLambda]
 alchLambda2 $Lambda0
 run  10000
}
```

TCL script to increment $\lambda$:
(1) set `lambda` value;
(2) increment $\lambda$;
(3) set `lambda2` value;
(4) run 10,000 MD steps.

The user should be reminded that by setting `run 10000`, 10,000 MD steps will be performed, which includes the preliminary `fepEquilSteps` equilibration steps. This means that here, the ensemble average of equation (38) will be computed over 5,000 MD steps.

Alternatively, $\lambda$–states may be declared explicitly, avoiding the use of TCL scripting:

```
alchLambda      0.0
alchLambda2     0.1
run             10000
```

(1) set `alchLambda` value;
(2) set `alchLambda2` value;
(3) run 10,000 MD steps.

This option is generally preferred to set up windows of diminishing widths as $\lambda \to 0$ or 1 — a way to circumvent end–point singularities caused by appearing atoms that may clash with their surroundings.

The following second input is proposed for the measuring via TI the free energy of a particle insertion.

```
alch           On                ;# Enable alchemical simulation module
alchType       ti                ;# Set method to thermodynamic integration
alchFile       ion.alch.pdb      ;# PDB file with perturbation flags
alchCol        B                 ;# Perturbation flags in Beta column
alchOutfile    ion.ti.out
alchOutFreq    5
alchEquilSteps 5000

alchVdWShiftCoeff    1           ;# Enable soft-core vdW potential
alchElecLambdaStart  0.1         ;# Introduce electrostatics for lambda > 0.1
```

```
alchLambda 0
run 10000
alchLambda 0.00001
run 10000
alchLambda 0.0001
run 10000
alchLambda 0.001
run 10000
alchLambda 0.01
run 10000

set Lambda           0.1

while {$Lambda <= 0.9} {
  alchLambda $Lambda
  run 10000
  set Lambda [expr $Lambda + 0.1]
}

alchLambda 0.99
run 10000
alchLambda 0.999
run 10000
alchLambda 0.9999
run 10000
alchLambda 0.99999
run 10000
alchLambda 1
run 10000
```

Robust sampling of the free energy of particle insertion is enabled by the use of soft-core van der Waals scaling with the `alchVdWShiftCoeff` parameter, delayed introduction of electrostatics with a non-zero `alchElecLambdaStart` value, and very gradual scaling of $\lambda$ towards its end points.

## 10.4 Description of a free energy calculation output

### 10.4.1 Free Energy Perturbation

When running FEP, the `alchOutFile` contains electrostatic and van der Waals energy data calculated for `alchLambda` and `alchLambda2`, written every `alchOutFreq` steps. The column `dE` is the energy difference of the single configuration, `dE_avg` and `dG` are the instantaneous ensemble average of the energy and the calculated free energy at the time step specified in column 2, respectively. The temperature is specified in the penultimate column. Upon completion of `alchEquilSteps` steps, the calculation of `dE_avg` and `dG` is restarted. The accumulated net free energy change is written at each lambda value and at the end of the simulation.

Whereas the FEP module of NAMD supplies free energy differences determined from equation (37), the wealth of information available in `alchOutFile` may be utilized profitably to explore different routes towards the estimation of $\Delta A$. Both BAR and SOS methods, which combine advantageously *direct* and *reverse* transformations to improve convergence and accuracy of the calculation, represent relevant alternatives to brute–force application of the FEP formula [41].

Within the SOS framework, the free energy difference between states $\lambda_i$ and $\lambda_{i+1}$ is expressed as:

$$\exp(-\beta \Delta A_{i \to i+1}) = \frac{\left\langle \exp\left\{-\frac{\beta}{2}\left[\mathcal{H}(\mathbf{x}, \mathbf{p}_x; \lambda_{i+1}) - \mathcal{H}(\mathbf{x}, \mathbf{p}_x; \lambda_i)\right]\right\}\right\rangle_i}{\left\langle \exp\left\{-\frac{\beta}{2}\left[\mathcal{H}(\mathbf{x}, \mathbf{p}_x; \lambda_i) - \mathcal{H}(\mathbf{x}, \mathbf{p}_x; \lambda_{i+1})\right]\right\}\right\rangle_{i+1}} \tag{40}$$

and can be readily used with the statistical information provided by the forward and the backward runs.

### 10.4.2 Thermodynamic Integration

When running TI free energy calculations, the `elec_dU/dl`, `vdW_dU/dl`, and `bond_dU/dl` values reported in `alchOutFile` are the derivatives of the internal energy with respect to the scaling factors for each interaction type (*i.e.* electrostatics, etc.). `dU/dl` values are locally averaged over the last `alchOutFreq` steps. Cumulative averages for each component are reported alongside in the `_avg` columns.

The electrostatic, vdW, and bond values are separated following a partition scheme — that is, the "appearing" and the "disappearing" atoms are accounted for separately. "Partition 1" contains those atoms whose interactions are switched up as $\lambda$ increases — *i.e.* flagged with `1` in the `alchFile`. "Partition 2" represents those atoms whose interactions are switched down as $\lambda$ increases — *i.e.* flagged with `-1`. $\Delta A$ values for each component are obtained by integrating from $\lambda = 0$ to 1 using the respective `ELEC / VDW / BOND LAMBDA` listed for each partition after the title.

**New as of version 2.12:** The output in `alchOutFile` has been extensively revised and now more closely matches the NAMD standard output. Additional accounting for bonded term scaling is now also included.

The choice of $\lambda$ values will depend on the application, but in general it is important to examine the shape of the curve to ensure that sampling is adequate to give a good estimate of the integral. In particular, it will be necessary to sample more finely towards the end points in order to accurately account for the strong repulsive van der Waals forces encountered when inserting particles into a system (see Figure 8).

## 10.5 Hybrid single–dual topology approach for relative binding free energy calculation of ligand to receptor

An effective hybrid single–dual topology protocol is designed for the calculation of relative binding affinities of small ligands to a receptor. The protocol was developed as an expansion of the existing dual-topology relative alchemical free energy calculations [30], for either free energy perturbation or thermodynamic integration. In this protocol, the alchemical end states are represented as two separate molecules sharing a common substructure identified through maximum structural mapping. Within the substructure, an atom-to-atom correspondence is established, and each pair of corresponding atoms are holonomically constrained to share identical coordinates at all time throughout the simulation, as shown in Figure 9. The forces are projected and combined at each step for propagation.

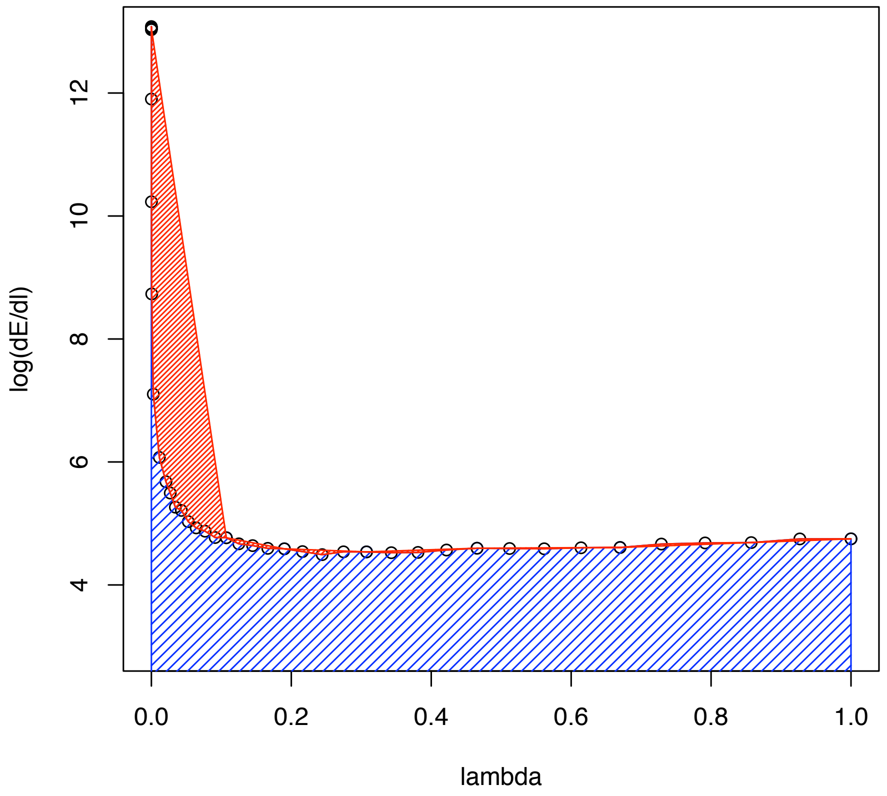


Figure 8: Sample TI data ($log(\langle\frac{\partial U}{\partial\lambda}\rangle)$ against $\lambda$). The blue shaded area shows the integral with fine sampling close to the end point. The red area shows the difference when $\lambda$ values are more sparse. In this example, insufficient sampling before $\lambda \simeq$0.1 can result in a large overestimation of the integral. Beyond $\simeq$0.2, sparser sampling is justified as dE/d$\lambda$ is not changing quickly.

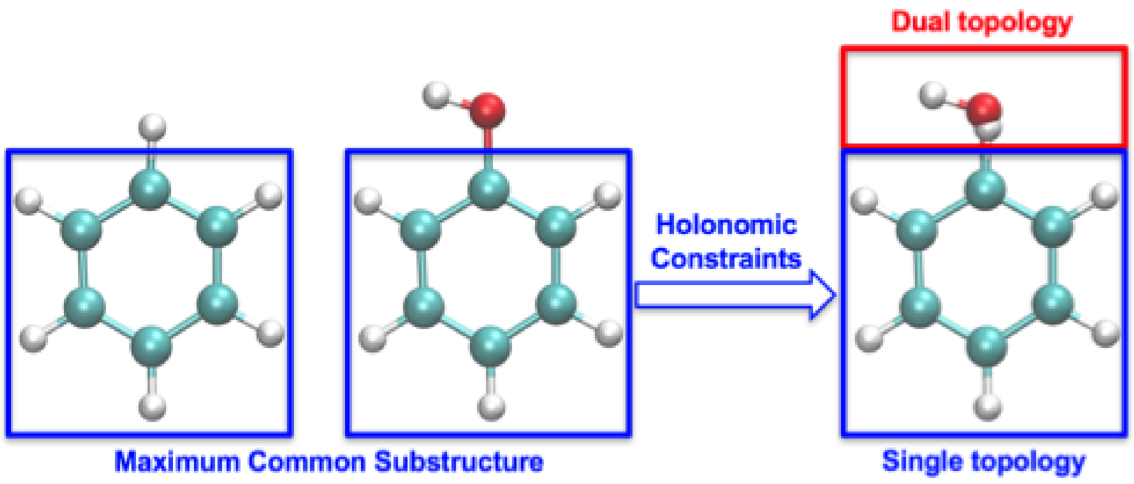


Figure 9: Hybrid single–dual topology setup generated by applying holonomic constraints on the maximum common substructure.

As it is based on the existing dual topology setup, the major input files including PDB, PSF and alchemical flag files adopt the same format as before, with two more partitions accommodating the initial/end states of the single topology region. Determining the common substructure generally requires a special setup tool to determine the maximum structural mapping that generate the partitions present in the PDB and PSF files. The dual-topology setup also implements Shobana bonded terms to support the ring topology change problem [69], for which a separate input file lists all unperturbed bonded terms on a ring. The current implementation supports both relative solvation free energies of small molecules and relative binding affinities of drug compounds to proteins. To enhance sampling of the dual-topology region, the alchemical calculations can be carried out within a replica-exchange MD scheme supported by the multiple-copy algorithm module of NAMD, with periodic attempted swapping of the thermodynamic coupling parameter $\lambda$ betwen neighboring states.

It needs to be noted that the protocol is currently implemented only on CPU, with a GPU implementation in development. VMD does not yet provide a hybrid topology setup tool, and CHARMM-GUI is testing a beta version (that is not yet available online) to automatically generate all input files for NAMD. For the time being, users can utilize an alternative hybrid structure preparation tool, such as FESetup or AmberTools, and then manually convert the generated CHARMM-formatted input files into a format that can be read by NAMD.

The following keywords enable hybrid single–dual topology simulation.

- `singleTopology` < Enable hybrid single–dual topology? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Enable the use of hybrid single–dual topology for alchemical transformation, which extends the default dual topology setup.

- `sdBondScaling` < Are Shobana terms enabled? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Enable the use of selected Shobana terms, the unperturbed bond, angle, and dihedral terms on a transformed ring, that remove the possible artificial effects of dummy atoms. For a more detailed elucidation, please see reference [69].

- `unperturbedBondFile` < file listing unperturbed bonded terms >
  **Acceptable Values:** filename
  **Description:** This must be defined if `sdBondScaling` is `on`. The file lists the selected unperturbed bond, angle, and dihedral terms that remove the possible artificial effects of dummy atoms. When `sdBondScaling` is `off`, the file will be skipped.

## 11 Accelerated Sampling Methods

### 11.1 Accelerated Molecular Dynamics

Accelerated molecular dynamics (aMD) [25] is an enhanced-sampling method that improves the conformational space sampling by reducing energy barriers separating different states of a system. The method modifies the potential energy landscape by raising energy wells that are below a certain threshold level, while leaving those above this level unaffected. As a result, barriers separating adjacent energy basins are reduced, allowing the system to sample conformational space that cannot be easily accessed in a classical MD simulation.

Please include the following two references in your work using the NAMD implementation of aMD:

- Accelerated Molecular Dynamics: A Promising and Efficient Simulation Method for Biomolecules, D. Hamelberg, J. Mongan, and J. A. McCammon. *J. Chem. Phys.*, 120:11919-11929, 2004.
- Implementation of Accelerated Molecular Dynamics in NAMD, Y. Wang, C. Harrison, K. Schulten, and J. A. McCammon, *Comp. Sci. Discov.*, 4:015002, 2011.

#### 11.1.1 Theoretical background

In the original form of aMD [25], when the system's potential energy falls below a threshold energy, $E$, a boost potential is added, such that the modified potential, $V^*(\mathbf{r})$, is related to the original potential, $V(\mathbf{r})$, via

$$V^*(\mathbf{r}) = V(\mathbf{r}) + \Delta V(\mathbf{r}), \tag{41}$$

where $\Delta V(\mathbf{r})$ is the boost potential,

$$\Delta V(\mathbf{r}) = \begin{cases} 0 & V(\mathbf{r}) \geq E \\ \frac{(E-V(\mathbf{r}))^2}{\alpha+E-V(\mathbf{r})} & V(\mathbf{r}) < E. \end{cases} \tag{42}$$

As shown in the following figure, the threshold energy $E$ controls the portion of the potential surface affected by the boost, while the acceleration factor $\alpha$ determines the shape of the modified potential. Note that $\alpha$ cannot be set to zero, otherwise the derivative of the modified potential is discontinuous.

From an aMD simulation, the ensemble average, $\langle A \rangle$, of an observable, $A(\mathbf{r})$, can be calculated using the following reweighting procedure:

$$\langle A \rangle = \frac{\langle A(\mathbf{r}) \exp(\beta \Delta V(\mathbf{r})) \rangle^*}{\langle \exp(\beta \Delta V(\mathbf{r})) \rangle^*}, \tag{43}$$

in which $\beta$=$1/k_B T$, and $\langle ... \rangle$ and $\langle ... \rangle^*$ represent the ensemble average in the original and the aMD ensembles, respectively.

Currently, aMD can be applied in three modes in NAMD: aMDd, aMDT, and aMDdual [84]. The boost energy is applied to the dihedral potential in the aMDd mode (the default mode), and to the total potential in the aMDT mode. In the dual boost mode (aMDdual) [24], two independent boost energies are applied, one on the dihedral potential and the other on the (Total - Dihedral) potential.

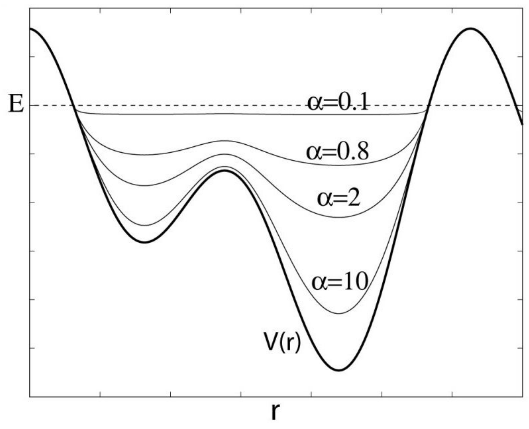


Figure 10: Schematics of the aMD method. When the original potential (thick line) falls below a threshold energy $E$ (dashed line), a boost potential is added. The modified energy profiles (thin lines) have smaller barriers separating adjacent energy basins.

### 11.1.2 NAMD parameters

The following parameters are used to enable accelerated MD:

- `accelMD` < Is accelerated molecular dynamics active? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies if accelerated MD is active.

- `accelMDdihe` < Apply boost to dihedrals? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `on`
  **Description:** Only applies boost to the dihedral potential. By default, `accelMDdihe` is turned on and the boost energy is applied to the dihedral potential of the simulated system. When `accelMDdihe` is turned off, aMD switches to the `accelMDT` mode, and the boost is applied to the total potential.

- `accelMDE` < Threshold energy $E$ >
  **Acceptable Values:** Real number
  **Description:** Specifies the threshold energy $E$ in the aMD equations.

- `accelMDalpha` < Acceleration factor $\alpha$ >
  **Acceptable Values:** Positive real number
  **Description:** Specifies the acceleration factor $\alpha$ in the aMD equations.

- `accelMDdual` < Use dual boost mode? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** When `accelMDdual` is on, aMD switches to the dual boost mode. Two independent boost potentials will be applied: one to the dihedral potential that is controlled by

the parameters `accelMDE` and `accelMDalpha`, and a second to the (Total - Dihedral) potential that is controlled by the `accelMDTE` and `accelMDTalpha` parameters described below.

- `accelMDTE` < Threshold energy $E$ in the dual boost mode >
  **Acceptable Values:** Real number
  **Description:** Specifies the threshold energy $E$ used in the calculation of boost energy for the (Total - Dihedral) potential. This option is only available when `accelMDdual` is turned on.

- `accelMDTalpha` < Acceleration factor $\alpha$ in the dual boost mode >
  **Acceptable Values:** Positive real number
  **Description:** Specifies the acceleration factor $\alpha$ used in the calculation of boost energy for the (Total - Dihedral) potential. This option is only available when `accelMDdual` is turned on.

- `accelMDFirstStep` < First accelerated MD step >
  **Acceptable Values:** Zero or positive integer
  **Default Value:** 0
  **Description:** Accelerated MD will only be performed when the current step is equal to or higher than `accelMDFirstStep`, and equal to or lower than `accelMDLastStep`. Otherwise regular MD will be performed.

- `accelMDLastStep` < Last accelerated MD step >
  **Acceptable Values:** Zero or positive integer
  **Default Value:** 0
  **Description:** Accelerated MD will only be performed when the current step is equal to or higher than `accelMDFirstStep`, and equal to or lower than `accelMDLastStep`. Otherwise regular MD will be performed. Note that the accelMDLastStep parameter only has an effect when it is positive. When accelMDLastStep is set to zero (the default), aMD is 'open-ended' and will be performed till the end of the simulation.

- `accelMDOutFreq` < Frequency in steps of aMD output >
  **Acceptable Values:** Positive integer
  **Default Value:** 1
  **Description:** An aMD output line will be printed to the log file at the frequency specified by `accelMDOutFreq`. The aMD output will contain the boost potential ($dV$) at the current timestep, the average boost potential ($dVAVG$) since the last aMD output, and various potential energy values at the current timestep. The boost potential $dV$ can be used to reconstruct the ensemble average described earlier.

## 11.2 Gaussian Accelerated Molecular Dynamics

Gaussian accelerated molecular dynamics (GaMD) [50] is a type of accelerated molecular dynamics (aMD) calculation. It is an enhanced sampling method that works by adding a harmonic boost potential to smoothen the system's potential energy surface. By constructing a boost potential that follows Gaussian distribution, accurate reweighting of the GaMD simulations is achieved using cumulant expansion to the second order.

Please include the following two references in your work using the NAMD implementation of GaMD:

- Gaussian Accelerated Molecular Dynamics: Unconstrained Enhanced Sampling and Free Energy Calculation, Y. Miao, V. Feher, and J. A. McCammon. *J. Chem. Theory Comput.*, 11:3584-3595, 2015.
- Gaussian Accelerated Molecular Dynamics in NAMD, Y.T. Pang, Y. Miao, Y. Wang, and J. A. McCammon, *J. Chem. Theory Comput.*, 13:9-19, 2017.

### 11.2.1 Theoretical background

GaMD enhances conformational sampling of biomolecules by adding a harmonic boost potential to smoothen the system's potential energy surface [50], as illustrated below:

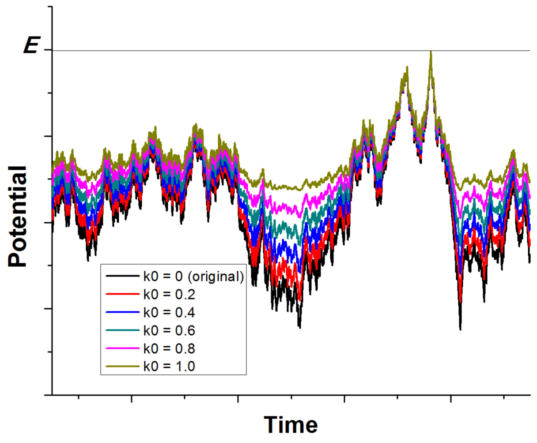


Figure 11: Schematic illustration of GaMD. When the threshold energy $E$ is set to the maximum potential ($iE = 1$ mode), the system's potential energy surface is smoothened by adding a harmonic boost potential that follows a Gaussian distribution. The coefficient $k_0$, which falls in the range of $0 - 1.0$, determines the magnitude of the applied boost potential.

Consider a system with $N$ atoms at positions $\mathbf{r} = \{\mathbf{r}_1, \cdots, \mathbf{r}_N\}$. When the system's potential energy $V(\mathbf{r})$ is lower than a threshold energy $E$, the following boost potential is added:

$$V^*(\mathbf{r}) = V(\mathbf{r}) + \Delta V(\mathbf{r}), \tag{44}$$

where $\Delta V(\mathbf{r})$ is the boost potential,

$$\Delta V(\mathbf{r}) = \begin{cases} \frac{1}{2}k\left(E - V(\mathbf{r})\right)^2, & V(\mathbf{r}) < E \\ 0, & V(\mathbf{r}) \geq E. \end{cases} \tag{45}$$

where $k$ is the harmonic force constant.

As explained in reference [50], the two adjustable parameters $E$ and $k$ are automatically determined by the following three criteria. First, $\Delta V$ should not change the relative order of the biased potential values, i.e., for any two arbitrary potential values $V_1(\mathbf{r})$ and $V_2(\mathbf{r})$ found on the original energy surface, if $V_1(\mathbf{r}) < V_2(\mathbf{r})$, then one should have $V_1^*(\mathbf{r}) < V_2^*(\mathbf{r})$. Second, the difference between potential energy values on the smoothened energy surface should be smaller than that of the original, i.e., if $V_1(\mathbf{r}) < V_2(\mathbf{r})$, then one should have $V_2^*(\mathbf{r}) - V_1^*(\mathbf{r}) < V_2(\mathbf{r}) - V_1(\mathbf{r})$. By combining the above two criteria and plugging in the formula of $V^*(\mathbf{r})$ and $\Delta V$, one obtains

$$V_{\max} \leq E \leq V_{\min} + \frac{1}{k} \tag{46}$$

where $V_{\rm min}$ and $V_{\rm max}$ are the system's minimum and maximum potential energies. To ensure that Eqn. (46) is valid, $k$ needs to satisfy: $k \leq \frac{1}{V_{\rm max} - V_{\rm min}}$. Define $k \equiv k_0 \cdot \frac{1}{V_{\rm max} - V_{\rm min}}$, then $0 < k_0 \leq 1$. Third, the standard deviation of $\Delta V$ needs to be small enough (i.e., narrow distribution) to ensure accurate reweighting using cumulant expansion to the second order: $\sigma_{\Delta V} = k\,(E - V_{\rm avg})\,\sigma_V \leq \sigma_0$, where $V_{\rm avg}$ and $\sigma_V$ are the average and standard deviation of the system's potential energies, $\sigma_{\Delta V}$ is the standard deviation of $\Delta V$, while $\sigma_0$ is a user-specified upper limit (e.g., $10k_BT$) in order to achieve accurate reweighting.

**iE = 1 mode:** When $E$ is set to $E = V_{\rm max}$ according to Eqn. (46), $k_0$ is calculated as:

$$k_0 = \min(1.0, k_0') = \min\left(1.0, \frac{\sigma_0}{\sigma_V} \cdot \frac{V_{\rm max} - V_{\rm min}}{V_{\rm max} - V_{\rm avg}}\right) \tag{47}$$

**iE = 2 mode:** Alternatively, when $E$ is set to $E = V_{\rm min} + \frac{1}{k}$, $k_0$ is calculated as:

$$k_0 = k_0'' \equiv \left(1 - \frac{\sigma_0}{\sigma_V} \cdot \frac{V_{\rm max} - V_{\rm min}}{V_{\rm avg} - V_{\rm min}}\right) \tag{48}$$

If $k_0''$ obtained from the above equation is smaller than 0 or greater than 1, then $k_0$ will be calculated using Eqn. (47).

For more details on GaMD and the corresponding reweighting using cumulant expansion, see reference [50][58].

### 11.2.2 NAMD parameters

Same as aMD, three modes are available for applying boost potential in GaMD: (1) boosting the dihedral energy only, (2) boosting the total potential energy, and (3) boosting both the dihedral and total potential energy (i.e., "dual-boost").

Some parameters from aMD, including: `accelMD`, `accelMDdihe`, `accelMDdual`, `accelMDFirstStep`, `accelMDLastStep` and `accelMDOutFreq` are shared by GaMD (see Section 11.1 for details). The following is a list of input parameters unique to a GaMD run:

- `accelMDG` < Is Gaussian accelerated MD on? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether Gaussian accelerated MD (GaMD) is on. Only available when `accelMD` is on.

- `accelMDGiE` < Flag to set the threshold energy for adding boost potential >
  **Acceptable Values:** 1 or 2
  **Default Value:** 1
  **Description:** Specifies how the threshold energy $E$ is set in GaMD. A value of 1 indicates that the threshold energy $E$ is set to its lower bound $E = V_{\rm max}$. A value of 2 indicates that the threshold energy is set to its upper bound $E = V_{\rm min} + (V_{\rm max} - V_{\rm min})/k_0$.

- `accelMDGcMDPrepSteps` < Number of preparatory cMD steps >
  **Acceptable Values:** Zero or Positive integer
  **Default Value:** 200,000
  **Description:** The number of preparatory conventional MD (cMD) steps in GaMD. This value should be smaller than `accelMDGcMDSteps` (see below). Potential energies

are not collected for calculating the values of $V_{\max}$, $V_{\min}$, $V_{\text{avg}}$, $\sigma_V$ during the first accelMDGcMDPrepSteps.

- accelMDGcMDSteps < Number of total cMD steps >
**Acceptable Values:** Zero or Positive integer
**Default Value:** 1,000,000
**Description:** The number of total cMD steps in GaMD. With accelMDGcMDPrepSteps $< t <$ accelMDGcMDSteps, $V_{\max}$, $V_{\min}$, $V_{\text{avg}}$, $\sigma_V$ are collected and at $t =$ accelMDGcMDSteps, $E$ and $k_0$ are computed.

- accelMDGEquiPrepSteps < Number of preparatory equilibration steps in GaMD >
**Acceptable Values:** Zero or Positive integer
**Default Value:** 200,000
**Description:** The number of preparatory equilibration steps in GaMD. This value should be smaller than accelMDGEquiSteps (see below). With accelMDGcMDSteps $< t <$ accelMDGEquiPrepSteps + accelMDGcMDSteps, GaMD boost potential is applied according to $E$ and $k_0$ obtained at $t =$ accelMDGcMDSteps.

- accelMDGEquiSteps < Number of total equilibration steps in GaMD >
**Acceptable Values:** Zero or Positive integer
**Default Value:** 1,000,000
**Description:** The number of total equilibration steps in GaMD. With accelMDGEquiPrepSteps + accelMDGcMDSteps $< t <$ accelMDGEquiSteps + accelMDGcMDSteps, GaMD boost potential is applied, and $E$ and $k_0$ are updated every step.

- accelMDGStatWindow < Number of steps to calculate average and standard deviation in GaMD >
**Acceptable Values:** Integer
**Default Value:** -1
**Description:** The number of simulation steps used to calculate the average and standard deviation of potential energies, as well as the frequency of recalculating the boost potential during equilibration steps. When it is set to a negative number, all the steps throughout the cMD and equilibration stage (except the preparatory steps) will be used to calculate the average and standard deviation without resetting, and the boost potential will be updated every step during equilibration steps. When used, it is recommended to be set to about 4 times the total number of atoms in the system. Note that accelMDGcMDPrepSteps, accelMDGcMDSteps, accelMDGEquiPrepSteps and accelMDGEquiSteps need to be multiples of accelMDGStatWindow.

- accelMDGSigma0P < Upper limit of the standard deviation of the total boost potential in GaMD >
**Acceptable Values:** Positive real number
**Default Value:** 6.0 (kcal/mol)
**Description:** Specifies the upper limit of the standard deviation of the total boost potential. This option is only available when accelMDdihe is off or when accelMDdual is on.

- accelMDGSigma0D < Upper limit of the standard deviation of the dihedral boost potential in GaMD >

**Acceptable Values:** Positive real number
**Default Value:** 6.0 (kcal/mol)
**Description:** Specifies the upper limit of the standard deviation of the dihedral boost potential. This option is only available when `accelMDdihe` or `accelMDdual` is on.

- `accelMDGRestart` < Flag to restart GaMD simulation >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether the current GaMD simulation is the continuation of a previous run. If this option is turned on, the GaMD restart file specified by `accelMDGRestartFile` (see below) will be read.

- `accelMDGRestartFile` < Name of GaMD restart file >
  **Acceptable Values:** UNIX filename
  **Description:** A GaMD restart file that stores the current number of steps, maximum, minimum, average and standard deviation of the dihedral and/or total potential energies (depending on the `accelMDdihe` and `accelMDdual` parameters). This file is saved automatically every `restartfreq` steps. If `accelMDGRestart` is turned on, this file will be read and the simulation will restart from the point where the file was written.

## 11.3 Solute Scaling and REST2

Solute scaling improves sampling efficiency by scaling the intramolecular potential energy of a protein to lower barriers separating different confirmations [83]. The potential is scaled based on a parameter $\beta$,

$$U^{\mathrm{SS}}(\vec{r}) = \beta U_{\mathrm{pp}}(\vec{r}) + \sqrt{\beta} U_{\mathrm{pw}}(\vec{r}) + U_{\mathrm{ww}}(\vec{r}), \tag{49}$$

with $U_{\mathrm{pp}}$ denoting protein–protein interactions, $U_{\mathrm{pw}}$ denoting protein–water interactions, and $U_{\mathrm{ww}}$ denoting water–water interactions, effectively "heating" the protein's interatomic interactions whenever $\beta < 1$. The NAMD implementation is made efficient by rescaling the force field parameters for the affected atoms [32]. In particular, this parameter scaling approach makes the calculation compatible with existing CUDA force kernels.

The NAMD implementation provides additional flexibility to solute scaling by allowing different scaling factors for electrostatics, van der Waals, and bonded interactions, as described in the following section. Solute scaling can be combined with replica exchange to produce a powerful sampling enhancement method that is highly transferable and provides higher efficiency than traditional temperature exchange methods. In the literature, this replica exchange solute scaling method is known as REST2, due to its improvement of the earlier REST (replica exchange solute tempering) method that directly scaled the temperature of the solute. Sample files are available in directory `lib/replica/REST2`, with script file `lib/replica/REST2/rest2_remd.namd` demonstrating use of solute scaling with multiple replicas.

### 11.3.1 NAMD parameters

The following parameters are used to control solute scaling:

- `soluteScaling` < Is replica exchange solute tempering enabled? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`

**Description:** Specifies whether or not REST2 is enabled. If set on, then `soluteScaling` must also be set.

- `soluteScalingFactor` < Solute scaling factor >
  **Acceptable Values:** non-negative
  **Default Value:** 1.0
  **Description:** This option sets the scaling factor $\beta$, and is typically set lower than 1 to reduce potential energy barriers for the solute. In a replica-exchange run, this option and the related options `soluteScalingFactorCharge` and `soluteScalingFactorVdw` serve also as Tcl scripting commands to update the corresponding parameters during a simulation. *After using* `reinitatoms`*it is highly recommended that these commands are repeated even if the corresponding parameters are not being updated, to preserve their intended value.*

- `soluteScalingFactorCharge` < Solute scaling factor for electrostatics >
  **Acceptable Values:** non-negative
  **Default Value:** `soluteScalingFactor`
  **Description:** Scaling factor applied to just the electrostatics interactions. If not specified, this is set to `soluteScalingFactor`.

- `soluteScalingFactorVdw` < Solute scaling factor for van der Waals >
  **Acceptable Values:** non-negative
  **Default Value:** `soluteScalingFactor`
  **Description:** Scaling factor applied to just the van der Waals interactions. If not specified, this is set to `soluteScalingFactor`.

- `soluteScalingFile` < PDB file with scaling flags >
  **Acceptable Values:** UNIX filename
  **Default Value:** `coordinates`
  **Description:** PDB file used to flag solute atoms for scaling. If undefined, this defaults to the coordinate PDB file.

- `soluteScalingCol` < Column of PDB file >
  **Acceptable Values:** `X`, `Y`, `Z`, `O`, or `B`
  **Default Value:** `O`
  **Description:** Column of the PDB file used to flag solute atoms for scaling. If undefined, this defaults to the `O` (occupancy) column. A value of 1.0 marks the atom for scaling.

- `soluteScalingAll` < Apply scaling also to bond and angle interactions? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** If set on, `scalingFactor` is applied also to bond and angle interactions. Otherwise, `scalingFactor` is applied only to dihedral, improper, and crossterm interactions.

## 11.4 Adaptive Tempering

Adaptive tempering is akin to a single-copy replica exchange method for dynamically updating the simulation temperature. The temperature $T$ is a new random variable in the range $[Tmin, Tmax]$ that is governed by the equation $dE/dT = E - E(T) - 1/T + sqrt(2)T\xi$, where $\xi$ is Gaussian white noise. The effect is that when the potential energy for a given structure is lower than the (so far calculated) average energy, the temperature is lowered. Conversely when the current energy is higher than the average energy, the temperature is raised. The effect is faster conformational sampling to find minimum energy structures. The method is implemented exactly as described by Zhang and Ma in J. Chem. Phys. 132, 244101 (2010) (using Equation 18 of their paper to calculate the average energy at a given temperature from the histogram of energies).

The dynamic temperature is realized either by changing the temperature of the Langevin thermostat or by velocity rescaling.

### 11.4.1 NAMD parameters

The following parameters are used to adaptive tempering:

- `adaptTempMD` < Is adaptive tempering active? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not adaptive tempering is used. If set to on then the following parameters are required to be set: either all of (`adaptTempTmin`, `adaptTempTmax`, `adaptTempBins`, `adaptTempDt`) or `adaptTempInFile` (but not both).

- `adaptTempFreq` < steps between temperature updates >
  **Acceptable Values:** Positive integers
  **Default Value:** 10
  **Description:** The number of steps between temperature updates. Note that the potential energy at the current is calculated and added to the temperature-energy histogram at every step.

- `adaptTempTmin` < minimum temperature (K) >
  **Acceptable Values:** Positive real number
  **Description:** Sets the minimum temperature to be used in the simulation.

- `adaptTempTmax` < maximum temperature (K) >
  **Acceptable Values:** Positive real number
  **Description:** Sets the maximum temperature to be used in the simulation.

- `adaptTempBins` < number of temperature bins >
  **Acceptable Values:** Positive integer
  **Default Value:** 1000
  **Description:** Sets the number of bins to subdivide the temperature range. Each bin stores the average energy for the given temperature

- `adaptTempDt` < stepsize for temperature updates >
  **Acceptable Values:** Positive real numbers
  **Default Value:** $10^{-4}$
  **Description:** Integration timestep for temperature updates. This is unrelated to the simulation timestep and only scales the size of the step taken in temperature space every `adaptTempFreq` steps.

- `adaptTempInFile` < adaptive tempering input filename >
  **Acceptable Values:** UNIX filename

**Description:** The input file containing restart information for adaptive tempering (written out by `adaptTempRestartFile`).

- `adaptTempRestartFile` < adaptive tempering restart filename >
  **Acceptable Values:** UNIX filename
  **Description:** The file to write out restart information for adaptive tempering.

- `adaptTempRestartFreq` < steps between writing restart file >
  **Acceptable Values:** Positive integer
  **Description:** Frequency of writing restart file.

- `adaptTempLangevin` < send temperature updates to langevin thermostat? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `on`
  **Description:** Setting this to on will cause the langevin thermostat to use the updated temperatures from adaptive tempering. Note that either one of adaptTempLangevin or adaptTempRescaling have to be on.

- `adaptTempRescaling` < send temperature to velocity rescaling thermostat? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `on`
  **Description:** Setting this to on will cause the veloctiy rescaling thermostat to use the updated temperatures from adaptive tempering. Note that either one of adaptTempLangevin or adaptTempRescaling have to be on.

- `adaptTempOutFreq` < steps between printing adaptive tempering output >
  **Acceptable Values:** Positive integers
  **Default Value:** 10
  **Description:** The number of timesteps between printing adaptive tempering output to the log file.

- `adaptTempFirstStep` < step to start adaptive tempering >
  **Acceptable Values:** Non-negative integers
  **Default Value:** 0
  **Description:** The first timestep from which adaptive tempering will be run.

- `adaptTempLastStep` < step to stop adaptive tempering >
  **Acceptable Values:** Positive integers
  **Description:** The last timestep to apply adaptive tempering.

- `adaptTempCgamma` < dynamic bin averaging constant >
  **Acceptable Values:** Non-negative real number
  **Default Value:** 0.1
  **Description:** The calculation of the mean energy for a given bin is weighted by a factor of 1 - Cgamma / samples to damp out old statistics. Setting Cgamma to zero restores the use of a standard arithmetic mean to calculate the mean energy for each bin.

- `adaptTempRandom` < assign random temperature if we step out of range? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`

**Description:** If set to on and the temperature steps out of [`adaptTempTmin`, `adaptTempTmax`], a random temperature in that range is assigned. Otherwise the previous temperature is kept.

## 11.5 Locally enhanced sampling

Locally enhanced sampling (LES) [64, 70, 71] increases sampling and transition rates for a portion of a molecule by the use of multiple non-interacting copies of the enhanced atoms. These enhanced atoms experience an interaction (electrostatics, van der Waals, and covalent) potential that is divided by the number of copies present. In this way the enhanced atoms can occupy the same space, while the multiple instances and reduces barriers increase transition rates.

### 11.5.1 Structure generation

To use LES, the structure and coordinate input files must be modified to contain multiple copies of the enhanced atoms. `psfgen` provides the `multiply` command for this purpose. NAMD supports a maximum of 255 copies, which should be sufficient.

Begin by generating the complete molecular structure and guessing coordinates as described in Sec. 4. As the last operation in your script, prior to writing the psf and pdb files, add the `multiply` command, specifying the number of copies desired and listing segments, residues, or atoms to be multiplied. For example, `multiply 4 BPTI:56 BPTI:57` will create four copies of the last two residues of segment BPTI. You must include all atoms to be enhanced in a single `multiply` command in order for the bonded terms in the psf file to be duplicated correctly. Calling `multiply` on connected sets of atoms multiple times will produce unpredictable results, as may running other commands after `multiply`.

The enhanced atoms are duplicated exactly in the structure—they have the same segment, residue, and atom names. They are distinguished only by the value of the B (beta) column in the pdb file, which is 0 for normal atoms and varies from 1 to the number of copies created for enhanced atoms. The enhanced atoms may be easily observed in VMD with the atom selection `beta != 0`.

### 11.5.2 Simulation

In practice, LES is a simple method used to increase sampling; no special output is generated. The following parameters are used to enable LES:

- `les` < is locally enhanced sampling active? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not LES is active.

  `lesFactor` < number of LES images to use >
  **Acceptable Values:** positive integer equal to the number of images present
  **Description:** This should be equal to the factor used in `multiply` when creating the structure. The interaction potentials for images is divided by `lesFactor`.

- `lesReduceTemp` < reduce enhanced atom temperature? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Enhanced atoms experience interaction potentials divided by `lesFactor`.

This allows them to enter regions that would not normally be thermally accessible. If this is not desired, then the temperature of these atoms may be reduced to correspond with the reduced potential. This option affects velocity initialization, reinititialization, reassignment, and the target temperature for langevin dynamics. Langevin dynamics is recommended with this option, since in a constant energy simulation energy will flow into the enhanced degrees of freedom until they reach thermal equilibrium with the rest of the system. The reduced temperature atoms will have reduced velocities as well, unless `lesReduceMass` is also enabled.

- `lesReduceMass` < reduce enhanced atom mass? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Used with `lesReduceTemp` to restore velocity distribution to enhanced atoms. If used alone, enhanced atoms would move faster than normal atoms, and hence a smaller timestep would be required.

- `lesFile` < PDB file containing LES flags >
  **Acceptable Values:** UNIX filename
  **Default Value:** `coordinates`
  **Description:** PDB file to specify the LES image number of each atom. If this parameter is not specified, then the PDB file containing initial coordinates specified by `coordinates` is used.

- `lesCol` < column of PDB file containing LES flags >
  **Acceptable Values:** `X`, `Y`, `Z`, `O`, or `B`
  **Default Value:** `B`
  **Description:** Column of the PDB file to specify the LES image number of each atom. This parameter may specify any of the floating point fields of the PDB file, either X, Y, Z, occupancy, or beta-coupling (temperature-coupling). A value of 0 in this column indicates that the atom is not enhanced. Any other value should be a positive integer less than `lesFactor`.

## 11.6 Replica exchange simulations

The `lib/replica/` directory contains Tcl scripts that implement replica exchange both for parallel tempering (temperature exchange) and umbrella sampling (exchanging collective variable biases). This replaces the old Tcl server and socket connections driving a separate NAMD process for every replica used in the simulation.

**A NAMD build based on Charm++ 6.5.0 or later using one of the "LRTS" (low-level runtime system) machine layers is required!** Current LRTS machine layers include mpi, netlrts, verbs (for InfiniBand), gemini_gni-crayxe, gni-crayxc, and pamilrts-bluegeneq.

Only temperature-exchange simulations are described below. To employ replicas for umbrella sampling you will need to understand this material, collective variable-based calculations (Sec. 9), and basic Tcl programming to adapt the examples in `lib/replica/umbrella/` and `lib/replica/umbrella2d/` until further documentation and a tutorial are available.

This implementation is designed to be modified to implement exchanges of parameters other than temperature or via other temperature exchange methods. The scripts should provide a good starting point for any simulation method requiring a number of loosely interacting systems.

Replica exchanges and energies are recorded in the .history files written in the output directories. These can be viewed with, e.g., "`xmgrace output/*/*.history`" and processed via awk or other tools. There is also a script to load the output into VMD and color each frame according to replica index. An example simulation folds a 66-atom model of a deca-alanine helix in about 10 ns.

`replica.namd` is the master script for replica temperature-exchange simulations. To run:

```
cd example
mkdir output
(cd output; mkdir 0 1 2 3 4 5 6 7)
mpirun namd3 +replicas 8 job0.conf +stdout output/%d/job0.%d.log
mpirun namd3 +replicas 8 job1.conf +stdout output/%d/job1.%d.log
```

The number of MPI ranks must be a multiple of the number of replicas (+replicas). Be sure to increment jobX for +stdout option on command line.

`show_replicas.vmd` is a script for loading replicas into VMD; first source the replica exchange conf file and then this script, then repeat for each restart conf file or for example just do "`vmd -e load_all.vmd`". This script will likely destroy anything else you are doing in VMD at the time, so it is best to start with a fresh VMD. `clone_reps.vmd` provides the `clone_reps` commmand to copy graphical representation from the top molecule to all other molecules.

`sortreplicas`, found in the namd3 binary directory, is a program to un-shuffle replica trajectories to place same-temperature frames in the same file. Usage:

```
sortreplicas <job_output_root> <num_replicas> <runs_per_frame> [final_step]
```

where job_output_root is the job specific output base path, including %s or %d for separate directories as in output/%s/fold_alanin.job1 This will be extended with .%d.dcd .%d.history for input files and .%d.sort.dcd .%d.sort.history for output files. The optional final_step parameter will truncate all output files after the specified step, which is useful in dealing with restarts from runs that did not complete. Colvars trajectory files are similarly processed if they are found.

A replica exchange config file should define the following Tcl variables:

- `num_replicas`, the number of replica simulations to use,
- `min_temp`, the lowest replica target temperature,
- `max_temp`, the highest replica target temperature,
- `steps_per_run`, the number of steps between exchange attempts,
- `num_runs`, the number of runs before stopping (should be divisible by `runs_per_frame` $\times$ `frames_per_restart`).
- `runs_per_frame`, the number of runs between trajectory outputs,
- `frames_per_restart`, the number of frames between restart outputs,
- `namd_config_file`, the NAMD config file containing all parameters, needed for the simulation except `seed`, `langevin`, `langevinTemp`, `outputEnergies`, `outputname`, `dcdFreq`, `temperature`, `bincoordinates`, `binvelocities`, or `extendedSystem`, which are provided by `replica.namd`,

- `output_root`, the directory/fileroot for output files, optionally including a "%s" that is replaced with the replica index to use multiple output directories,
- `psf_file`, the psf file for `show_replicas.vmd`,
- `initial_pdb_file`, the initial coordinate pdb file for `show_replicas.vmd`,
- `fit_pdb_file`, the coodinates that frames are fit to by `show_replicas.vmd` (e.g., a folded structure),

The `lib/replica/example/` directory contains all files needed to fold a 66-atom model of a deca-alanine helix:

- `alanin_base.namd`, basic config options for NAMD,
- `alanin.params`, parameters,
- `alanin.psf`, structure,
- `unfolded.pdb`, initial coordinates,
- `alanin.pdb`, folded structure for fitting in `show_replicas.vmd`,
- `fold_alanin.conf`, config file for `replica_exchange.tcl` script,
- `job0.conf`, config file to start alanin folding for 10 ns,
- `job1.conf`, config file to continue alanin folding another 10 ns, and
- `load_all.vmd`, load all output into VMD and color by replica index.

The `fold_alanin.conf` config file contains the following settings:

```
set num_replicas 8
set min_temp 300
set max_temp 600
set steps_per_run 1000
set num_runs 10000
# num_runs should be divisible by runs_per_frame * frames_per_restart
set runs_per_frame 10
set frames_per_restart 10
set namd_config_file "alanin_base.namd"
set output_root "output/%s/fold_alanin" ; # directories must exist

# the following used only by show_replicas.vmd
set psf_file "alanin.psf"
set initial_pdb_file "unfolded.pdb"
set fit_pdb_file "alanin.pdb"
```

## 11.7 Random acceleration molecular dynamics simulations

The “lib/ramd” directory stores the tcl scripts and the example files for the implementation of the Random Acceleration Molecular Dynamics (RAMD) simulation method in NAMD. The RAMD method can be used to carry out molecular dynamics (MD) simulations with an additional randomly oriented acceleration applied to the center of mass of one group of atoms (referred to below as “ligand”) in the system. It can, for example, be used to identify egress routes for a ligand from a buried protein binding site. Since its original implementation in the ARGOS [42, 88] program, the method has been implemented in AMBER 8 [66], and CHARMM [14]. The first implementation of RAMD in NAMD using a tcl script (available as supplementary material in [82]) provided only limited functionality compared to the AMBER 8 implementation and was followed with an implementation of RAMD and RAMD–MD in NAMD [19, 7]. Recently the RAMD method was improved in speed by using NAMD vector implementations and streamlining the code. The current implementation is now focused on the RAMD simulation and was used in the $\tau$RAMD procedure for the estimation of relative drug-target residence times [35].

Additional information is found in the README file in the “lib/ramd” directory. The user is encouraged to carefully read this information before starting production runs.

The two required scripts are stored in “lib/ramd/scripts”: (i) ramd–5.tcl defines the simulation parameters and passes them from the NAMD configuration file to the main script, (ii) “ramd–5_script.tcl” adds the randomly oriented force and performs all related computations.

Two examples for running RAMD are included in the directory “lib/ramd/example/”. The examples can be started using the RAMD-force.sh shell scripts.

The specific RAMD simulation parameters to be provided in the NAMD configuration file (listed below) should be preceded by the keyword “ramd”. The default values for these parameters are only given as guidance. They may not to be suitable for other systems.

Mandatory parameter settings:

- `ramd lastProtAtom` < Last index of protein atom >
  **Acceptable Values:** `positive integer`
  **Description:** Specifies the index of the last protein atom.

- `ramd firstRamdAtom` < First index of ligand atom >
  **Acceptable Values:** `positive integer`
  **Description:** Specifies the index of the first ligand atom.

- `ramd lastRamdAtom` < Last index of ligand atom >
  **Acceptable Values:** `positive integer`
  **Description:** Specifies the index of the last ligand atom.

- `ramd ramdfilename` < Name of ramd output file >
  **Acceptable Values:** `Valid file name`
  **Description:** Specified the name of the file where the ramd logs are written.

Optional parameter settings with a default. Depending on your simulation system, you might want to change these settings:

- `ramd firstProtAtom` < First index of protein atom >
  **Acceptable Values:** `positive integer`

**Default Value:** `1`
**Description:** Specifies the index of the first protein atom.

- `ramd ramdSteps` < Set number of steps in RAMD block >
  **Acceptable Values:** `positive integer`
  **Default Value:** `50`
  **Description:** Specifies the number of steps in 1 RAMD block; the simulations are evaluated every ‘ramdSteps’ steps.

- `ramd forceRAMD` < Set acceleration force >
  **Acceptable Values:** `positive decimal`
  **Default Value:** `16.0`
  **Description:** Specifies the force to be applied. Replaces the acceleration (accel) specified in previous releases. Defaults to 16 kcal/mol/Angstrom

- `ramd rMinRamd` < Set threshold for distance travelled RAMD >
  **Acceptable Values:** `positive decimal`
  **Default Value:** `0.01`
  **Description:** Specifies a threshold value for the distance in Angstroms travelled by the ligand in 1 RAMD block. In RAMD simulations the direction of the acceleration is changed if the ligand has travelled less than ‘rMinRamd’ Å in the evaluated block.

- `ramd forceOutFreq` < Set frequency of RAMD forces output >
  **Acceptable Values:** `positive integer`, Must be divisor of `ramdSteps`
  **Default Value:** `10`
  **Description:** Every ‘forceOutFreq’ steps, detailed output of forces will be written.

- `ramd maxDist` < Set center of mass separation >
  **Acceptable Values:** `positive decimal`
  **Default Value:** `50`
  **Description:** Specifies the distance in Angstroms between the the centers of mass of the ligand and the protein when the simulation is stopped.

- `ramd ramdSeed` < Set RAMD seed >
  **Acceptable Values:** `positive integer`
  **Default Value:** `14253`
  **Description:** Specifies seed for the random number generator for generation of RAMD force directions. Change this parameter if you wish to run different trajectories with identical parameters.

- `ramd debugLevel` < Set debug level of RAMD >
  **Acceptable Values:** `integer value`
  **Default Value:** 0
  **Description:** Activates verbose output if set to an integer greater than 0. Should be used only for testing purposes because the very dense output is full of information only relevant for debugging.

- `ramd namdVersion` < Set the NAMD version >
  **Acceptable Values:** `float value`
  **Default Value:** 2.13

**Description:** After NAMD version 2.10 a call to *enabletotalforces* is done to enable tcl processing in NAMD

Note: In the current RAMD implementation, combined RAMD-MD simulations, where RAMD blocks alternate with standard MD blocks are not available. In case you are are interested in this feature, please contact the RAMD developers at mcmsoft@h-its.org

Scripts for using RAMD in the $\tau$RAMD procedure for computing residence times are available at: `https://www.h-its.org/downloads/ramd/`.

# 12 Structure based simulations

## 12.1 Hybrid MD-Go Simulation

### 12.1.1 Hybrid MD-Go model

NAMD incorporates a hybrid MD-Go model (hereby referred to as Go) to study the conformation changes in biomolecular systems. The method replaces the physical-based nonbonded interactions with a smoother knowledge-based potential energy surface. Bonded interactions are taken from the classical force fields. By removing energetic traps along a MD trajectory, the system will be able to sample states not normally accessible to classical MD simulations.

### 12.1.2 Hybrid MD-Go considerations

Typically, Go simulations are conducted in the **absence of solvent** and with **electrostatic and van der Waals forces** in the system **turned off** to improve conformational space exploration. Due to the current implementation of Go, the partial charges and van der Waals radii need to be set to zero in the psf and parameter file to remove the physical nonbonded interactions. Additionally, NAMD uses a **reference PDB structure** to construct the Go pairwise potential between atoms.

Finally, the Go model in NAMD introduces the idea of chain types. Consider modeling a protein-nucleic acid complex. Using classical all-atom MD, a single force field describes all possible nonbonded interactions. With Go, however, one can create separate nonbonded force fields to describe the protein and nucleic acid interactions. In order to create separate force fields, atoms are grouped together using chain types where the chain types are taken from the occupancy field of the reference PDB file. For argument sake, assume that the protein atoms have an occupancy value of 1.0 and that the nucleic acid atoms have an occupancy value of 2.0. One now must define three separate Go potentials for intra-protein, intra-nucleic acid, and inter-protein-nucleic acid interactions. In terms of chain types, this corresponds to (1) between atom pairs fully in chain 1, (2) between atom pairs fully in chain 2, (3) between atom pairs where one atom is in chain 1 and the other atom is in chain 2 respectively. To run Go, a minimum of one chain type must be defined.

### 12.1.3 Configuration file modifications

The following configuration parameters are used to setup and run a Go simulation:

- `GoForcesOn` < Are Go forces turned on? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether or not Go forces should be calculated. If turned `'off'`, Go forces will not be calculated. If turned `'on'`, Go forces will be calculated. By default, the Go forces will be calculated in addition to the electrostatic and van der Waals forces. To simulate a system using only Go forces, the partial charges and Lennard-Jones parameters can be set to zero in the force field files.

- `GoParameters` < Parameter file defining Go potential >
  **Acceptable Values:** file

**Description:** File contains parameters to define the Go pairwise forces between different chain types. All possible chain type pairing combinations must be enumerated. Chain types are defined in the GoCoordinates file. The format for the GoParameters file is described in the next section.

- `GoCoordinates` < Reference structure for Go simulation >
  **Acceptable Values:** PDB file
  **Description:** PDB file contains the reference structure used to define the Go potential. The file need not be the same file used to initialize the coordinates of the MD simulation; however, it must contain the same number of atoms in the same order as given in the structure (.psf) and coordinates (.coor) file. Additionally, the occupancy fields of the PDB file will be read to determine which chain type an individual atom belongs to, and, thus, which pairwise Go potential to use to calculate forces. By default, the occupancy value of 0.0 turns off the Go potential for that particular atom.

- `GoMethod` < controls method for storing Go contact information >
  **Acceptable Values:** `lowmem` or `matrix`
  **Description:** Specifies whether the Go contacts should be calculated on the fly or stored in a matrix respectively. In most cases, `'lowmem'` will be sufficient. However, for smaller systems, the `'matrix'` does offer a slight performance speedup in terms of wall time. Variable is only used if GoForcesOn is `'on'`

The following sections describe the format of the GoParameter file.

### 12.1.4 GoParameter format

When running a Go simulation, the atoms are partitioned into chains according to the occupancy value given in the GoCoordinates file. For every possible pairwise combination between chains, a Go potential is defined by the following equations:

Let $r_{i,j}^{ref}$ be the pairwise distance between atoms i and j in the reference structure. If $r_{i,j}^{ref}$ is less than the Go cutoff distance, the pairwise potential between atoms i and j is given by:

$$V_{Go}(r_{i,j}, \epsilon, \sigma_{i,j}^{ref}, a, b) = 4\epsilon\left[\left(\frac{\sigma_{i,j}^{ref}}{r_{i,j}}\right)^a - \left(\frac{\sigma_{i,j}^{ref}}{r_{i,j}}\right)^b\right]$$

where $\sigma_{i,j}^{ref}$ is given as $\left(\frac{b}{a}\right)^{\frac{1}{b-a}} r_{i,j}^{ref}$. If $r_{i,j}^{ref}$ is greater than the Go cutoff distance, the pairwise potential between atoms i and j is given by:

$$V_{Go}(r_{i,j}, \epsilon^{rep}, \sigma^{rep}, expRep) = 4\epsilon^{rep}(\frac{\sigma_{i,j}^{rep}}{r_{i,j}})^{expRep}$$

For each pairwise chain combination, the following parameters are needed to define the Go potential:

- **chaintypes (2 floats):** (*first_chain second_chain*) Defines the pairwise chain interaction

- **epsilon (1 float):** ($\epsilon$) Determines the $\epsilon$ constant of the Go potential in units of $kcal \cdot mol^{-1} \cdot \AA^{-2}$
- **exp_a (1 integer):** ($a$) Determines the 'a' constant for the Go potential
- **exp_b (1 integer):** ($b$) Determines the 'b' constant for the Go potential
- **expRep (1 integer):** (*expRep*) Determines the 'expRep' constant for the Go potential
- **sigmaRep (1 float):** ($\sigma^{rep}$) Determines the $\sigma^{rep}$ constant for the Go potential in units of $\AA$
- **epsilonRep (1 float):** ($\epsilon^{rep}$) Determines the $\epsilon^{rep}$ constant for the Go potential in units of $kcal \cdot mol^{-1} \cdot \AA^{-2}$
- **cutoff (1 float):** (*cutoff*) Defines the Go cutoff distance for this particular pairwise chain in units of $\AA$
- **[Optional] restriction (1 integer):** Determines if interactions between the $i^{th}$ and $i^{th} + integer$ adjacent residue should be excluded. Multiple restriction between adjacent residues can be defined within a chaintype. Each additional new restriction is given on its own line.

Each pairwise chaintype should be written in its own block of text with each entry given its own line. It is recommended that individual pairwise potential be separated by a blank line.

## 12.2 Running SMOG simulations

### 12.2.1 SMOG model considerations

NAMD supports the SMOG model from published from Onuchic's lab [86, 87]. The input files for SMOG can be generated from the SMOG website (`http://smog-server.org`) [55]. It is recommended to run these simulations with 1-4 exclusions (as opposed to scaled 1-4), a 0.5fs timestep, and with a 0.5fs timestep (as described in [86, 87]).

### 12.2.2 Configuration file modifications

As the SMOG model uses GROMACS topology and coordinate files, the GROMACS configuration parameters—`gromacs`, `grotopfile`, grocoorfile—must be defined. The description for the GROMACS configuration parameters are reproduced below:

- `gromacs` < use GROMACS format force field? >
  **Acceptable Values:** on or off
  **Default Value:** off
  **Description:** If `gromacs` is set to on, then `grotopfile` must be defined, and `structure` and `parameters` should not be defined.
- `grotopfile` < GROMACS format topology/parameter file >
  **Acceptable Values:** UNIX filename
  **Description:** This file contains complete topology and parameter information of the system.

- `grocoorfile` < GROMACS format coordinate file >
  **Acceptable Values:** UNIX filename
  **Description:** This file contains the coordinates of all the atoms. Note that `coordinates` can also be used for PDB format coordinate file. When `gromacs` is set to `on`, either `grocoorfile` or `coordinates` must be defined, but not both.

To run a SMOG simulation, the following extra parameters must be defined.

- `GromacsPair` < Are GROMACS pair forces turned on? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** This variable determines if the pair section of the GROMACS topology file grotopfile is evaluated. Currently, only Lennard-Jones type pairs are supported. Variable is only used if gromacs variable is `'on'`.

- `staticAtomAssignment` < Optimization to fix atoms onto a specific node >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies if atoms should be statically fixed to a node. This will change the internode communication and will give a significant speed-up to MD simulations if the atoms are moving rapidly. It is suggested that SMOG simulations use the staticAtomAssignment flag.

## 13 Constant-pH Simulations [1]

Constant-pH MD in NAMD is based on the scheme first proposed by Stern [74] and later revised and extended by Chen and Roux [15]. A detailed description of the modifications and improvements made in the NAMD implementation has been presented elsewhere by Radak, *et al.* [60] and this is likely the best comprehensive description of the method, its uses, and its limitations/pitfalls. Herein the goal is to provide a working understanding of how the implementation works and what kinds of data it produces.

### 13.1 Overview and Theoretical Background

Constant-pH MD is a simulation methodology specially formulated for the treatment of variable protonation states. This is to be contrasted with conventional force-field based MD simulations, which generally treat protonation states by assuming they are fixed. Consider, for example, a protein with two titratable residues which may both be either protonated or deprotonated (Figure 12); the system has four possible protonation states. In the conventional route, the user must enumerate these possibilities, construct distinct topologies, and then simulate the cases individually. The simulations for each state must then be connected by either asserting knowledge about the system (*e.g.*, by assuming that only certain states are of biological importance) or by performing additional simulations to probe transitions between states directly (*e.g.*, by performing free energy calculations). In a constant-pH MD simulation, knowledge of the transformations is not assumed and is instead actively explored by interconverting between the various protonation states. This is especially useful when the number of protonation states is extremely large and/or prior information on the importance of particular states is not available.

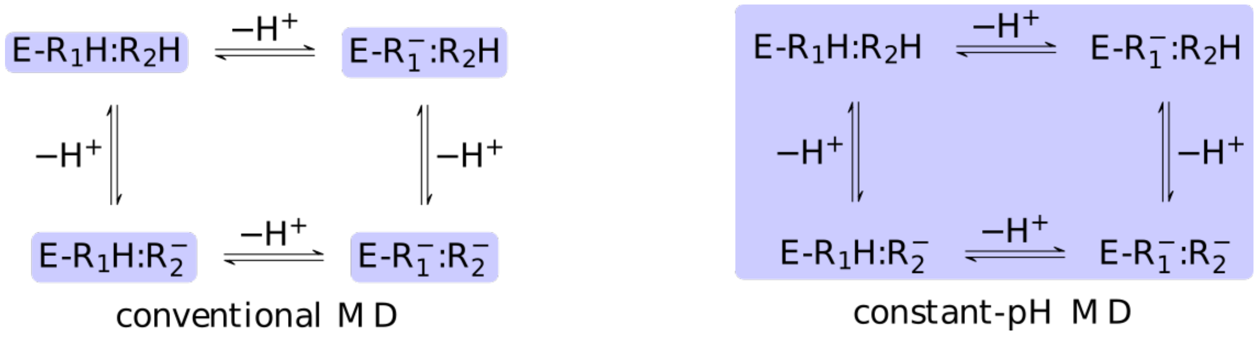


Figure 12: The core difference between conventional and constant-pH MD can be illustrated by a simple enzyme $E$ with four protonation states describing the occupancy of two titratable residues, $R_1$ and $R_2$. A conventional MD simulation handles the states *separately* (left panel). The relative importance of the states must be known beforehand or computed by other means. Conversely, a constant-pH MD simulation handles the states *collectively* and actively simulates interconversion (right panel). Determining the relative importance of the states is a direct result of the simulation.

[1]The features described in this section were implemented by Brian K. Radak (Argonne National Laboratory, Argonne, IL USA) with considerable technical support from James C. Phillips (University of Illinois, Urbana, IL USA) and Wei Jiang (Argonne National Laboratory). The algorithm draws heavily from earlier work by Yunjie Chen and Benoît Roux and later by Donghyuk Suh (University of Chicago, Chicago, IL USA), as well as time spent as a postdoctoral scholar at University of Chicago. Testing and validation were also aided by Christophe Chipot (Université de Lorraine, Vandœuvre-lès-Nancy cedex France and University of Illinois).

In formal terms, conventional MD samples from a canonical ensemble, whereas constant-pH MD samples from a semi-grand canonical ensemble. The new partition function,

$$\Xi(\mathrm{pH}) = \sum_{\boldsymbol{\lambda} \in \mathcal{S}} Q_{\boldsymbol{\lambda}} 10^{-n_{\boldsymbol{\lambda}} \mathrm{pH}}, \tag{50}$$

is essentially a weighted summation of canonical partition functions, $Q_{\boldsymbol{\lambda}}$, each of which are defined by an occupancy vector, $\boldsymbol{\lambda}$. The elements of $\boldsymbol{\lambda}$ are either one or zero depending on whether a given protonation site is or is not occupied, respectively. For a vector of length $m$, the set of all protonation states, $\mathcal{S}$, has at most $2^m$ members. In order to sample from the corresponding semi-grand canonical distribution function, a simulation must explore *both* the phase space defined by the canonical paritition functions and the state space defined by the different occupancy vectors. The fraction of simulation time spent in each state is dictated by the weights in the summation and these depend on the pH and the number of protons, $n_{\boldsymbol{\lambda}}$, in the system (*i.e.*, the sum of the elements in $\boldsymbol{\lambda}$).

Although a constant-pH MD system may contain any number of titratable protons, the base transformation is always the movement of *one* proton from a molecule into a bath of non-interacting protons "in solution." For a generic chemical species A, this corresponds to the usual deprotonation reaction definition, except with fixed pH:

$$\mathrm{HA} \underset{\text{pH fixed}}{\overset{-\mathrm{H}^+}{\rightleftharpoons}} \mathrm{A}^-.$$

In the language of statistical mechanics the species HA and A$^-$ refer to all terms in Eq. (50) which do and do not, respectively, contain the specific proton in question (*i.e.*, the particular element of $\boldsymbol{\lambda}$ is one or zero). By taking out a factor of $10^{-\mathrm{pH}}$, this can be re-written as

$$\Xi(\mathrm{pH}) = \Xi_{\mathrm{A}^-}(\mathrm{pH}) + \Xi_{\mathrm{HA}}(\mathrm{pH}) 10^{-\mathrm{pH}}$$

and then recast as a statistical mechanical analog of the Henderson-Hasselbalch equation by recognizing that $\Xi_{\mathrm{A}^-}(\mathrm{pH})/\Xi_{\mathrm{HA}}(\mathrm{pH})$ is just the ratio of deprotonated / protonated fractions of species A. The *protonated* fraction is then

$$P_{\mathrm{HA}}(\mathrm{pH}) = \frac{1}{1 + 10^{\mathrm{pH} - \mathrm{p}K_{\mathrm{a}}(\mathrm{pH})}}; \qquad \mathrm{p}K_{\mathrm{a}}(\mathrm{pH}) \equiv -\log \frac{\Xi_{\mathrm{A}^-}(\mathrm{pH})}{\Xi_{\mathrm{HA}}(\mathrm{pH})}. \tag{51}$$

In practice, $P_{\mathrm{HA}}(\mathrm{pH})$ can be calculated from a simulation by simply counting the fraction of time spent in state HA (*e.g.*, the fraction of time a specific element of $\boldsymbol{\lambda}$ is one). Note also that $\mathrm{p}K_{\mathrm{a}}(\mathrm{pH})$ is formally a pH dependent function unless the system only contains one proton (or type of proton).

In most experimental contexts, a different form of Eq. (51) is used which is often referred to as a "generalized" Hill equation. This corresponds to a specific choice of pH dependence such that

$$\mathrm{p}K_{\mathrm{a}}(\mathrm{pH}) \approx \mathrm{p}K_{\mathrm{a}}^{(\mathrm{a})} + (1 - n)\left(\mathrm{pH} - \mathrm{p}K_{\mathrm{a}}^{(\mathrm{a})}\right).$$

The constant $n$ is then known as the Hill coefficient and the so-called apparent p$K_{\mathrm{a}}$, $\mathrm{p}K_{\mathrm{a}}^{(\mathrm{a})}$, generally corresponds to the inflection point of a plot of $P_{\mathrm{HA}}(\mathrm{pH})$. Both quantities are usually determined by non-linear regression after $P_{\mathrm{HA}}$ has been determined at different pH values.

## 13.2 Implementation Details

In NAMD, each canonical partition function is represented by a specific force field description embodied in a PSF – in order to change the protonation state the underlying PSF must also be modified. This is accomplished by a close coupling to `psfgen`. The models that can be used with constant-pH MD are thus limited to only those which can be completely implemented within `psfgen`. This also means that NAMD requires access to residue topology files (RTFs) during the course of a simulation. These must be specified with the `psfgen topology` command. The `top_cph36_prot.rtf` and `top_cph36_cgenff.rtf` files already provided in directory `/lib/namdcph/toppar` include additional information necessary to generate custom hybrid topologies that are specific for constant pH MD. Before including the custom topology files, the corresponding standard residue topology files, such as `top_all36_prot.rtf` or `top_all36_cgenff.rtf`, must first be included.

For consistency between topological descriptions, NAMD uses "dummy" atoms to represent non-interacting protons. These atoms have the same mass as protons but only interact with the system via a minimal number of force field bonded terms. This formalism guarantees that: 1) the number of atoms/coordinates during the simulation remains fixed and 2) the thermodynamics of the model is unchanged. The latter point is subtle and warrants comment. As implemented in NAMD, constant-pH MD only captures the thermodynamics of the semi-grand canonical ensemble. There is no active description of proton dissociation events. However, this is more of a limitation of classical MD than a particular shortcoming of NAMD. A useful analogy may be the use of Langevin dynamics as a thermostat as opposed to a phenomonological model for Brownian motion.

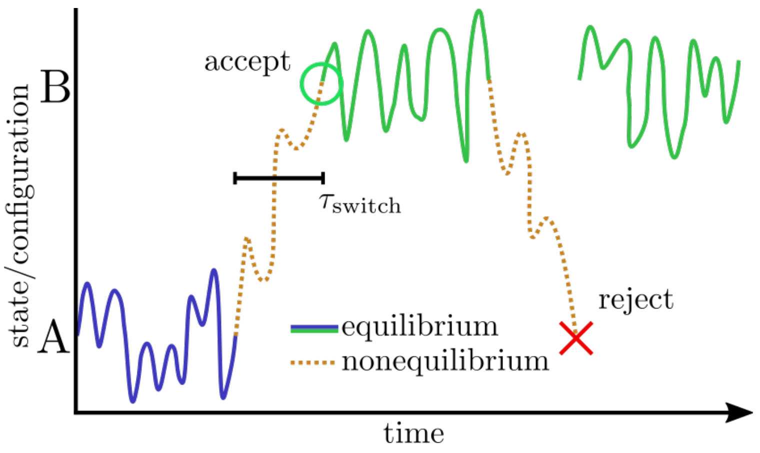


Figure 13: The basic constant-pH MD scheme in NAMD is to alternate equilibrium sampling in a fixed protonation state followed by a nonequilibrium MD Monte Carlo move to sample other protonation states. The latter move can be accepted or rejected. If accepted, the simulation continues in the new protonation state. If the move is rejected, sampling continues as if the move were never attempted at all.

The basic scheme in NAMD is to alternately sample the protonation state and then the configuration space within that state. Protonation state sampling is accomplished by an alchemical coupling scheme that forcibly turns off interactions with the current protonation state and turns on interactions with a candidate protonation state. This nonequilibrium "switching" is accomplished with the alchemy code (specifically the thermodynamic integration code branch) and necessarily has lower performance (by about 30%) than regular MD due to the added electrostatic calculations in the reciprocal space (*i.e.*, when using PME). However, the configuration space sampling should still have normal performance. The switching process exerts work on the system and thus drives the system out of equilibrium. However, an appropriately designed Monte Carlo (MC) move using an accept/reject criterion can recover the correct semi-grand canonical equilibrium distribution in both the state and configuration spaces [54, 16]. The resulting scheme is a hybrid nonequilibrium MD/MC (neMD/MC) algorithm. The most important conceptual change from conventional MD is that, rather than being a continuous trajectory, the simulation now becomes a series of cycles composed of an MD and neMD/MC step. This means that the length of the simulation is no longer simply determined by the number of steps (`numsteps`) but rather the number of cycles. The length of a cycle is also determined by two parts – the amount of time on equilibrium sampling and the amount of time executing the switch.

It may be profitable/necessary to vary the switch time depending on the type of protonation change that is being effected. Indeed, this is a critical factor in the efficiency of the method. That is, if the switch is too short, then moves are unlikely to be accepted and effort will be wasted when the move is rejected. However, if the switch is too long, then an inordinate amount of effort will be spent sampling the state space and there will be fewer resources left for exploring the configuration space. Some basic qualities of the system that affect sampling have been determined using nonequilibrium linear response theory [61]. In short, there are intrinsic limits based on: 1) the extent that differing interactions between each state fluctuate (according to some variance, $\sigma_0^2$) and 2) the "molecular" time scale, $\tau_{\mathrm{m}}$, on which these fluctuations change. These effects are roughly captured by the expression [61, 60]:

$$\tau_{\mathrm{opt}} \leq \frac{\sigma_0^2 \tau_{\mathrm{m}}}{2.83475},$$

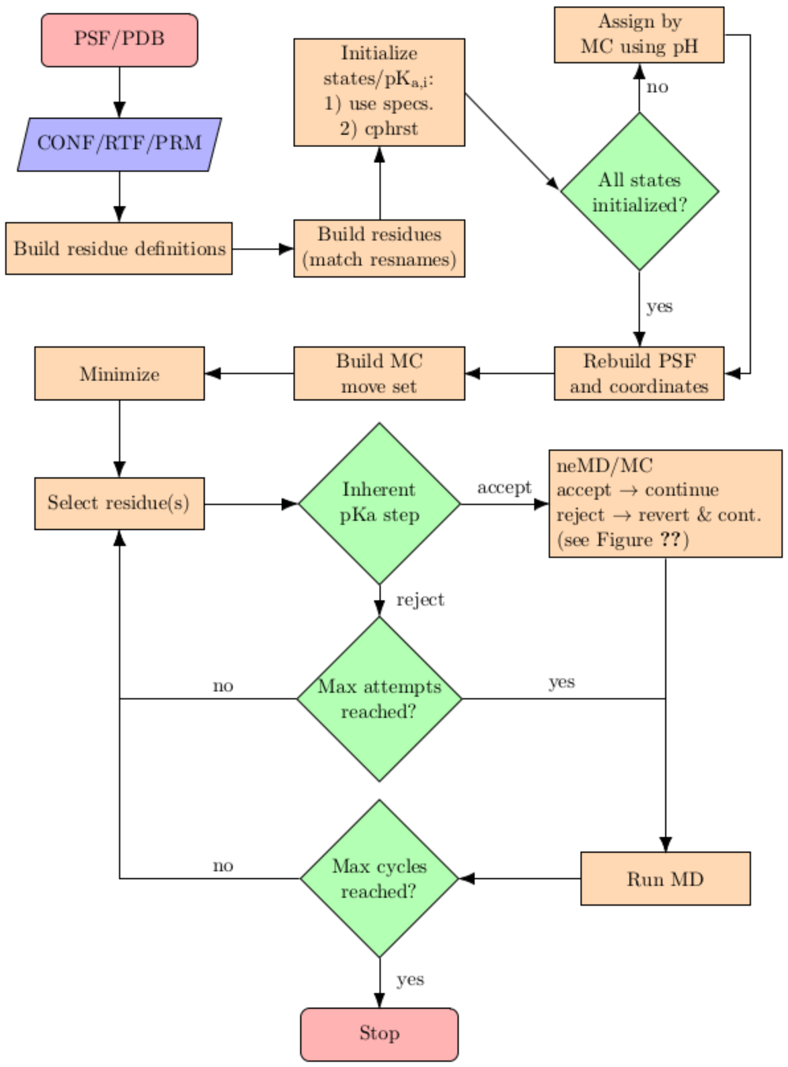
where $\tau_{\mathrm{opt}}$ is some optimal switching time, in the sense of maximizing the rate at which protonation states interconvert. Overall, switching times on the order of tens of picoseconds tend to be optimal in that they balance the high cost of switching versus the high acceptance rate at longer switching times (in the infinite time limit the perturbation is adiabatic and exerts zero work). For titratable groups exposed primarily to aqueous solvent, a switch on the order of 10-20 ps appears to give near optimal results [61, 60]. An equivalent formulation of the above expression is that mean acceptance rates around 20-25% are likely near optimal.

**Important Limitations:**
For various reasons concerning the implementation, constant-pH simulations are currently *incompatible* with the following NAMD functionalities in all or most situations:

- Any system using GPUs/CUDA
- Generalized Born implict solvent (`GBIS`)
- Alchemical free energy calculations, *e.g.*, ligand binding (`alch`)
- Drude polarizable force fields
- Hybrid quantum mechanical/molecular mechanical simulations
- Collective variables (`colvars`)
- `extraBonds`

This list is neither exhaustive nor definitive. In many instances the problem may be overcome by modest additional developments.

`namdcph` Execution Flow Chart


PSF/PDB
CONF/RTF/PRM
Build residue definitions
Build residues (match resnames)
Initialize states/pK$_{a,i}$: 1) use specs. 2) cphrst
All states initialized?
no
Assign by MC using pH
yes
Rebuild PSF and coordinates
Build MC move set
Minimize
Select residue(s)
Inherent pKa step
accept
neMD/MC accept → continue reject → revert & cont. (see Figure ??)
reject
Max attempts reached?
no
yes
Run MD
Max cycles reached?
no
yes
Stop

## 13.3 New Commands and Keywords

The constant-pH implementation is largely implemented in Tcl and can be found in `/lib/namdcph/namdcph.tcl`, where the base directory is the NAMD source home directory. When that file has been loaded with a suitable `source` command, the following commands and keywords are available and appear to the user in a way similar to NAMD syntax. The most significant change from normal NAMD usage is that there is generally no need to use the `run` command. One should instead use the new `cphRun` command; this can only be used *once* per script for now. *NB*, all commands and keywords are currently case sensitive!

`cphRun` < Run constant-pH MD >
**Arguments:** <*numsteps*> [*numcycles*]
**Defaults:** `numcycles` = 1
**Description:** Execute `numcycles` cycles of constant-pH MD with the current settings. Each cycle consists of 1) a neMD/MC move in both configuration and protonation space and 2) MD based sampling in configuration space. By default, configuration space sampling simply consists of `numsteps` dynamics, as in conventional MD. The nature of the neMD/MC moves, however, is more elaborate and controlled by other keywords, *many of which are required* (see below).

### 13.3.1 Required Keywords

- `pH` < pH value that the system is in contact with >
  **Acceptable Values:** decimal (usually between 0 and 14)
  **Description:** The `pH` is effectively a chemical potential applied to protons *only.* This value affects the details of neMD/MC moves but otherwise has no effect on the system dynamics.

- `cphConfigFile` < File defining titratable residues >
  **Acceptable Values:** filename
  **Description:** The `cphConfigFile` contains definitions for the available titratable residues. This is essentially meta information regarding the RTF contents, but also includes experimental references and additional force field parameterization.

- `cphNumstepsPerSwitch` < Number of steps during nonequilibrium switching >
  **Acceptable Values:** [*integer* [<*move label*> <*integer*>] ...]
  **Description:** Each move must have an associated number of steps per switch. If an odd number number of arguments is specified, then the first such argument is assumed to be a default number for all such moves. After this (or if an even number of arguments is specified) all remaining arguments are assumed to be specific assignments for a given move label of the form <*segid*>:<*resid*>:<*resname*>/<*segid*>:<*resid*>:<*resname*>/....

### 13.3.2 Commonly Used Options

- `cphSetResidueState` < Set the initial state of one or more titratable residues. >
  **Acceptable Values:** <*segid*>:<*resid*>:<*resname*> <*state*> [...]
  **Description:** Initial residue states can be assigned in three ways (in descending order of precedence): 1) via this command, 2) from a `cphRestartFile`, and 3) randomly from the assigned `pH` and the current inherent pKa of each residue.

- `cphSetResiduepKai` < Set the inherent pKa of one or more titratable residues. >
  **Acceptable Values:** <*segid*>:<*resid*>:<*resname*> <*pKai*> [...]
  **Description:** The two step inherent pKa algorithm implemented here permits on-the-fly update of an estimate for the pKa(s) of each residue. These can either be guessed at the outset (the default is to use the reference pKa) or updated as the simulation progresses. A more accurate estimate of the inherent pKa increases the statistical efficiency of the method, but the long time result is formally unbiased regardless of the value. If an extremely large or extremely small value is assigned, then the residue will be assigned the most probable protonation state at the given pH and likely remain fixed in that state.

- `cphExcludeResidue` < Exclude one or more residues from being titratable >
  **Acceptable Values:** <*segid*>:<*resid*>:<*resname*> [...]
  **Description:** By default, any residue that matches a titratable residue type will be allowed to change protonation state. This command permits specific residues to be excluded from consideration in a manner that is similar to assigning an extreme inherent pKa (see `cphSetResiduepKai`). The main differences are that 1) the protonation state will not be modified and remain as it is in the original PSF and 2) the protons in the residue will *not* be tracked in the `cphlog` file. This command is not always recommended, but is currently necessary for handling disulfide linkages.

- `cphRestartFile` < Restart file for constant-pH >
  **Acceptable Values:** filename
  **Description:** Constant pH requires additional checkpoint information regarding the state of the titratable residues and the nature of the neMD/MC moves. This (optional) information is read from the file specified here. After/during a simulation, this information is written to `[outputname]`.cphrst.

- `cphRestartFreq` < Frequency at which constant-pH checkpoint files are written >
  **Acceptable Values:** Non-negative integer
  **Default Value:** 0
  **Description:** Checkpoint information is written to `[outputname]`.cphrst every `cphRestartFreq` cycles (*not* MD steps). A checkpoint file is *always* written at the end of the last cycle.

- `cphOutFile` < Log file for constant-pH >
  **Acceptable Values:** filename
  **Default Value:** `[outputname]`.cphlog
  **Description:** Titratable residue state information is logged here after every cycle.

- `cphProposalWeight` < MC move label and weight specifications >
  **Acceptable Values:** <*move label*> <*weight*> [[<*move label*> <*weight*>] ...]
  **Description:** During each cycle, MC moves are selected from the move set and then accepted/rejected according to a Metropolis criterion based on the combined inherent pKa information and pH. The move weight affects the probability that such a move is selected. Note that *this does not affect the probability that any given proposal is accepted*, it merely increases the number of attempts at the given proposal. This may be useful in a system where one desires specific attention on a given process, such as proton transfer or the exchange of a given residue, but one does not want to assume that all other residue protonation states are

nominally fixed. By default all moves are assigned equal weights of 1.0. During the simulation these are automatically normalized to a discrete probability mass function.

- `cphMaxProposalAttempts` < Maximum number of switch proposal attempts per cycle >
  **Acceptable Values:** integer
  **Default Value:** 0
  **Description:** During each cycle, MC moves are selected from the move set and then accepted/rejected according to a Metropolis criterion based on the combined inherent pKa information and pH. This process stops when either a switch move is accepted or a maximum limit is reached. Any value less than one defaults to the number of titratable residues in the system.

- `cphNumMinSteps` < Number of steps of minimization before dynamics >
  **Acceptable Values:** integer
  **Default Value:** 0
  **Description:** This is a replacement for the normal minimize command, which is not compatible with constant-pH due to PSF modifications during initialization. Setting this option to a modest number (100–200, say) might be necessary when randomizing protonation states based on pH, since in that case it cannot be assumed that the starting structure is representative of the initial protonation state.

### 13.3.3 Specialized Options

- `cphForceConstant` < force constant for alchemical switches (in kcal/mol-Å$^2$) >
  **Acceptable Values:** Non-negative decimal
  **Default Value:** 100.0
  **Description:** During "dual-topology" alchemical switches, a harmonic bond is formed between analogous atoms in each alchemical region. This rigorously leaves all static thermodynamic quantities intact and is generally expected to improve the stability of dynamic quantities.

- `cphMDBasename` < basename of intermediate files for equilibrium MD >
  **Acceptable Values:** string
  **Default Value:** namdcph.md
  **Description:** PSF/coordinate modifications are currently done via the file system and utilize intermediate files. It may be advantageous to direct this I/O to a fast temporary directory.

- `cphSWBasename` < basename of intermediate files for nonequilibrium (switch) MD >
  **Acceptable Values:** string
  **Default Value:** namdcph.sw
  **Description:** PSF/coordinate modifications are currently done via the file system and utilize intermediate files. It may be advantageous to direct this I/O to a fast temporary directory.

- `cphUseGPU` < use GPU-resident mode for dynamics calculations? >
  **Acceptable Values:** on or off
  **Default Value:** off
  **Description:** Specifies whether or not to use GPU-resident mode for executing the

nonequilibrium switch and the equilibrium MD. If on, these tasks are executed using GPU-resident dynamics, which improves performance significantly. If off (default), all calculations are performed on the CPU.

When `cphUseGPU` is on, the constant-pH scripts make use of the following environment variables as shown below:

- `export CONFIG_FILE start.namd`
- `export CPU_PATH=/path/to/cpu-only-build/namd3`
- `export GPU_PATH=/path/to/gpu-resident-build/namd3`
- `export CATDCD_PATH=/path/to/catdcd`
- `export NUM_THREADS=$SLURM_CPUS_PER_TASK`

where `start.namd` refers to the main configuration file and `NUM_THREADS` is to be the `+p` argument for launching NAMD. The `catdcd` program comes with VMD. The alternative to defining these environment variables is to hardcode path names into the `namdcph.core.tcl` file.

Two different builds of NAMD are needed for GPU-accelerated constant-pH to work, a CPU-only build and a GPU-resident build. The workflow starts the calculation using the CPU-only build, e.g.,

- `$CPU_PATH +p$NUM_THREADS +setcpuaffinity $CONFIG_FILE`

With `cphUseGPU` on, the scripts launch GPU-resident NAMD to run as a subprocess. To maintain output compatibility with the original constant-pH implementation, `catdcd` is used to glue DCD frames generated by the NAMD subprocess. Performance can be very fast, but is sensitive to the number of cores used for launching NAMD and degrades quickly when attempting to schedule with too many cores. The recommendation is to benchmark starting with a single core (`NUM_THREADS`=1) and to compare equilibrium MD performance against a similarly sized system to determine an optimal core count. Presently, this implementation supports only single-GPU runs. Interested users are encouraged to look at the example in directory `lib/namdcph/examples/bbl/`.

- `cphNumEquilMDSteps` < number of steps of low-temperature equilibration dynamics >
  **Acceptable Values:** integer
  **Default Value:** 0
  **Description:** When `cphUseGPU` is on, the `cphNumEquilMDSteps` parameter may be used instead of the minimize command (`cphNumMinSteps`). Specifying `cphNumEquilMDSteps` will implement low-temperature equilibration with a 0.1 fs timestep. Although GPU-resident dynamics is very fast, generally more steps are required than when using the minimization protocol.

- `GalvaniOffset` < offset potential (in mV) >
  **Acceptable Values:** integer
  **Default Value:** 0
  **Description:** For complex systems such as membrane proteins, where the content of the simulation box includes large nonaqueous regions, the magnitude of the Galvani potential of the bulk water phase can depart considerably from 0 mV. By introducing an offset potential

correction ($\phi_{\text{offset}}$), the Galvani potential of the bulk phase may be restored to 0 mV. With the correction, the work calculated from a nonequilibrium switch becomes $W_{\lambda\rightarrow\lambda'} + \Delta Q_{\text{tot}}\phi_{\text{offset}}$, where $\Delta Q_{\text{tot}}$ is the total charge increment resulting from the change in protonation state. An appropriate value of this offset parameter must be carefully determined that is specific to the system of interest. The user is referred to Bignucolo, et al., for further details [8].

**Undocumented Features:**
The constant-pH code is actively under development, although future work will almost exclusively be in adding new features and capabilities as well as improving performance. Because the code is fairly lightweight and available in `Tcl`, the intrepid user may discover "easter egg" features which are not listed in the documentation. **USE UNDOCUMENTED FEATURES AT YOUR OWN RISK.** Such undocumented features may work (and even be advisable) for specific problems, but have not undergone as rigorous of testing and may be prone to unintended consequences.

## 13.4 Minimal Examples

Constant-pH simulations can be carried out with largely the same options as conventional MD simulations (with some exceptions, see previous sections). The follwing examples assume that: 1) PSF and PDB files for the system of interest have already been constructed and 2) appropriate simulation keywords have already been chosen (*e.g.*, for PME, Langevin dynamics, *etc.*).

```
# End conventional settings...
source .../namd/lib/namdcph/namdcph.tcl
# Constant-pH MD requires additional force field files _during_ the simulation.
# In general, all RTFs used to construct the system need to be included with
# the ‘‘topology’’ command (just as in psfgen). Additional constant-pH specific
# RTF and PRM files are also necessary, as well as an accompanying
# configuration file in JSON format.
#
cphConfigFile <path to JSON config file>
topology <path to RTF>
topology <path to another RTF>
pH 7.0
# The following defaults all nonequilibrium switches to 5000 steps and then
# increases the time for residue 5 of segid PROA to 7500 steps -- multiple
# residues can be specified
#
cphNumStepsPerSwitch 5000 PROA:5:ASP 7500
# Run 100 minimization cycles before starting dynamics.
cphNumMinSteps 100
# Run 2500 steps of MD between attempted protonation state changes. Run 10
# cycles of MD and neMD/MC. The _upper_ bound of the simulation is thus:
#
# 10*(2500 + 7500) = 100000 steps
#
```

```
# but the actual simulation may be shorter in length.
#
cphRun 2500 10
```

**Restarting a simulation**
The following assumes that a simulation has already been run (as in the example above). For clarity we shall assume that `outputname` was set to ”foo” such that restart files have been written to foo.coor and foo.vel (normal output) as well as foo.psf, foo.pdb, and foo.cphrst (constant-pH specific output).

```
# End conventional settings...
source .../namd/lib/namdcph/namdcph.tcl
# Constant-pH MD requires additional force field files _during_ the simulation.
# In general, all RTFs used to construct the system need to be included with
# the ‘‘topology’’ command (just as in psfgen). Additional constant-pH specific
# RTF and PRM files are also necessary, as well as an accompanying
# configuration file in JSON format.
#
cphConfigFile <path to JSON config file>
topology <path to RTF>
topology <path to another RTF>
pH 7.0

structure foo.psf
coordinates foo.pdb
binCoordinates foo.coor
binVelocities foo.vel
cphRestartFile foo.cphrst
# NB: switch times and inherent pKa values are read here and no longer need to
# be specified as during initialization

cphRun 2500 10
```

# 14 Hybrid QM/MM Simulations

Even though molecular mechanics (MM) force-fields are based on quantum mechanical calculations and experimental observations, only quantum mechanics (QM) can give a complete and accurate understanding of many biochemical processes, particularly those involving chemical reactions or charge redistribution. Nevertheless, even with the advanced hardware technology available today, the computational cost of studying nanosecond-long dynamics of entire systems relying solely on QM methodologies is usually prohibitive. A common route to circumvent this cost barrier is to confine the QM formalism to a sub-region of a system and to include the effects of the surrounding system through MM simulations, leading to hybrid QM/MM simulations [67].

NAMD's comprehensive QM/MM suite [49] was developed to provide easy setup, visualization and analysis of QM/MM simulations through the graphical user interface VMD/QwikMD [62], and a broad range of QM methods through NAMD's new "QMForces" module. The QM/MM interface in NAMD supports the simulation of many independent QM regions, and smooth integration with a vast collection of enhanced sampling methods. In hybrid QM/MM simulations, NAMD offloads part of its standard force and energy calculations to a QM program, either through native interfaces to MOPAC [75, 44] or ORCA [53], or through a flexible generic interface requiring a wrapper script, where exemplary Python wrappers are provided for Gaussian, TeraChem and Q-CHEM. Multiple QM-MM coupling schemes are implemented, allowing for both mechanically and electrostatically embedded QM regions to be used (see description in Nature Methods [49]). QM/MM simulations require the same input files used for classical MD, with additional options in the configuration file. QM and MM atoms covalently bound are usually treated by redistributing the MM atom's charge over its nearest MM neighbors and by capping the QM atom with a hydrogen atom, as shown in Figure 14 for a solvated tri-alanine QM/MM calculation using the NAMD/ORCA interface. Tests of the QM/MM interface for accuracy, stability and performance, are provided as supporting information in Nature Methods [49].

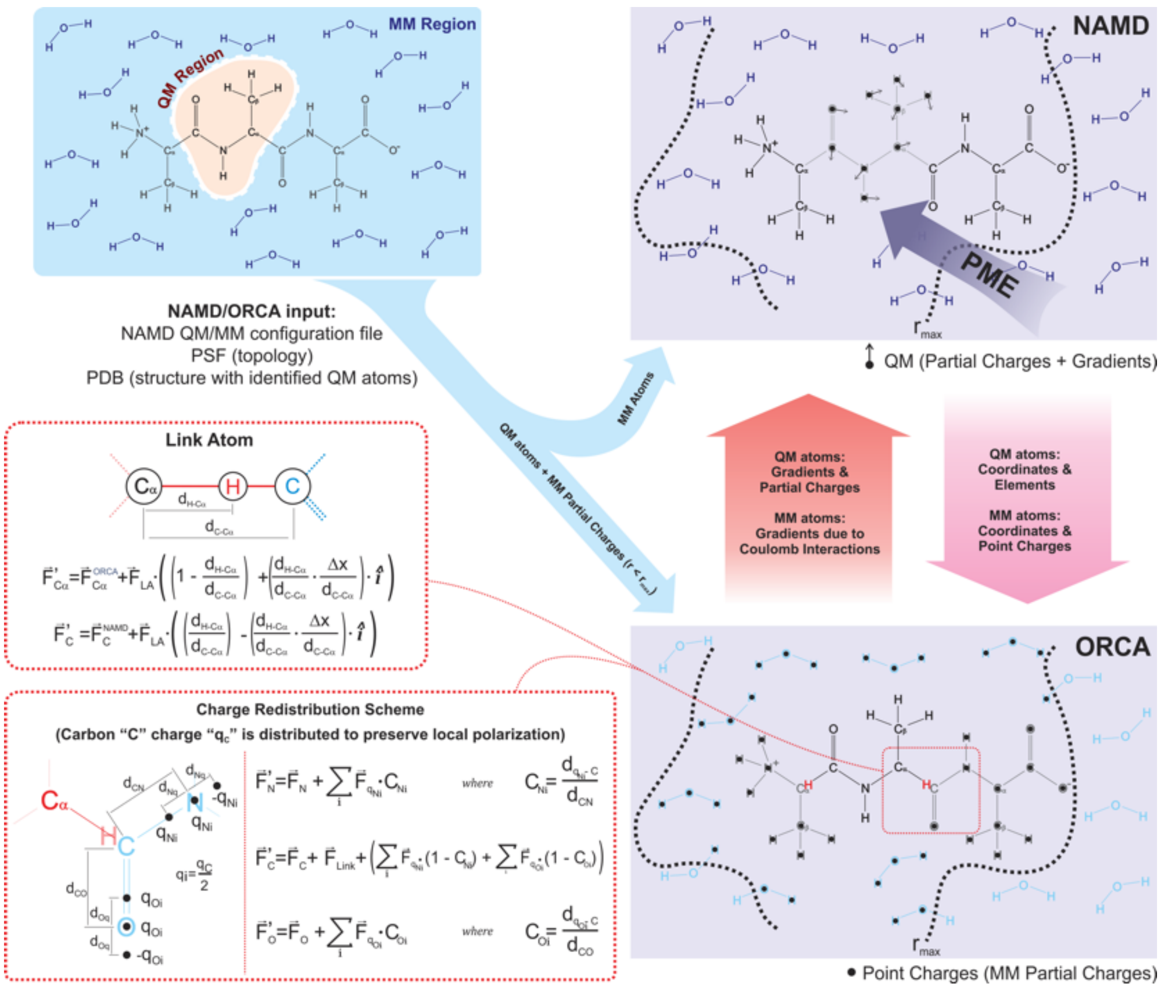
If employing NAMD QM/MM please cite:

> NAMD goes quantum: An integrative suite for hybrid simulations. Melo*, M. C. R.; Bernardi*, R. C.; Rudack T.; Scheurer, M.; Riplinger, C.; Phillips, J. C.; Maia, J. D. C.; Rocha, G. D.; Ribeiro, J. V.; Stone, J. E.; Neese, F.; Schulten, K.; Luthey-Schulten, Z.; Nature Methods, 2018 (doi:10.1038/nmeth.4638)

## 14.1 Division of Labor

The basic idea behind a hybrid QM/MM simulation in NAMD is to use a classical force field to treat the classical atoms in the system (or "MM atoms"), and pass the information that describes the quantum atoms in the system (or "QM atoms") to a Quantum Chemistry (QC) software, which is expected to produce gradients for all QM atoms, as well as the total energy of the QM region (and optionally partial charges). All bonded and non-bonded interactions among MM atoms are handled by NAMD's force field. Similarly, all bonded and non-bonded interactions among QM atoms are handled by the QC software in its chosen theory level. Treatment of covalent bonds between QM and MM atoms will be described in a following section.

The non-bonded interactions between QM and MM atoms are handled differently, and can be modified and regulated by the user. Van der Waals interactions are always calculated, and can be done using either the default force field parameters, or specific (user-defined) parameters for QM atoms. Parameter modifications for QM atoms have been proposed in order to compensate



MM Region
QM Region
NAMD/ORCA input:
NAMD QM/MM configuration file
PSF (topology)
PDB (structure with identified QM atoms)
NAMD
PME
r_max
QM (Partial Charges + Gradients)
MM Atoms
QM atoms + MM Partial Charges (r < r_max)
QM atoms: Gradients & Partial Charges
MM atoms: Gradients due to Coulomb Interactions
QM atoms: Coordinates & Elements
MM atoms: Coordinates & Point Charges
ORCA
Point Charges (MM Partial Charges)
Link Atom
Charge Redistribution Scheme
(Carbon "C" charge "q_c" is distributed to preserve local polarization)
where

Figure 14: Graphical representation of NAMD-ORCA interconnection. Only the contribution of MM charges beyond rmax are calculated by NAMD (via PME), with the direct electrostatic calculation performed by ORCA. The image assumes the charge shift redistribution scheme, where the partial charge of the linking MM atom is shifted to its nearest MM neighbors.

for over-polarization that these atoms may exhibit in hybrid QM/MM simulations. Larger van der Waals radii and/or shallower well depths should then be provided for all element types that occur among QM atoms (see the “qmVdwParams” keyword).

## 14.2 Mechanical and Electrostatic Embedding

Electrostatic interactions between QM and MM atoms deserve a more detailed discussion due to the abundance and diversity of available alternatives. The first decision to be made is whether there will be electrostatic interactions between the two portions of a system, QM and MM. In the “mechanical embedding” scheme, only positions and elements of atoms in the QM region are passed on to the chosen QC software for energy and force calculations. This way, QM and MM atoms share only van der Waals interactions.

In the “electrostatic embedding” scheme, on the other hand, the partial charges of MM atoms surrounding all QM atoms are used to approximate the electrostatic environment where QM atoms are found (the scheme is selected with the “qmElecEmbed” keyword). See Figure 15. This process can be customized in a variety of ways, the first of which is deciding if a smoothing function will be used to avoid an abrupt decay in electrostatic force due to the cutoff used in the selection of surrounding point charges (this option is activated with the “qmSwitching” keyword).

Classical point charge utilization can be further customized by choosing which smoothing function will be used, and if the total charge of selected partial charges should be modified to (A) have a whole charge or (B) have a complementary charge to that of the QM region, so that the sum of charges from QM atoms and classical partial charges add to zero (see Figure 15).

With electrostatic embedding, QM atoms are influenced by the charges in the classical region. In order to balance the forces acting on the system, NAMD uses partial charges for the QM atoms to calculate the electrostatic interaction with classical point charges. There are two possibilities for the origin of the QM partial charges: the original partial charges found in the force field parameter files can be used, or updated partial charges can be gathered at each step from the QC software output (controllable through the “qmChargeMode” keyword). The continuous update in charge distribution allows for a partial re-parameterization of the targeted molecule at each time step, which can lead to an improved description of the interactions of a ligand as it repositions over the surface of a protein, for example, or as it moves through a membrane.

In case PME is activated by the user, NAMD will automatically apply the necessary corrections to handle the QM region, allowing it to be influenced by long range interactions from the entire system.

## 14.3 Covalent Bonds Divided by the QM/MM Barrier

Hybrid QM/MM simulations of biomolecular systems often present situations where only a portion of a molecule should be treated quantum mechanically, usually to save computational resources since the cost of simulating QM regions rises rapidly with the number of simulated toms. In order to deal with chemical bonds that are split by the QM/MM division of the biomolecular system, that is, bonds that have one atom in the quantum (QM) region and another in the classical (MM) region (we will call these “QM/MM bonds”), NAMD makes approximations to the molecular system in order to bridge differences in simulation type (QM vs. MM), and minimize errors involved in the QM/MM division of the system (Figure 16 A and B).

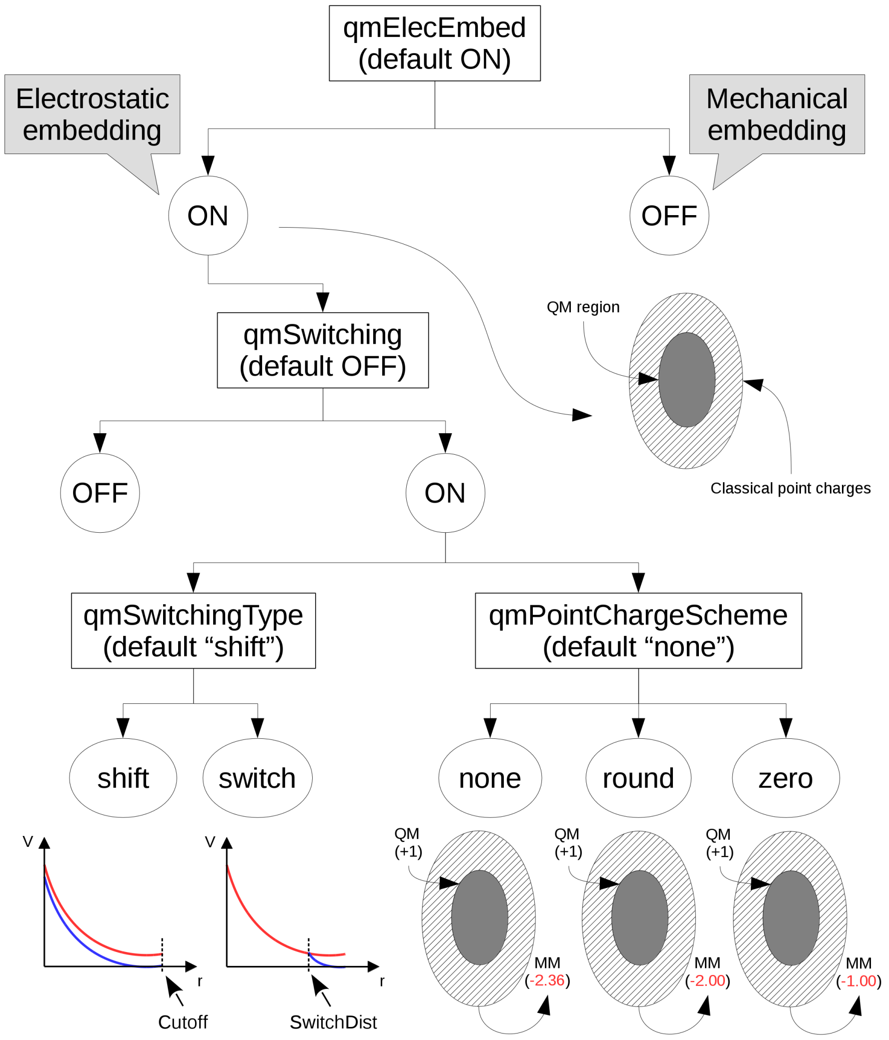


Figure 15: Diagram of options that control the use and manipulation of classical point charges. Default values are indicated below their respective keyword. “Cutoff” and “SwitchDist” are keywords used in NAMD to configure the calculations of electrostatic and van der Waals interactions.

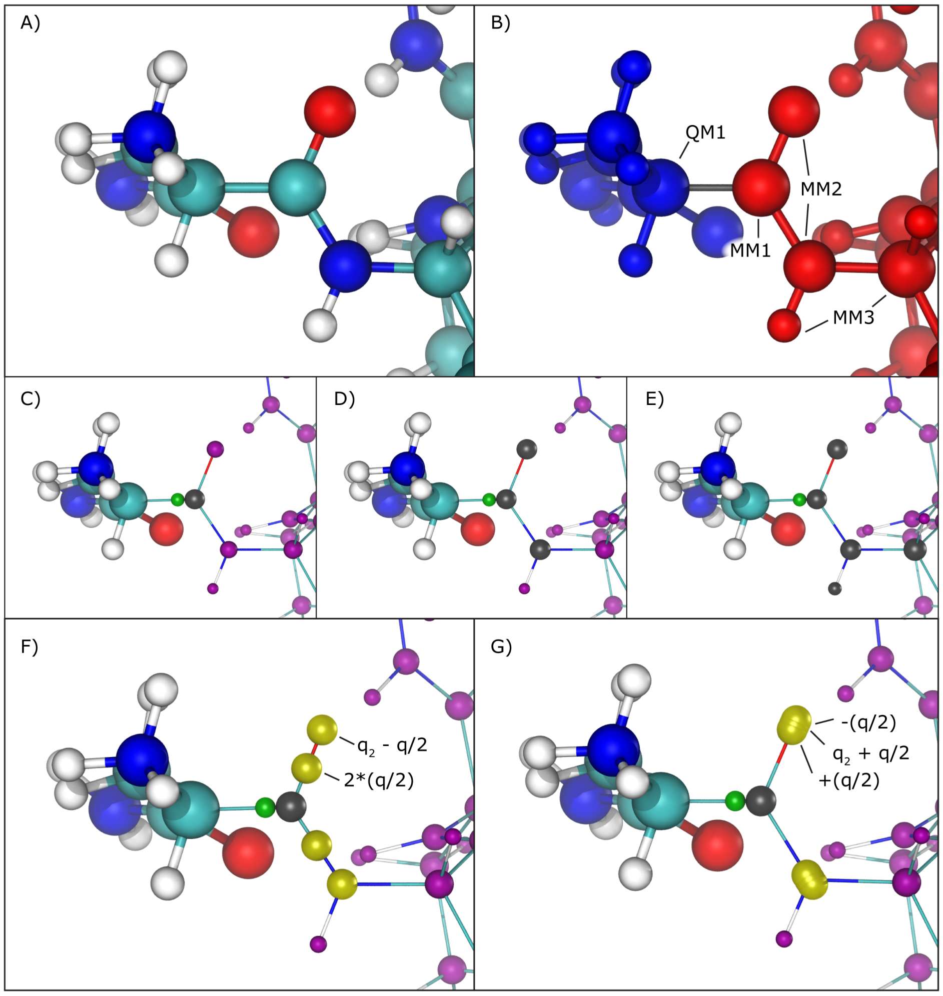


Figure 16: Treatment of QM/MM bonds. A) Illustration of all atoms in the vicinity of the QM/MM bond, colored by element: cyan for carbon, white for hydrogen, blue for nitrogen and red for oxygen. B) To the left, in blue, is the region that will be treated with the chosen QC software. To the right, in red, the region treated classically by NAMD. The bond in gray is the one crossing the QM/MM division. The atom marked as **QM1** is the quantum atom directly connected to the classical atom on the other side of the QM/MM division. Analogously, the atom marked as **MM1** is the classical atom directly connected to the quantum atom on the other side of the QM/MM division. Atoms marked as **MM2** are directly bonded to the **MM1** atom, and atoms marked **MM3** are directly bonded to **MM2** atoms. C) **Z1** method. Ignored partial charges are indicated in the image with a gray sphere taking the place of its respective classical atom. Directly adjacent to MM1 is a green sphere representing the link atom that is placed along the QM1-MM1 covalent bond. All remaining partial charges representing classical atoms that are passed on to the QC software are indicated in purple spheres. D) **Z2** method. E) **Z3** method. F) RCD method. Virtual point charges, are represented in yellow spheres. The text indicates the total charge placed at each position, where **q** indicates the charge of the MM1 atom and **q2** represents the partial charge of the MM2 atom at that position. The yellow spheres at MM2 atom positions indicate their partial charge has been changed from its original value. G) CS method. Since in this case the virtual point charges are placed very close to the MM2 atom position, the yellow spheres representing them show significant overlapping.

### 14.3.1 Link Atoms

As previously mentioned, the information regarding atoms in the QM region is passed on to the chosen QC software, that is, their respective positions and element types, but in order to maintain (or approximate) the effect of the chemical bond between the QM atom and the MM atom, NAMD creates and places a link atom (usually a hydrogen) along the "broken" QM/MM bond. The user can fine-tune this process by choosing the method of placement of the link atom and even the element of such atom (keywords "qmBondDist" and "qmLinkElement").

The introduction of the link atom will invariably place it very near the classical atom involved in the QM/MM bond, therefore the use and placement of partial charges from classical atoms becomes highly relevant. Under the mechanical embedding scheme, the QC software only receives the atoms in the QM region and the link atoms created to approximate QM/MM bonds, so no manipulation of partial charges is required. On the other hand, usual QM/MM simulations are done under the electrostatic embedding scheme, in which case the partial charges of classical atoms involved in the QM/MM bonds and classical atoms directly connected to them require special treatment.

### 14.3.2 Point Charge Alterations

Several methods have been proposed to handle this situation, and the QM/MM interface developed here implements the most widely accepted ones. One can be chosen using the "qmBondScheme" keyword (Figure 16 C to G). In all implemented methods, the classical atom participating in the QM/MM bond (MM1 atom) does not have its partial charge passed on to the QC software, since this would create excessive repulsion (or attraction) on the link atom. This is, in fact, the entirety of the "Z1" method: ignoring the partial charge of the MM1 atom. Analogously, "Z2" and "Z3" ignore all partial charges up to MM2 and MM3 atoms, respectively (Figure 16 C to E).

The Redistributed Charge and Dipole (RCD) method (Figure 16 F) is more elaborate, as it rearranges the partial charge of the MM1 atom (indicated as $q$) so that the total charge of the region is maintained as well as the dipole moments of the bonds between MM1 and MM2 atoms. This is done by creating "virtual" point charges, which are passed on to the QC software as if they represented partial charges of classical atoms. More specifically, the RCD method creates a virtual point charge in the middle of all MM1-MM2 bonds with a charge of $2q/n$, where $n$ is the number of MM2 atoms connected to MM1, and also subtracts a charge $q/n$ from each of the MM2 atoms, so that the total charge of the region remains constant while approximating the dipole moment of the MM1-MM2 bonds. This way there will be no point charge placed at the position of the MM1 atom, but its partial charge is not simply removed, it is redistributed.

A similar approach is taken by the Charge Shifting (CS) method (Figure 16 G). In this case, the MM1 partial charge is equally distributed across the MM2 atoms, and two virtual point charges are placed along the direction of the MM1-MM2 bond, one before the MM2 atom and one after, each one with a charge of $+q/n$ and $-q/n$ respectively. This method will also keep the total charge of the region constant while trying to preserve the local dipoles formed by all MM1-MM2 bonds.

### 14.3.3 Link Atom Charge and Charge Groups

Along with the gradient over all QM atoms, NAMD can also use the partial charge derived from the QC calculation to update the charge distribution of atoms in the QM region. When a QM/MM bond exists, however, part of the charge of the region will be placed on the link atom, and in order to keep the charge of the QM region constant, the link atom charge is re-distributed on the QM region. This seemingly simple mechanism can cause problems unless special care is be taken when deciding which bond will mark the division of QM and MM regions.

Many force fields divide the topologies of biomolecules in "charge groups" (Figure 17 A and B). What this means is that not only will the partial charges of all atoms of a molecule add up to the whole number that represents the charge of the molecule, they will also add up to whole numbers in sub groups of atoms (look for the "GROUP" statements in `http://www.ks.uiuc.edu/Training/Tutorials/namd/namd-tutorial-unix-html/node24.html` to see an example). Therefore, one needs to make sure that the chosen QM/MM bond(s) sits in between "charge groups", so the total sum of partial charges of atoms defining a QM region is a whole number. This is especially important in order to keep the total charge of the system constant. Since the QC calculation will always distribute a whole charge over all atoms of a system (QM atoms plus a link atom), if the partial charge of QM atoms is not initially a whole number, it will be forced into a whole number after the first QC step, where the charge of the link atom is distributed over the QM region. This will create a mismatch between QM and MM charges, changing the total charge of the entire system (QM plus MM regions).

An example can be seen in Figure 17, bonds 1 and 3 are chosen as the QM/MM bonds, the charge distribution seen in Figure 17 C shows a whole charge for the QM region (and consequently for the MM region). Therefore, any charge placed on link atoms can be redistributed to the QM atoms with no change in total system charge. However, if bonds 2 and 3 are chosen for the QM/MM bond (Figure 17 D), the charge of the MM region would be $+1.16$, while the charge of the QM region would be $-1.16$. Since the QC calculation would place a pre-determined whole charge on the region ($-1$, in this case), the updated charge of the QM region after the first simulation step would change the total charge of the system to $+0.16$, in this example.

## 14.4 Custom Quantum Chemistry Software

In order to offer the broad range of tools and technologies present in NAMD to all researchers who develop and/or employ specialized Quantum Chemistry tools, the QM/MM interface is prepared to utilize any QC tool that can be wrapped in a script that converts input and output files to specified formats. This flexible interface will improve development and testing of new tools, as well as their quick integration utilization in hybrid dynamics.

We currently provide in the `libs/qmmm/` directory (and mirrored at `http://www.ks.uiuc.edu/Research/qmmm/Scripts/`) Python wrapper scripts for GAUSSIAN, TeraChem, and Q-Chem. Other wrapper scripts can be generated, based on these templates, in any other language, as long as they are provided to NAMD in an executable form. Although natively supported, we also provide a python wrapper script for ORCA, with extended comments explaining the format in which NAMD will write data for the QC software and the format in which NAMD expects to find the results.

## 14.5 Independent QM Regions

Aiming at large macromolecular simulations that could take advantage of localized QM resolution, NAMD allows the user to set up multiple independent QM regions in the same molecular system. For example, one could study a multimeric complex that contains several active sites and have all active sites be calculated with a chosen QC software simultaneously (Figure 18). Each active site would be calculated independently of all others, by its own execution of the QC software, keeping the calculation cost low and without impacting the overall efficiency of the simulation, since all QM regions would be calculated in parallel.

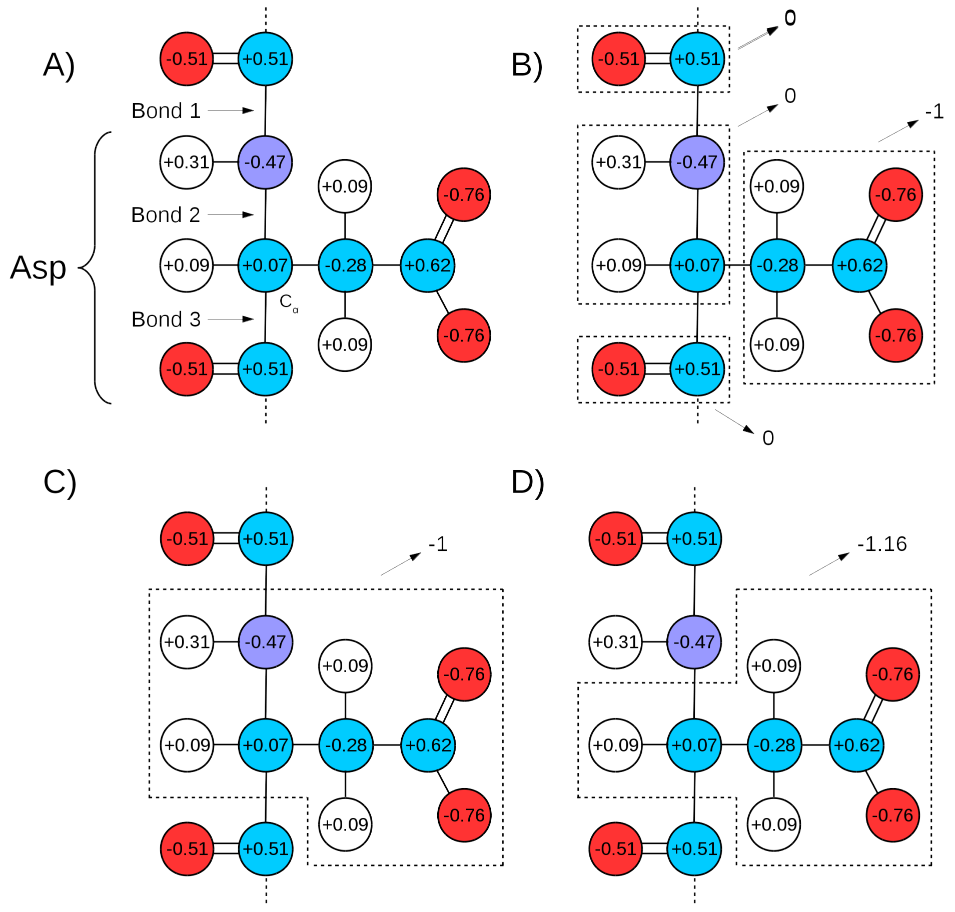


Figure 17: Charge Groups and QM/MM Bonds. A) Illustration of aspartate and the distribution of charge over its atoms as in CHARMM36 force field parameters. Circles in red indicate oxygen atoms, blue indicate nitrogen atoms, cyan for carbon atoms, and white for hydrogen atoms. “Bond 1” indicates the peptide bond, “Bond 2” indicates the one between the alpha carbon and the peptide bond nitrogen, and “Bond 3” the bond between the alpha carbon and the peptide bond carbon. B) Charge groups are indicated with dashed squares, along with their total charges. C) Depiction of the atoms in the QM region if Bonds 1 and 3 are used to separate it from the MM region. The total charge of QM region is $-1$. D) Depiction of QM region if the same is defined by Bonds 2 and 3. In this case, the total charge of QM region is $-1.16$.

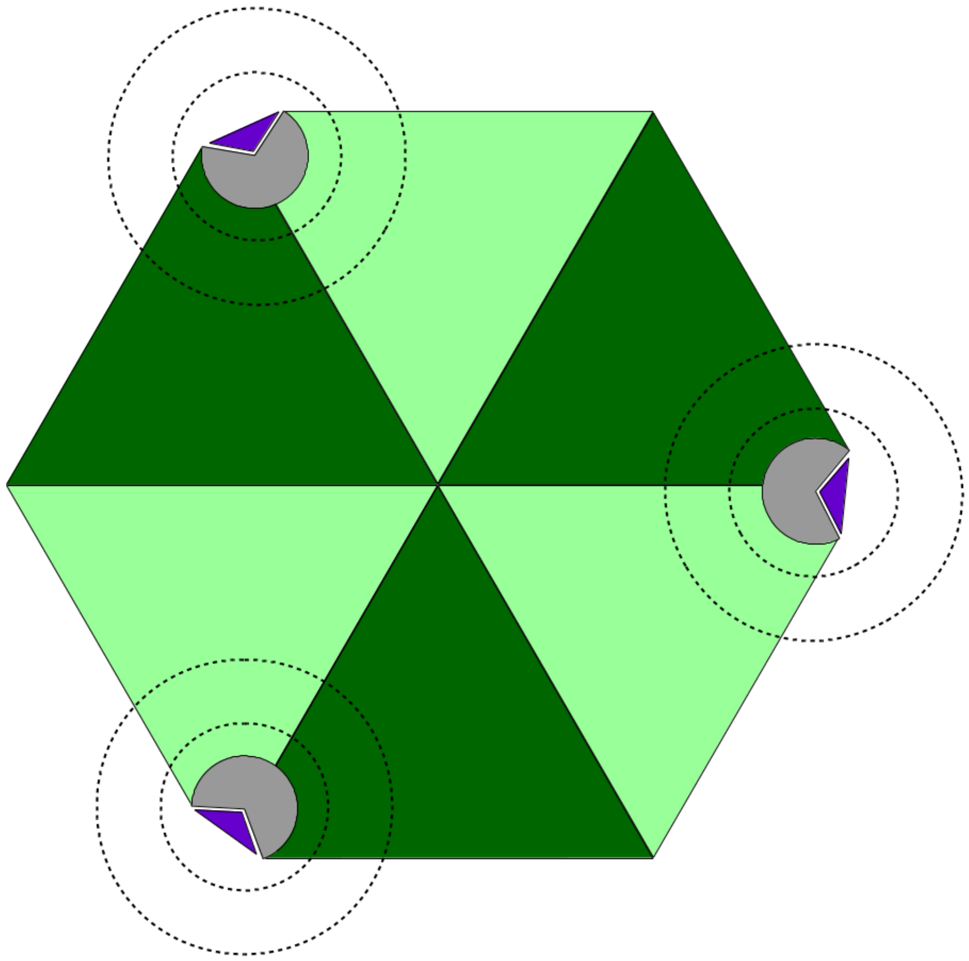


Figure 18: Diagram of Multiple QM Regions. The illustration depicts a hetero-hexameric complex (light and dark green triangles) that combine to create three active sites (in gray). Active sites are bound to a target molecule (purple triangle). The inner and outer dashed circles represent, respectively, the boundary of a QM region and the limits of the classical point charge shell around that region.

Identifying the different QM regions and which atoms belong to each one of them can be simply accomplished in the input PDB file, or in a dedicated PDB file (keyword “qmParamPDB”). Since each region can contain different sets of molecules, their charges and multiplicities are indicated separately (see keywords “qmCharge” and “qmMult”).

For simulations of large systems that are distributed across several computer nodes, one can control how many independent QM regions are calculated in each node. This would prevent large simulations from running out of memory if two or more large QM regions are placed in the same node (see keyword “qmSimsPerNode”).

## 14.6 Keywords

- `qmForces` < Calculate QM? >
  **Acceptable Values:** yes or no
  **Default Value:** no
  **Description:** Turns on or off the QM calculations.

- `qmParamPDB` < Set QM atoms >
  **Acceptable Values:** PDB file
  **Description:** Name of an optional secondary PDB file where the OCCupancy or BETA column has the indications for QM or MM atoms. QM atoms should have an integer bigger than zero (0) and MM atoms should have zero as the beta or occupancy field. The same file may have indications for bonds between a QM atom and an MM atom. This should be provided in case the PDB file passed to the “coordinates” keyword already has data on its beta or occupancy columns, such as when a SMD simulations is being performed.

- `qmColumn` < Which column? >
  **Acceptable Values:** “beta” or “occ”
  **Description:** Indicates which column has the QM/MM field. Required.

- `qmSimsPerNode` < Sims per node >
  **Acceptable Values:** postive integer
  **Default Value:** 1
  **Description:** Number of independent simultaneous QM simulations per node.

- `qmBondColumn` < Which bond column? >
  **Acceptable Values:** “beta” or “occ”
  **Description:** Indicates which column has the QM/MM bond information. This will tell NAMD which atoms are at the ends of a covalent bond split by the QM/MM barrier, with one atom being quantum and one being classical. There is no default value. If this parameter is provided, NAMD will parse options for dealing with QM/MM bonds.

- `qmBondDist` < Use qmBondColumn value for distance? >
  **Acceptable Values:** “on” or “off”
  **Default Value:** off
  **Description:** Indicates whether the value in the BondColumn will be used to define the

distance between the QM atom and the Link Atom that will replace the MM atom in the QM system.

- `qmBondValueType` < Does qmBondColumn value give length or ratio? >
  **Acceptable Values:** "len" or "ratio"
  **Default Value:** len
  **Description:** Indicates if the values in the BondColumn represent either the length ("len") between the QM and link atoms or the ratio ("ratio") between the QM-MM distance and the one which will be used as the QM-Link distance.

- `qmLinkElement` < Set link atom element >
  **Acceptable Values:** string, for example "4 9 Cl"
  **Default Value:** H
  **Description:** User defined link atom element. Two syntaxes are allowed: if there is only one QM-MM bond, a string with the element symbol is allowed (such as "H" or "Cl"). If there are two or more bonds, the string needs to have the two atoms that compose the bond, and then the element (such as "4 9 Cl"). The default element for all link atoms is hydrogen.

- `qmBondScheme` < Select QM-MM bond scheme >
  **Acceptable Values:** "CS" or "RCD" or "Z1" or "Z2" or "Z3"
  **Default Value:** CS
  **Description:** Indicates what will be the treatment given to QM-MM bonds in terms of charge distribution and link atom creation and placement. CS: Charge Shift Scheme; RCD: Redistributed Charge and Dipole method; Z1: Only ignored MM1 partial charge, with no charge redistribution; Z2: Ignores MM1 and all MM2 partial charges, with no charge redistribution; Z3: Ignores MM1 and all MM2 and MM3 partial charges, with no charge redistribution.

- `qmBondGuess` < Guess QM-MM bonds from topology and QM atom definitions >
  **Acceptable Values:** yes or no
  **Default Value:** no
  **Description:** With this option, NAMD will ONLY use QM region definitions and topology data to determine QM-MM bonds. NAMD will ignore the user-provided QM bond column values (in case the user assigned a value to the qmBondColumn variable and populated the column in the PDB file).

- `qmElecEmbed` < Should point charges be used in QM? >
  **Acceptable Values:** "on" or "off"
  **Default Value:** on
  **Description:** Indicates if classical point charges should be used in QM calculations.

- `qmSwitching` < Use switching on point charges? >
  **Acceptable Values:** "on" or "off"
  **Default Value:** off
  **Description:** This will scale down the point charges representing the classical system

as to replicate the switching procedure that NAMD applies to all charged interaction (see “switching”).

- `qmSwitchingType` < Set functional form of switching >
**Acceptable Values:** “switch” or “shift”
**Default Value:** shift
**Description:** This option is used to decide which kind of function will be used to scale down point charges sent to QM calculations. SHIFT: This will ”shift down” the entire shell of point charges so that electrostactic interactions reach zero at the cutoff distance. SWITCH: This will only change point charges in the sub-volume between the switchdist and cutoff distance, so that electrostactic interactions reach zero at the cutoff distance.

- `qmPointChargeScheme` < Set point charge scheme >
**Acceptable Values:** “none” or “round” or “zero”
**Default Value:** none
**Description:** This option allows the user to decide if and how the point charges presented to the QM system will be altered. NONE: Nothing will be done. ROUND: This will change the most distant point charges so that the total sum of point charges is a whole number. ZERO: This will adjust the most distant point charges so that the total sum of point charges is 0.

- `qmBaseDir` < Set directory for file I/O >
**Acceptable Values:** directory path
**Description:** This should be a fast read/write location, such as a RAM drive (/dev/shm on most linux distros). The user needs to make sure this directory exists in the node(s) running the QM calculation(s).

- `qmReplaceAll` < Use only QM gradients for forces? >
**Acceptable Values:** “on” or “off”
**Default Value:** off
**Description:** Indicates to NAMD that ALL forces form NAMD will be ignored and only the gradients from the QM software will be applied on the atoms. This IS NOT NECESSARY in any regular QM/MM simulation, and will prevent the use of any other feature from NAMD such as SMD.

- `qmVdwParams` < Modify type names for QM atoms? >
**Acceptable Values:** “on” or “off”
**Default Value:** off
**Description:** The QM code will change all QM atoms’ van der Waals types to ”q”+element (e.g., all carbons will be qC and all hydrogens will be qH) for VdW interactions. This means that a parameter file with epsilon and sigma values for these atom types must be provided along with the regular parameter files. For example, if using CHARMM force field, the new file should be in CHARMM format.

- `qmPCStride` < Set stride for point charge >
**Acceptable Values:** integer

**Default Value:** 1
**Description:** Sets a stride for new point charge determination. The same set of classical atoms will be sent to QM calculations as point charges, but with updated positions.

- `qmCustomPCSelection` < Provide custom point charge selection? >
  **Acceptable Values:** "on" or "off"
  **Default Value:** off
  **Description:** Indicates that one or more file(s) will be provided with a custom selection of point charges. Each file will have a selection for a single QM group. This selection will be kept during the entire simulation.

- `qmCustomPCFile` < File for custom point charge selection >
  **Acceptable Values:** PDB file
  **Description:** The file will have, in the "qmColumn", the same QM ID provided for a single QM group. All other groups will have zero (0) in this column. In the second column (beta or occupancy), the classical or quantum atoms (from other QM regions) that need to be passed as point charges will be identified by a non-zero number.
  **Example/Format:**
  ```
  qmCustomPCFile system/system.customPC.1.pdb
  qmCustomPCFile system/system.customPC.2.pdb
  qmCustomPCFile system/system.customPC.3.pdb
  qmCustomPCFile system/system.customPC.4.pdb
  ```

- `qmLiveSolventSel` < Keep track of solvent? >
  **Acceptable Values:** "on" or "off"
  **Default Value:** off
  **Description:** With Live Solvent Selection (LSS), NAMD will automatically keep track of the solvent molecules for all QM Groups, and will exchange classical solvent molecules with QM solvent molecules every "QMLSSFreq" steps.

- `qmLSSResname` < Set residue name for LSS >
  **Acceptable Values:** residue name
  **Default Value:** TIP3
  **Description:** Indicates which residue name will be used in LSS.

- `qmLSSFreq` < Set frequency of LSS >
  **Acceptable Values:** integer, multiple of stepspercycle
  **Default Value:** 100
  **Description:** Frequency of LSS. Must be a multiple of stepspercycle.

- `qmLSSMode` < How solvent molecules are selected >
  **Acceptable Values:** "dist" or "COM"
  **Default Value:** dist
  **Description:** For LSS, this indicates how solvent molecules are selected. In all cases, the closest solvent molecules are selected, and if a classical solvent molecule is closer than a QM one, they are swaped. DIST: This mode will use the smallest distance between a solvent atom

and a non-solvent QM atom to sort solvent molecules. This is best used when the non-solvent QM atoms form irregular volumes (when the COM is not very representatve), and/or volumes with high solvent accessibility (such as a drug, or a small peptide, in solution). COM: This mode will sort solvent molecules based on Center Of Mass distance between the solvent COM and the COM of a selection for each QM group (see "qmLSSRef" keyword). Best used with small QM regions that have limited solvent accessibility, such as an active site.

- `qmLSSRef` < Which residues for COM of QM atoms? >
  **Acceptable Values:** string
  **Description:** This will indicate which residues are to be used in the determination of the COM of non-solvent QM atoms. Only these atoms will be used to determine the closest set of solvent molecules. The keyword takes a string composed of the QM group ID, the segment name and the residue ID.
  **Example/Format:**
  ```
  qmLSSRef "1 RP1 9"
  qmLSSRef "2 RP1 3"
  qmLSSRef "2 RP1 2"
  qmLSSRef "3 AP1 9"
  qmLSSRef "3 AP1 3"
  qmLSSRef "4 AP1 9"
  ```

- `qmConfigLine` < Pass string to QM configuration >
  **Acceptable Values:** string
  **Description:** The string passed to "qmConfigLine" will be copied and pasted at the very begining of the configuration file for the chosen QM software if either ORCA or MOPAC are selected.
  **Example/Format (QM/MM NAMD-ORCA):**
  ```
  qmConfigLine "!  PM3 ENGRAD"
  qmConfigLine "%%output PrintLevel Mini Print\[P_Mulliken\] 1 Print\[P_AtCharges_M\] 1 end"
  ```
  **Example/Format (QM/MM NAMD-MOPAC):**
  ```
  qmConfigLine "PM7 XYZ T=2M 1SCF MOZYME CUTOFF=9.0 AUX LET GRAD QMMM GEO-OK"
  qmConfigLine "Test System"
  ```

- `qmMult` < Set multiplicity of QM region >
  **Acceptable Values:** string
  **Description:** Multiplicity of the QM region. This is needed for proper construction of ORCA's input file. Each string must be composed of the QM region ID and its multiplicity.
  **Example/Format:**
  ```
  qmMult "1 1"
  qmMult "2 1"
  qmMult "3 1"
  qmMult "4 1"
  ```

- `qmCharge` < Set charge of each QM region >
  **Acceptable Values:** string
  **Description:** Indicates the charge of each QM region. If no charge is provided for a QM region, NAMD calculates the total charge automatically based on the given parameter set. Each string must be composed of the QM region ID and its total charge.
  **Example/Format:**

```
qmCharge "1 1"
qmCharge "2 -1"
qmCharge "3 1"
qmCharge "4 -1"
```

- `qmSoftware` < Which QM software? >
  **Acceptable Values:** "mopac" or "orca" or "custom"
  **Description:** Required for QM/MM, this indicates which QM software should be used. In case the user wants to apply another QM code, this can be done by using the "custom" qmSoftware. In this case, NAMD will call the executable defined in the qmExecPath variable and will give it one argument: the full path to the input file.

  INPUT: This input file will contain on the first line the number of QM atoms (X) and the number of point charges in the file (Y, which may be 0 or more), separated by a space. The following X+Y lines will have four (4) fields: X, Y and Z coordinates, and a fourth field which will depend on the type of entry. For QM atoms, the field will contain the element of the QM atom. For point charge lines, the field will contain the charge of the point charge.

  OUTPUT: The expected output file whould be placed in the same directory as the input file, and should be named `"*inputfile*.result"` (meaning it will have the same path and name of the input file, plus the suffix `".result"`). This file should have, on its first line, the energy of the system and the number of point charges that were passed to ORCA, and that ORCA calculated forces on (zero, if using mechanical embedding). The two numbers should be separated by a single white space. Following the standard for the INPUT file, there will be another X+Y lines in the OUTPUT file. On the following X lines (where X is the number of QM atoms passed in the input file), there must be four (4) fields: the x, y and z components of the TOTAL FORCE applied on that atom, and on the fourth field, the charge of the atom. If the user indicates that charges from the QM software should not be used (see "qmChargeMode"), the fourth field should have zeroes, but should not be empty. On the following Y lines (where Y is the number of point charges), there must be only three (3) fields: the x, y and z components of the electrostatic force applied on that point charge. Energy should be in Kcal/mol and forces in Kcal/mol/Angstrom.

- `qmExecPath` < Set path to QM executable >
  **Acceptable Values:** path
  **Description:** Required for QM/MM, this indicates the path to the QM code executable.

- `qmSecProc` < Set path to secondary executable >
  **Acceptable Values:** path
  **Description:** Indicates a secondary executable that NAMD will call AFTER each QM software execution for each QM group. The executable is called with two arguments: the complete path and name of the input file used for each QM software execution; and the simulation step. This option can be used for an extra-processing at every step, e.g., for saving all QM outputs every step.

- `qmPrepProc` < Set path to initial executable >
  **Acceptable Values:** path
  **Description:** Indicates an executable that must be called BEFORE the FIRST QM software

execution of each QM group. The executable is called with one argument: the complete path and name of the input file used for each QM software execution. This can be used to setup a charge distribution for a molecule of interest, for example.

- `qmChargeMode` < Set charge calculation mode >
  **Acceptable Values:** "none" or "mulliken" or "chelpg"
  **Default Value:** mulliken
  **Description:** Charge calculation mode expected from the QM software. This indicates if charges should be read from the QM software and updated at every step, or if the original force field atom charges should be used. In case you are using ORCA, two charge options are allowed, Mulliken or CHELPG. We must know the kind of charge requested by the user so that the proper format is expected, read and processed. NONE: No charges are read from the QM software output and the original force field charges are preserved. MULLIKEN: This is the only other option for MOPAC and one possibility for ORCA. In case you are using the custom QM software interface, choose this option in order to use the charges calculated in the custom QM software, irrespective of the actual theory used for that calculation. CHELPG: This is a second possibility for ORCA.

- `qmOutStride` < Set frequency of QM charge output >
  **Acceptable Values:** integer
  **Default Value:** 0 (not saving)
  **Description:** Frequency of QM charge output. A dedicated DCD file will be created to store the charge data for all QM atoms in the system. This independent frequency allows the user to store whole-system data at a larger stride to save time and space.

- `qmPositionOutStride` < Set frequency of QM-only position output >
  **Acceptable Values:** integer
  **Default Value:** 0 (not saving)
  **Description:** Frequency of QM-only position output. A dedicated DCD file will be created to store the position data for all QM atoms in the system. This independent frequency allows the user to store whole-system data at a larger stride to save time and space.

- `qmEnergyStride` < Set frequency of QM specific energy output >
  **Acceptable Values:** integer
  **Default Value:** 1
  **Description:** Frequency of QM-only energy output. A dedicated energy output line will be created to indicate the energy calculated by the QM code. This independent frequency allows the user to store QM-specific energy data at a larger stride to save time and space.

- `qmChargeFromPSF` < Set charge of QM region from PSF file >
  **Acceptable Values:** "on" or "off"
  **Default Value:** off
  **Description:** Automatically determine charge of QM regions by adding the charges of atoms in each region.

- `qmCSMD` < Apply conditional-SMD to QM atoms? >

**Acceptable Values:** "on" or "off"
**Default Value:** off
**Description:** Apply conditional SMD to QM atoms in order to steer the simulation within the QM region, while avoiding bringing atoms too close together and destabilizing the molecule. C-SMD works like regular SMD, but with pairs of atoms. The first atom is pulled by a string connected to a virtual particle, and the direction of motion of the virtual particle is updated to follow a second atom. The force on the first atom will stop being applied when they come closer than a cutoff value.

- `qmCSMDFile` < Set cSMD information >
  **Acceptable Values:** cSMD file
  **Description:** Name of a text file indicating pairs of atoms that will be brought closer in space. In the file, each line defines a cSMD bias, with the following syntax:
  Atom1 Atom2 Force(Kcal/Mol/A) Speed(A/step) Cutoff(A)

# 15 Runtime Analysis

## 15.1 Pair interaction calculations

NAMD supportes the calculation of interaction energy calculations between two groups of atoms. When enabled, pair interaction information will be calculated and printed in the standard output file on its own line at the same frequency as energy output. The format of the line is `PAIR INTERACTION: STEP:` *step* `VDW_FORCE:` *fx fy fz* `ELECT_FORCE:` *fx fy fz*. The displayed force is the force on atoms in group 1 and is units of kcal/mol/Å.

For trajectory analysis the recommended way to use this set of options is to use the NAMD Tcl scripting interface as described in Sec. 2.2.2 to run for 0 steps, so that NAMD prints the energy without performing any dynamics.

- `pairInteraction` < is pair interaction calculation active? >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** Specifies whether pair interaction calculation is active.

- `pairInteractionFile` < PDB file containing pair interaction flags >
  **Acceptable Values:** UNIX filename
  **Default Value:** `coordinates`
  **Description:** PDB file to specify atoms to use for pair interaction calculations. If this parameter is not specified, then the PDB file containing initial coordinates specified by `coordinates` is used.

- `pairInteractionCol` < column of PDB file containing pair interaction flags >
  **Acceptable Values:** `X`, `Y`, `Z`, `O`, or `B`
  **Default Value:** `B`
  **Description:** Column of the PDB file to specify which atoms to use for pair interaction calculations. This parameter may specify any of the floating point fields of the PDB file, either X, Y, Z, occupancy, or beta-coupling (temperature-coupling).

- `pairInteractionSelf` < compute within-group interactions instead of bewteen groups >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** When active, NAMD will compute bonded and nonbonded interactions only for atoms within group 1.

- `pairInteractionGroup1` < Flag to indicate atoms in group 1? >
  **Acceptable Values:** integer
  **Description:**

- `pairInteractionGroup2` < Flag to indicate atoms in group 2? >
  **Acceptable Values:** integer
  **Description:** These options are used to indicate which atoms belong to each interaction group. Atoms with a value in the column specified by `pairInteractionCol` equal to `pairInteractionGroup1` will be assigned to group 1; likewise for group 2.

## 15.2 Pressure profile calculations

NAMD supports the calculation of lateral pressure profiles as a function of the z-coordinate in the system. The algorithm is based on that of Lindahl and Edholm (JCP 2000), with modifications to enable Ewald sums based on Sonne et al (JCP 122, 2005).

The simulation space is partitioned into slabs, and half the virial due to the interaction between two particles is assigned to each of the slabs containing the particles. This amounts to employing the Harasima contour, rather than the Irving-Kirkwood contour, as was done in NAMD 2.5. The diagonal components of the pressure tensor for each slab, averaged over all timesteps since the previous output, are recorded in the NAMD output file. The units of pressure are the same as in the regular NAMD pressure output; i.e., bar.

The total virial contains contributions from up to four components: kinetic energy, bonded interactions, nonbonded interactions, and an Ewald sum. All but the Ewald sums are computed online during a normal simulation run (this is a change from NAMD 2.5, when nonbonded contributions to the Ewald sum were always computed offline). If the simulations are performed using PME, the Ewald contribution should be estimated using a separate, offline calculation based on the saved trajectory files. The nonbonded contribution using a cutoff different from the one used in the simulation may also be computed offline in the same fashion as for Ewald, if desired.

Pressure profile calculations may be performed in either constant volume or constant pressure conditions. If constant pressure is enabled, the slabs thickness will be rescaled along with the unit cell; the dcdUnitCell option will also be switched on so that unit cell information is stored in the trajectory file.

NAMD 2.6 now reports the lateral pressure partitioned by interaction type. Three groups are reported: kinetic + rigid bond restraints (referred to as "internal", bonded, and nonbonded. If Ewald pressure profile calculations are active, the Ewald contribution is reported in the nonbonded section, and no other contributions are reported.

NAMD 2.6 also permits the pressure profile to be partitioned by atom type. Up to 15 atom groups may be assigned, and individual contribution of each group (for the "internal" pressures) and the pairwise contributions of interactions within and between groups (for the nonbonded and bonded pressures) are reported in the output file.

- `pressureProfile` < compute pressure profile >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** When active, NAMD will compute kinetic, bonded and nonbonded (but not reciprocal space) contributions to the pressure profile. Results will be recorded in the NAMD output file in lines with the format `PRESSUREPROFILE: ts Axx Ayy Azz Bxx Byy Bzz ...` , where `ts` is the timestep, followed by the three diagonal components of the pressure tensor in the first slab (the slab with lowest $z$), then the next lowest slab, and so forth. The output will reflect the pressure profile averaged over all the steps since the last output.

  NAMD also reports kinetic, bonded and nonbonded contributions separately, using the same format as the total pressure, but on lines beginning with `PPROFILEINTERNAL`, `PPROFILEBONDED`, and `PPROFILENONBONDED`.

- `pressureProfileSlabs` < Number of slabs in the spatial partition >
  **Acceptable Values:** Positive integer
  **Default Value:** 10

**Description:** NAMD divides the entire periodic cell into horizontal slabs of equal thickness; `pressureProfileSlabs` specifies the number of such slabs.

- `pressureProfileFreq` < How often to output pressure profile data >
  **Acceptable Values:** Positive integer
  **Default Value:** 1
  **Description:** Specifies the number of timesteps between output of pressure profile data.

- `pressureProfileEwald` < Enable pressure profile Ewald sums >
  **Acceptable Values:** `on` or `off`
  **Default Value:** `off`
  **Description:** When enabled, only the Ewald contribution to the pressure profile will be computed. For trajectory analysis the recommended way to use this option is to use the NAMD Tcl scripting interface as described in Sec. 2.2.2 to run for 0 steps, so that NAMD prints the pressure profile without performing any dynamics.

  The Ewald sum method is as described in Sonne et al. (JCP 122, 2005). The number of $k$ vectors to use along each periodic cell dimension is specified by the `pressureProfileEwald`$n$ parameters described below.

- `pressureProfileEwaldX` < Ewald grid size along X >
  **Acceptable Values:** Positive integer
  **Default Value:** 10
  **Description:**

- `pressureProfileEwaldY` < Ewald grid size along Y >
  **Acceptable Values:** Positive integer
  **Default Value:** 10
  **Description:**

- `pressureProfileEwaldZ` < Ewald grid size along Z >
  **Acceptable Values:** Positive integer
  **Default Value:** 10
  **Description:**

- `pressureProfileAtomTypes` < Number of atom type partitions >
  **Acceptable Values:** Positive integer
  **Default Value:** 1
  **Description:** If `pressureProfileAtomTypes` is greater than 1, NAMD will calculate the separate contributions of each type of atom to the internal, bonded, nonbonded, and total pressure. In the case of the internal contribution, there will be $n$ pressure profile data sets reported on each `PPROFILEINTERNAL` line, where $n$ is the number of atom types. All the partial pressures for atom type 1 will be followed by those for atom type 2, and so forth. The other three pressure profile reports will contain $n(n+1)/2$ data sets. For example, if there are $n = 3$ atom types, the six data sets arising from the three inter-partition and the three intra-partition interactions will be reported in the following order: 1–1, 1–2, 1–3, 2–2, 2–3, 3–3. The total pressure profile, reported on the `PRESSUREPROFILE` line, will contain the internal contributions in the data sets corresponding to 1–1, 2–2, etc.

- `pressureProfileAtomTypesFile` < Atom type partition assignments >
  **Acceptable Values:** PDB file
  **Default Value:** coordinate file
  **Description:** If `pressureProfileAtomTypes` is greater than 1, NAMD will assign atoms to types based on the corresponding value in `pressureProfileAtomTypesCol`. The type for each atom must be strictly less than `pressureProfileAtomTypes`!

- `pressureProfileAtomTypesCol` < `pressureProfileAtomTypesFile` PDB column >
  **Acceptable Values:** PDB file
  **Default Value:** B
  **Description:**

Here is an example snippet from a NAMD input that can be used to compute the Ewald component of the pressure profile. It assumes that the coordinates were saved in the dcd file `pp03.dcd`) every 500 timesteps.

```
Pme             on
PmeGridSizeX    64
PmeGridSizeY    64
PmeGridSizeZ    64

exclude         scaled1-4
oneFourScaling  1.0

switching on
switchdist      9
cutoff          10
pairlistdist    11

pressureProfile        on
pressureProfileSlabs   30
pressureProfileFreq    100
pressureProfileAtomTypes 6
pressureProfileAtomTypesFile atomtypes.pdb
pressureProfileEwald  on
pressureProfileEwaldX  16
pressureProfileEwaldY  16
pressureProfileEwaldZ  16

set ts 0
firstTimestep $ts

coorfile open dcd pp03.dcd
while { [coorfile read] != -1 } {
  incr ts 500
  firstTimestep $ts
  run 0
}
coorfile close
```

# 16 Performance Tuning

## 16.1 NAMD performance tuning concepts

The simulation performance obtained from NAMD depends on many factors. The particular simulation protocol being run is one of the largest single factors associated with NAMD performance, as different simulation methods invoke different code that can have substantially different performance costs, potentially with a different degree of parallel scalability, message passing activity, hardware acceleration through the use of GPUs or CPU vectorization, and other attributes that also contribute to overall NAMD performance.

**Measuring performance.** When NAMD first starts running, it does significant I/O, FFT tuning, GPU context setup, and other work that is unrelated to normal simulation activity, so it is important to measure performance only when NAMD has completed startup and all of the processing units are running at full speed. The best way to measure NAMD performance accurately requires running NAMD for at least 500 to 1,000 steps of normal dynamics (not minimization), so that load balancing has a chance to take place several times, and all of the CPUs and GPUs have ramped up to 100% clock rate. NAMD provides "Benchmark time:" and "TIMING:" measurements in its output, which can be used for this purpose. Here, we are only interested in the so-called wall clock time.

Modern GPU performance is now so fast, especially for GPU-resident mode, that it is expected to take a much longer run than 1,000 steps for hardware to achieve a steady thermal state and power utilization reflecting true long-term simulation performance behavior. The following config parameter can be used to run dynamics for some number of seconds.

- `benchmarkTime` < Perform benchmark for indicated number of seconds >
  **Acceptable Values:** positive integer
  **Description:** Use for benchmarking dynamics. Terminate simulation after running for the indicated number of seconds, where the termination condition is checked at the next "TIMING:" output.

A reasonable benchmark length might be in the range of 120 to 180 (two to three minutes) to provide enough time for the hardware to fully warm up. If `outputPerformance` is left enabled (the default setting) the final "PERFORMANCE:" line will show the average ns/day performance. A good value for `outputTiming` might be 100, 500, or even 1000, depending on the system size and hardware capability, where performance monitoring should probably not be done more than once per second of wall time. These practices are now the preferred way to benchmark GPU-accelerated builds of NAMD, because the measurements shown by the "Benchmark time:" output lines are gathered too quickly in the simulation to accurately determine true performance.

**NAMD configuration and I/O performance.** Aside from the choice of major simulation protocol and associated methods in use, it is also important to consider the performance impacts associated with routine NAMD configuration parameters such as those that control the frequency of simulation informational outputs and various types of I/O. Simulation outputs such as energy information may require NAMD to do additional computations above and beyond standard force evaluation calculations. We advise that NAMD simulation configuration parameters be selected such that output of energies (via the `outputEnergies` parameter) be performed only as much as is strictly necessary, since they otherwise serve to slow down the simulation due to the extra calculations they require. NAMD writes "restart" files to enable simulations that were terminated unexpectedly (for any reason) to be conveniently restarted from the most recently written restart file available. While it is desirable to have a relatively recent restart point to continue from, writing restart information costs NAMD extra network communication and disk I/O. If restart files are written too frequently, this extra activity and I/O will slow down the simulation. A reasonable estimate for restart frequency is to choose the value such that NAMD writes restart files about once every ten minutes of wall clock time. At such a rate, the extra work and I/O associated with writing the restart files should remain an insignificant factor in NAMD performance.

**Computational (arithmetic) performance.** NAMD is provided in a variety of builds that support platform-specific techniques such as CPU vectorization and GPU acceleration to achieve higher arithmetic performance, thereby increasing NAMD simulation throughput. Whenever possible NAMD builds should be compiled such that CPU vector instructions are enabled, and highly tuned platform-specific NAMD code is employed for performance-critical force computations. The so-called "SMP" builds of NAMD benefit from reduced memory use and can in many cases perform better overall, but one trade-off is that the communication thread is unavailable for simulation work. NAMD performance can be improved by explicitly setting CPU affinity using the appropriate Charm++ command line flags, e.g., `++ppn 7 +commap 0,8 +pemap 1-7,9-15` as an example.

It is often beneficial to reserve one CPU core for the operating system, to prevent harmful operating system noise or "jitter", particularly when running NAMD on large scale clusters or supercomputers. The Cray `aprun -r 1` command reserves and forces the operating system to run on the last CPU core.

State-of-the-art compute-optimized GPU accelerators, can provide NAMD with simulation performance equivalent to several CPU sockets (on the order of 100 CPU cores) when used to greatest effect, e.g., when GPUs have sufficient work per GPU. In general, effective GPU acceleration currently requires on the order of 10,000 to 100,000 atoms per GPU assuming a fast network interconnect. NAMD GPU-offload mode requires several CPU cores to drive each GPU effectively, ensuring that there is always work ready and available for the GPU. For contemporary CPU and GPU hardware, the most productive ratios of CPU core counts per GPU might range from 8:1 to 64:1 or higher depending on the details of the hardware involved. GPU-offload mode running on modern GPU hardware will be bottlenecked by the CPU, which means that it is especially important to follow NUMA domain mapping considerations and generally avoid running other than one hardware thread per core (i.e., SMT 1, avoiding use of "hyper-threading"). An advantageous hardware configuration might have one NUMA domain per GPU, which is used most effectively by scheduling one SMP rank per GPU device / NUMA domain, again leaving at least one core per NUMA domain available for the communication thread.

GPU-resident mode shifts almost all computational work to the GPU so generally requires less CPU core support per device than GPU-offload mode, where the CPU is used primarily for GPU kernel management and doing infrequent bursts of activity related to atom migration and file I/O or possibly doing frequent bursts of activity if some host-based computation such as Colvars is used. GPU-resident mode uses shared-memory parallelism but is capable of scaling across multiple GPUs on the same physical node when the GPUs are interconnected by a high-speed networking fabric, such as NVLink or NVSwitch for NVIDIA or Infinity Fabric for AMD. For standard simulations, CPU involvement is low enough to be unaffected by NUMA domain issues. However, heavier per-step use of the CPU cores, such as using Colvars with GPU-resident mode, might benefit from mapping device PEs to a shared NUMA domain.

**Networking performance.** When running NAMD on more than a single node, it is important to use a NAMD version that is optimal for the underlying network hardware and software you intend to run on. The Charm++ runtime system on which NAMD is based supports a variety of underlying networks, so be sure to select a NAMD/Charm++ build that is most directly suited for your hardware platform. In general, we advise users to avoid the use of an MPI-based NAMD build as it will underperform compared with a native network layer such as InfiniBand IB verbs (often referred to as "verbs"), the UCX (Unified Communication X) framework ("ucx"), the Cray-specific "ofi-crayshasta" (or "ofi-crayshasta cxi") layer for Slingshot-11, or the IBM PAMI message passing layer, as practical examples.

## 16.2 Non-bonded interaction distance-testing

The last critical parameter for non-bonded interaction calculations is the parameter `pairlistdist`. To reduce the cost of performing the non-bonded interactions, NAMD uses a *non-bonded pair list* which contained all pairs of atoms for which non-bonded interactions should be calculated. Performing the search for pairs of atoms that should have their interactions calculated is an expensive operation. Thus, the pair list is only calculated periodically, at least once per cycle. Unfortunately, pairs of atoms move relative to each other during the steps between preparation of the pair list. Because of this, if the pair list were built to include only those pairs of atoms that are within the cutoff distance when the list is generated, it would be possible for atoms to drift closer together than the cutoff distance during subsequent timesteps and yet not have their non-bonded interactions calculated.

Let us consider a concrete example to better understand this. Assume that the pairlist is built once every ten timesteps and that the cutoff distance is 8.0 Å. Consider a pair of atoms A and B that are 8.1 Å apart when the pairlist is built. If the pair list includes only those atoms within the cutoff distance, this pair would not be included in the list. Now assume that after five timesteps, atoms A and B have moved to only 7.9 Å apart. A and B are now within the cutoff distance of each other, and should have their non-bonded interactions calculated. However, because the non-bonded interactions are based solely on the pair list and the pair list will not be rebuilt for another five timesteps, this pair will be ignored for five timesteps causing energy not to be conserved within the system.

To avoid this problem, the parameter `pairlistdist` allows the user to specify a distance greater than the `cutoff` distance for pairs to be included in the pair list, as shown in Figure 19. Pairs that are included in the pair list but are outside the cutoff distance are simply ignored. So in the above example, if the `pairlistdist` were set to 10.0 Å, then the atom pair A and B would be included in the pair list, even though the pair would initially be ignored because they are further apart than the cutoff distance. As the pair moved closer and entered the cutoff distance, because the pair was already in the pair list, the non-bonded interactions would immediately be calculated and energy conservation would be preserved. The value of `pairlistdist` should be chosen such that no atom pair moves more than `pairlistdist` − `cutoff` in one cycle. This will insure energy conservation and efficiency.

The `pairlistdist` parameter is also used to determine the minimum patch size. Unless the `splitPatch` parameter is explicitly set to `position`, hydrogen atoms will be placed on the same patch as the "mother atom" to which they are bonded. These *hydrogen groups* are then distance tested against each other using only a cutoff increased by the the value of the `hgroupCutoff` parameter. The size of the patches is also increased by this amount. NAMD functions correctly even if a hydrogen atom and its mother atom are separated by more than half of `hgroupCutoff`

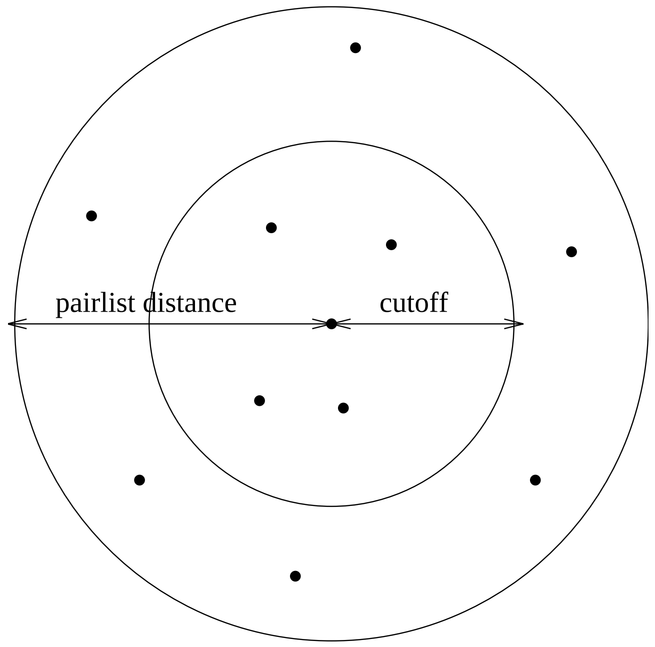


Figure 19: Depiction of the difference between the cutoff distance and the pair list distance. The pair list distance specifies a sphere that is slightly larger than that of the cutoff so that pairs are allowed to move in and out of the cutoff distance without causing energy conservation to be disturbed.

by breaking that group into its individual atoms for distance testing. Margin violation warning messages are printed if an atom moves outside of a safe zone surrounding the patch to which it is assigned, indicating that `pairlistdist` should be increased in order for forces to be calculated correctly and energy to be conserved.

Margin violations mean that atoms that are in non-neighboring patches may be closer than the cutoff distance apart. This may sometimes happen in constant pressure simulations when the cell shrinks (since the patch grid remains the same size). The workaround is to increase the margin parameter so that the simulation starts with fewer, larger patches. Restarting the simulation will also regenerate the patch grid.

In rare special circumstances atoms that are involved in bonded terms (bonds, angles, dihedrals, or impropers) or nonbonded exclusions (especially implicit exclusions due to bonds) will be placed on non-neighboring patches because they are more than the cutoff distance apart. This can result in the simulation dying with a message of "bad global exclusion count". If an "atoms moving too fast; simulation has become unstable", "bad global exclusion count", or similar error happens on the first timestep then there is likely something very wrong with the input coordinates, such as the atoms with uninitialized coordinates or different atom orders in the PSF and PDB file. Looking at the system in VMD will often reveal an abnormal structure. Be aware that the atom IDs in the "Atoms moving too fast" error message are 1-based, while VMD's atom indices are 0-based. If an "atoms moving too fast; simulation has become unstable", "bad global exclusion count", or similar error happens later in the simulation then the dynamics have probably become unstable, resulting in the system "exploding" apart. Energies printed at every timestep should show an exponential increase. This may be due to a timestep that is too long, or some other strange feature. Saving a trajectory of every step and working backwards in can also sometimes reveal the origin of the instability.

- `pairlistdist` < distance between pairs for inclusion in pair lists (Å) >
  **Acceptable Values:** positive decimal $\geq$ `cutoff`
  **Default Value:** `cutoff`
  **Description:** A pair list is generated `pairlistsPerCycle` times each cycle, containing pairs of atoms for which electrostatics and van der Waals interactions will be calculated. This parameter is used when `switching` is set to `on` to specify the allowable distance between atoms for inclusion in the pair list. This parameter is equivalent to the X-PLOR parameter `CUTNb`. If no atom moves more than `pairlistdist`−`cutoff` during one cycle, then there will be no jump in electrostatic or van der Waals energies when the next pair list is built. Since such a jump is unavoidable when truncation is used, this parameter may only be specified when `switching` is set to `on`. If this parameter is not specified and `switching` is set to `on`, the value of `cutoff` is used. A value of at least one greater than `cutoff` is recommended.

- `stepspercycle` < timesteps per cycle >
  **Acceptable Values:** positive integer
  **Default Value:** 20
  **Description:** Number of timesteps in each cycle. Each cycle represents the number of timesteps between the reassignment of atoms to patches, a procedure referred to as *atom migration*. For more details on non-bonded force evaluation, see Section 5.2.

- `splitPatch` < how to assign atoms to patches >
  **Acceptable Values:** `position` or `hydrogen`
  **Default Value:** `hydrogen`
  **Description:** When set to `hydrogen`, hydrogen atoms are kept on the same patch as their parents, allowing faster distance checking and rigid bonds.

- `hgroupCutoff (Å)` < used for group-based distance testing >
  **Acceptable Values:** positive decimal
  **Default Value:** 2.5
  **Description:** This should be set to twice the largest distance which will ever occur between a hydrogen atom and its mother. Warnings will be printed if this is not the case. This value is also added to the margin.

- `margin` < extra length in patch dimension (Å) >
  **Acceptable Values:** positive decimal
  **Default Value:** 0.0
  **Description:** An internal tuning parameter used in determining the size of the cubes of space with which NAMD uses to partition the system. The value of this parameter will not change the physical results of the simulation. Unless you are very motivated to get the *very* best possible performance, just leave this value at the default.

- `pairlistMinProcs` < min procs for pairlists >
  **Acceptable Values:** positive integer
  **Default Value:** 1
  **Description:** Pairlists may consume a large amount of memory as atom counts, densities, and cutoff distances increase. Since this data is distributed across processors it is normally

only problematic for small processor counts. Set pairlistMinProcs to the smallest number of processors on which the simulation can fit into memory when pairlists are used.

- `pairlistsPerCycle` < regenerate x times per cycle >
  **Acceptable Values:** positive integer
  **Default Value:** 2
  **Description:** Rather than only regenerating the pairlist at the beginning of a cycle, regenerate multiple times in order to better balance the costs of atom migration, pairlist generation, and larger pairlists.

- `outputPairlists` < how often to print warnings >
  **Acceptable Values:** non-negative integer
  **Default Value:** 0
  **Description:** If an atom moves further than the pairlist tolerance during a simulation (initially (pairlistdist - cutoff)/2 but refined during the run) any pairlists covering that atom are invalidated and temporary pairlists are used until the next full pairlist regeneration. All interactions are calculated correctly, but efficiency may be degraded. Enabling outputPairlists will summarize these pairlist violation warnings periodically during the run.

- `pairlistShrink` < tol *= (1 - x) on regeneration >
  **Acceptable Values:** non-negative decimal
  **Default Value:** 0.01
  **Description:** In order to maintain validity for the pairlist for an entire cycle, the pairlist tolerance (the distance an atom can move without causing the pairlist to be invalidated) is adjusted during the simulation. Every time pairlists are regenerated the tolerance is reduced by this fraction.

- `pairlistGrow` < tol *= (1 + x) on trigger >
  **Acceptable Values:** non-negative decimal
  **Default Value:** 0.01
  **Description:** In order to maintain validity for the pairlist for an entire cycle, the pairlist tolerance (the distance an atom can move without causing the pairlist to be invalidated) is adjusted during the simulation. Every time an atom exceeds a trigger criterion that is some fraction of the tolerance distance, the tolerance is increased by this fraction.

- `pairlistTrigger` < trigger is atom beyond (1 - x) * tol >
  **Acceptable Values:** non-negative decimal
  **Default Value:** 0.3
  **Description:** The goal of pairlist tolerance adjustment is to make pairlist invalidations rare while keeping the tolerance as small as possible for best performance. Rather than monitoring the (very rare) case where atoms actually move more than the tolerance distance, we reduce the trigger tolerance by this fraction. The tolerance is increased whenever the trigger tolerance is exceeded, as specified by pairlistGrow.

GPU-resident mode changes the default `margin` to 4 Å to improve performance. Since setting this parameter in the config file will override this default, it is important for `margin` to remain *unset* without first running benchmarking experiments to find an optimal value. Although `outputEnergies` will default to 100 steps, this tends to be much too short for all but the largest GPU-resident simulations. It is recommended to set `outputEnergies` to no less than 500 steps, maybe as high as 1000 or more. To monitor performance, `outputTiming` can be set to the same value as `outputEnergies`.

## 17 Translation between NAMD and X-PLOR configuration parameters

NAMD was designed to provide many of the same molecular dynamics functions that X-PLOR provides. As such, there are many similarities between the types of parameters that must be passed to both X-PLOR and NAMD. This section describes relations between similar NAMD and X-PLOR parameters.

- **NAMD Parameter:** `cutoff`
  **X-PLOR Parameter:** `CTOFNB`
  When full electrostatics are not in use within NAMD, these parameters have exactly the same meaning — the distance at which electrostatic and van der Waals forces are truncated. When full electrostatics are in use within NAMD, the meaning is still very similar. The van der Waals force is still truncated at the specified distance, and the electrostatic force is still computed at every timestep for interactions within the specified distance. However, the NAMD integration uses multiple time stepping to compute electrostatic force interactions beyond this distance every `stepspercycle` timesteps.

- **NAMD Parameter:** `vdwswitchdist`
  **X-PLOR Parameter:** `CTONNB`
  Distance at which the van der Waals switching function becomes active.

- **NAMD Parameter:** `pairlistdist`
  **X-PLOR Parameter:** `CUTNb`
  Distance within which interaction pairs will be included in pairlist.

- **NAMD Parameter:** `oneFourScaling`
  **X-PLOR Parameter:** `E14Fac`
  Scaling factor for 1-4 pair electrostatic interactions.

- **NAMD Parameter:** `dielectric`
  **X-PLOR Parameter:** `EPS`
  Dielectric constant.

- **NAMD Parameter:** `exclude`
  **X-PLOR Parameter:** `NBXMod`
  Both parameters specify which atom pairs to exclude from non-bonded interactions. The ability to ignore explicit exclusions is *not* present within NAMD, thus only positive values of `NBXMod` have NAMD equivalents. These equivalences are

  - `NBXMod=1` is equivalent to `exclude=none` — no atom pairs excluded,
  - `NBXMod=2` is equivalent to `exclude=1-2` — only 1-2 pairs excluded,
  - `NBXMod=3` is equivalent to `exclude=1-3` — 1-2 and 1-3 pairs excluded,
  - `NBXMod=4` is equivalent to `exclude=1-4` — 1-2, 1-3, and 1-4 pairs excluded,
  - `NBXMod=5` is equivalent to `exclude=scaled1-4` — 1-2 and 1-3 pairs excluded, 1-4 pairs modified.

- **NAMD Parameter:** `switching`
  **X-PLOR Parameter:** `SHIFt, VSWItch, and TRUNcation`
  Activating the NAMD option `switching` is equivalent to using the X-PLOR options `SHIFt` and `VSWItch`. Deactivating `switching` is equivalent to using the X-PLOR option `TRUNcation`.

- **NAMD Parameter:** `temperature`
  **X-PLOR Parameter:** `FIRSttemp`
  Initial temperature for the system.

- **NAMD Parameter:** `rescaleFreq`
  **X-PLOR Parameter:** `IEQFrq`
  Number of timesteps between velocity rescaling.

- **NAMD Parameter:** `rescaleTemp`
  **X-PLOR Parameter:** `FINAltemp`
  Temperature to which velocities are rescaled.

- **NAMD Parameter:** `restartname`
  **X-PLOR Parameter:** `SAVE`
  Filename prefix for the restart files.

- **NAMD Parameter:** `restartfreq`
  **X-PLOR Parameter:** `ISVFrq`
  Number of timesteps between the generation of restart files.

- **NAMD Parameter:** `DCDfile`
  **X-PLOR Parameter:** `TRAJectory`
  Filename for the position trajectory file.

- **NAMD Parameter:** `DCDfreq`
  **X-PLOR Parameter:** `NSAVC`
  Number of timesteps between writing coordinates to the trajectory file.

- **NAMD Parameter:** `velDCDfile`
  **X-PLOR Parameter:** `VELOcity`
  Filename for the velocity trajectory file.

- **NAMD Parameter:** `velDCDfreq`
  **X-PLOR Parameter:** `NSAVV`
  Number of timesteps between writing velocities to the trajectory file.

- **NAMD Parameter:** `numsteps`
  **X-PLOR Parameter:** `NSTEp`
  Number of simulation timesteps to perform.

# 18 Sample configuration files

This section contains some simple example NAMD configuration files to serve as templates.

This file shows a simple configuration file for alanin. It performs basic dynamics with no output files or special features.

```
# protocol params
numsteps        1000

# initial config
coordinates     alanin.pdb
temperature     300K
seed            12345

# output params
outputname      /tmp/alanin
binaryoutput    no

# integrator params
timestep        1.0

# force field params
structure       alanin.psf
parameters      alanin.params
exclude         scaled1-4
oneFourScaling  1.0
switching       on
switchdist      8.0
cutoff          12.0
pairlistdist    13.5
stepspercycle   20
```

This file is again for alanin, but shows a slightly more complicated configuration. The system is periodic, a coordinate trajectory file and a set of restart files are produced.

```
# protocol params
numsteps        1000

# initial config
coordinates     alanin.pdb
temperature     300K
seed            12345

# periodic cell
cellBasisVector1   33.0 0 0
cellBasisVector2   0 32.0 0
cellBasisVector3   0 0 32.5

# output params
outputname      /tmp/alanin
binaryoutput    no
DCDfreq         10
restartfreq     100

# integrator params
timestep        1.0

# force field params
structure       alanin.psf
parameters      alanin.params
exclude         scaled1-4
oneFourScaling  1.0
switching       on
switchdist      8.0
cutoff          12.0
pairlistdist    13.5
stepspercycle   20
```

This file shows another simple configuration file for alanin, but this time with full electrostatics using PME and multiple timestepping.

```
# protocol params
numsteps        1000

# initial config
coordinates     alanin.pdb
temperature     300K
seed            12345

# periodic cell
cellBasisVector1   33.0 0 0
cellBasisVector2   0 32.0 0
cellBasisVector3   0 0 32.5

# output params
outputname      /tmp/alanin
binaryoutput    no
DCDfreq         10
restartfreq     100

# integrator params
timestep        1.0
fullElectFrequency  4

# force field params
structure       alanin.psf
parameters      alanin.params
exclude         scaled1-4
oneFourScaling  1.0
switching       on
switchdist      8.0
cutoff          12.0
pairlistdist    13.5
stepspercycle   20

# full electrostatics
PME             on
PMEGridSizeX    32
PMEGridSizeY    32
PMEGridSizeZ    32
```

This file demonstrates the analysis of a DCD trajectory file using NAMD. The file `pair.pdb` contains the definition of pair interaction groups; NAMD will compute the interaction energy and force between these groups for each frame in the DCD file. It is assumed that coordinate frames were written every 1000 timesteps. See Sec. 15.1 for more about pair interaction calculations.

```
# initial config
coordinates     alanin.pdb
temperature     0

# output params
outputname      /tmp/alanin-analyze
binaryoutput    no

# integrator params
timestep        1.0

# force field params
structure       alanin.psf
parameters      alanin.params
exclude         scaled1-4
oneFourScaling  1.0
switching       on
switchdist      8.0
cutoff          12.0
pairlistdist    13.5
stepspercycle   20

# Atoms in group 1 have a 1 in the B column; group 2 has a 2.
pairInteraction on
pairInteractionFile pair.pdb
pairInteractionCol B
pairInteractionGroup1 1
pairInteractionGroup2 2

# First frame saved was frame 1000.
set ts 1000

coorfile open dcd /tmp/alanin.dcd

# Read all frames until nonzero is returned.
while { ![coorfile read] } {
  # Set firstTimestep so our energy output has the correct TS.
  firstTimestep $ts
  # Compute energies and forces, but don't try to move the atoms.
  run 0
  incr ts 1000
}
coorfile close
```

# 19 Running NAMD

NAMD runs on a variety of serial and parallel platforms. While it is trivial to launch a serial program, a parallel program depends on a platform-specific library such as MPI to launch copies of itself on other nodes and to provide access to a high performance network such as InfiniBand if one is available.

For typical workstations (Windows, Linux, Mac OS X, or other Unix) with only ethernet networking (hopefully gigabit), NAMD uses the Charm++ native communications layer and the program charmrun to launch namd3 processes for parallel runs (either exclusively on the local machine with the ++local option or on other hosts as specified by a nodelist file). The namd3 binaries for these platforms can also be run directly (known as standalone mode) for single process runs.

## 19.1 Individual Windows, Linux, Mac OS X, or Other Unix Workstations

Individual workstations use the same version of NAMD as workstation networks, but running NAMD is much easier. If your machine has only one processor core you can run any netlrts or multicore namd3 binary directly:

```
namd3 <configfile>
```

Mac OX X (Intel) and Linux-x86_64-multicore released binaries are based on "multicore" builds of Charm++ that can run multiple threads. These multicore builds lack a network layer, so they can only be used on a single machine. For best performance use one thread per processor with the +p option:

```
namd3 +p<procs> <configfile>
```

Windows native builds are no longer provided for download. Instead, the Windows platform is now supported by running Linux-x86_64-multicore builds using WSL2 on Windows 10 or 11. Using the Linux builds through WSL2 generally offers higher performance than running native builds.

For other multiprocessor workstations the included charmrun program is needed to run multiple namd3 processes. The ++local option is also required to specify that only the local machine is being used:

```
charmrun namd3 ++local +p<procs> <configfile>
```

You may need to specify the full path to the namd3 binary.

## 19.2 Linux Clusters with InfiniBand or Other High-Performance Networks

Charm++ provides a special verbs network layer that uses InfiniBand networks directly through the OpenFabrics OFED ibverbs library. This avoids efficiency and portability issues associated with MPI. This newer Charm++ verbs network layer replaces the earlier ibverbs layer, providing equivalent performance while also supporting multi-copy algorithms (replicas).

Intel Omni-Path networks are incompatible with the pre-built verbs NAMD binaries. Charm++ for verbs can be built with –with-qlogic to support Omni-Path, but for this case the Charm++ MPI network layer performs better than the verbs layer. Hangs have been observed with Intel MPI but not with OpenMPI, so OpenMPI is preferred. See "Compiling NAMD" below for MPI build instructions. NAMD MPI binaries may be launched directly with mpiexec rather than via the provided charmrun script.

Writing batch job scripts to run charmrun in a queueing system can be challenging. Since most clusters provide directions for using mpiexec to launch MPI jobs, charmrun provides a ++mpiexec option to use mpiexec to launch non-MPI binaries. If "mpiexec -n *procs* ..." is not sufficient to launch jobs on your cluster you will need to write an executable mympiexec script like the following from TACC:

```
#!/bin/csh
shift; shift; exec ibrun $*
```

The job is then launched (with full paths where needed) as:

```
charmrun +p<procs> ++mpiexec ++remote-shell mympiexec namd3 <configfile>
```

Charm++ now provides the option ++mpiexec-no-n for the common case where mpiexec does not accept "-n *procs*" and instead derives the number of processes to launch directly from the queueing system:

```
charmrun +p<procs> ++mpiexec-no-n ++remote-shell ibrun namd3 <configfile>
```

For workstation clusters and other massively parallel machines with special high-performance networking, NAMD uses the system-provided MPI library (with a few exceptions) and standard system tools such as mpirun are used to launch jobs. Since MPI libraries are very often incompatible between versions, you will likely need to recompile NAMD and its underlying Charm++ libraries to use these machines in parallel (the provided non-MPI binaries should still work for serial runs.) The provided charmrun program for these platforms is only a script that attempts to translate charmrun options into mpirun options, but due to the diversity of MPI libraries it often fails to work.

## 19.3 Linux or Other Unix Workstation Networks

The same binaries used for individual workstations as described above (other than pure "multicore" builds and MPI builds) can be used with charmrun to run in parallel on a workstation network. The only difference is that you must provide a "nodelist" file listing the machines where namd3 processes should run, for example:

```
group main
host brutus
host romeo
```

The "group main" line defines the default machine list. Hosts brutus and romeo are the two machines on which to run the simulation. Note that charmrun may run on one of those machines, or charmrun may run on a third machine. All machines used for a simulation must be of the same type and have access to the same namd3 binary.

By default, the "rsh" command is used to start namd3 on each node specified in the nodelist file. You can change this via the CONV_RSH environment variable, i.e., to use ssh instead of rsh run "setenv CONV_RSH ssh" or add it to your login or batch script. You must be able to connect to each node via rsh/ssh without typing your password; this can be accomplished via a .rhosts files in your home directory, by an /etc/hosts.equiv file installed by your sysadmin, or by a .ssh/authorized_keys file in your home directory. You should confirm that you can run "ssh hostname pwd" (or "rsh hostname pwd") without typing a password before running NAMD. Contact your local sysadmin if you have difficulty setting this up. If you are unable to use rsh or ssh, then add "setenv CONV_DAEMON" to your script and run charmd (or charmd_faceless, which produces a log file) on every node.

You should now be able to try running NAMD as:

```
charmrun namd3 +p<procs> <configfile>
```

If this fails or just hangs, try adding the ++verbose option to see more details of the startup process. You may need to specify the full path to the namd3 binary. Charmrun will start the number of processes specified by the +p option, cycling through the hosts in the nodelist file as many times as necessary. You may list multiprocessor machines multiple times in the nodelist file, once for each processor.

You may specify the nodelist file with the "++nodelist" option and the group (which defaults to "main") with the "++nodegroup" option. If you do not use "++nodelist" charmrun will first look for "nodelist" in your current directory and then ".nodelist" in your home directory.

Some automounters use a temporary mount directory which is prepended to the path returned by the pwd command. To run on multiple machines you must add a "++pathfix" option to your nodelist file. For example:

```
group main ++pathfix /tmp\_mnt /
host alpha1
host alpha2
```

There are many other options to charmrun and for the nodelist file. These are documented at in the Charm++ Installation and Usage Manual available at http://charm.cs.uiuc.edu/manuals/ and a list of available charmrun options is available by running charmrun without arguments.

If your workstation cluster is controlled by a queueing system you will need build a nodelist file in your job script. For example, if your queueing system provides a HOST_FILE environment variable:

```
set NODES = `cat $HOST_FILE`
set NODELIST = $TMPDIR/namd3.nodelist
echo group main >! $NODELIST
foreach node ( $NODES )
  echo host $node >> $NODELIST
end
@ NUMPROCS = 2 * $#NODES
charmrun namd3 +p$NUMPROCS ++nodelist $NODELIST <configfile>
```

Note that NUMPROCS is twice the number of nodes in this example. This is the case for dual-processor machines. For single-processor machines you would not multiply $#NODES by two.

Note that these example scripts and the setenv command are for the csh or tcsh shells. They must be translated to work with sh or bash.

## 19.4 Shared-Memory and Network-Based Parallelism (SMP Builds)

The Linux-x86_64-verbs-smp and Linux-x86_64-netlrts-smp released binaries are based on "smp" builds of Charm++ that can be used with multiple threads on either a single machine like a multicore build, or across a network. SMP builds combine multiple worker threads and an extra communication thread into a single process. Since one core per process is used for the communication thread SMP builds might be slower than non-SMP builds. The advantage of SMP builds is that many data structures are shared among the threads, reducing the per-core memory footprint when scaling large simulations to large numbers of cores.

SMP builds launched with charmrun use ++n to specify the total number of processes (Charm++ "nodes") and ++ppn to specify the number of PEs (Charm++ worker threads) per process. Prevous versions required the use of +p to specify the total number of PEs, but the new ++n option is now recommended. Thus, to run one process with one communication and three worker threads on each of four quad-core nodes one would specify:

```
charmrun namd3 ++n 4 ++ppn 3 <configfile>
```

For MPI-based SMP builds one would specify any mpiexec options needed for the required number of processes and pass +ppn to the NAMD binary as:

```
mpiexec -n 4 namd3 +ppn 3 <configfile>
```

See the Cray XE/XK/XC directions below for a more complex example.

## 19.5 Cray XE/XK/XC

First load modules for the GNU compilers (XE/XK only, XC should use Intel), topology information, huge page sizes, and the system FFTW 3 library:

```
module swap PrgEnv-cray PrgEnv-gnu
module load rca
module load craype-hugepages8M
module load fftw
```

The CUDA Toolkit module enables dynamic linking, so it should only be loaded when building CUDA binaries and never for non-CUDA binaries:

```
module load cudatoolkit
```

For CUDA or large simulations on XE/XK use gemini_gni-crayxe-persistent-smp and for smaller XE simulations use gemini_gni-crayxe-persistent. For XC similarly use gni-crayxc-persistent-smp or gni-crayxc-persistent.

For XE/XK use CRAY-XE-gnu and (for CUDA) the "–with-cuda" config option, the appropriate "–charm-arch" parameter, and –with-fftw3. For XC use instead CRAY-XC-intel but all other options the same.

Your batch job will need to load modules and set environment variables:

```
module swap PrgEnv-cray PrgEnv-gnu
module load rca
module load craype-hugepages8M
setenv HUGETLB_DEFAULT_PAGE_SIZE 8M
setenv HUGETLB_MORECORE no
```

To run an SMP build with one process per node on 16 32-core nodes:

```
aprun -n 16 -r 1 -N 1 -d 31 /path/to/namd3 +ppn 30 +pemap 1-30 +commap 0 <configfile>
```

or the same with 4 processes per node:

```
aprun -n 64 -N 4 -d 8 /path/to/namd3 +ppn 7 \
          +pemap 1-7,9-15,17-23,25-31 +commap 0,8,16,24 <configfile>
```

or non-SMP, leaving one core free for the operating system:

```
aprun -n 496 -r 1 -N 31 -d 1 /path/to/namd3 +pemap 0-30 <configfile>
```

The explicit +pemap and +commap settings are necessary to avoid having multiple threads assigned to the same core (or potentially all threads assigned to the same core). If the performance of NAMD running on a single compute node is much worse than comparable non-Cray host then it is very likely that your CPU affinity settings need to be fixed.

All Cray XE/XK/XC network layers support multi-copy algorithms (replicas).

## 19.6 Xeon Phi Processors (KNL)

Special Linux-KNL-icc and CRAY-XC-KNL-intel builds enable vectorizable mixed-precision kernels while preserving full alchemical and other functionality. Multi-host runs require multiple smp processes per host (as many as 13 for Intel Omni-Path, 6 for Cray Aries) in order to drive the network. Careful attention to CPU affinity settings (see below) is required, as is 1 or 2 (but not 3 or 4) hyperthreads per PE core (but only 1 per communication thread core).

There appears to be a bug in the Intel 17.0 compiler that breaks the non-KNL-optimized NAMD kernels (used for alchemical free energy, etc.) on KNL. Therefore the Intel 16.0 compilers are recommended on KNL.

## 19.7 SGI Altix UV

Use Linux-x86_64-multicore and the following script to set CPU affinity:

```
namd3 +setcpuaffinity ‘numactl --show | awk ’/^physcpubind/ {printf \
   "+p%d +pemap %d",(NF-1),$2; for(i=3;i<=NF;++i){printf ",%d",$i}}’‘ ...
```

## 19.8 IBM POWER Clusters

Run the verbs or ibverbs version of NAMD as on any other cluster, using poe in place of mpiexec as the process launcher, for example:

```
charmrun +p<procs> ++mpiexec-no-n ++remote-shell poe namd3 <configfile>
```

The details of job submission will vary between sites. For example, two nodes with two tasks per node on LSF are -n 4 -R ”span[ptile=2]” with charmrun options +p36 ++ppn 9 ++mpiexec-no-n ++remote-shell poe

## 19.9 CPU Affinity

NAMD may run faster on some machines if threads or processes are set to run on (or not run on) specific processor cores (or hardware threads). On Linux this can be done at the process level with the numactl utility, but NAMD provides its own options for assigning threads to cores. This feature is enabled by adding +setcpuaffinity to the namd3 command line, which by itself will cause NAMD (really the underlying Charm++ library) to assign threads/processes round-robin to available cores in the order they are numbered by the operating system. This may not be the fastest configuration if NAMD is running fewer threads than there are cores available and consecutively numbered cores share resources such as memory bandwidth or are hardware threads on the same physical core.

If needed, specific cores for the Charm++ PEs (processing elements) and communication threads (on SMP builds) can be set by adding the +pemap and (if needed) +commap options with lists of core sets in the form "lower[-upper[:stride[.run]]][,...]". A single number identifies a particular core. Two numbers separated by a dash identify an inclusive range (lower bound and upper bound). If they are followed by a colon and another number (a stride), that range will be stepped through in increments of the additional number. Within each stride, a dot followed by a run will indicate how many cores to use from that starting point. For example, the sequence 0-8:2,16,20-24 includes cores 0, 2, 4, 6, 8, 16, 20, 21, 22, 23, 24. On a 4-way quad-core system three cores from each socket would be 0-15:4.3 if cores on the same chip are numbered consecutively. There is no need to repeat cores for each node in a run as they are reused in order.

For example, the IBM POWER7 has four hardware threads per core and the first thread can use all of the core's resources if the other threads are idle; threads 0 and 1 split the core if threads 2 and 3 are idle, but if either of threads 2 or 3 are active the core is split four ways. The fastest configuration of 32 threads or processes on a 128-thread 32-core is therefore "+setcpuaffinity +pemap 0-127:4". For 64 threads we need cores 0,1,4,5,8,9,... or 0-127:4.2. Running 4 processes with +ppn 31 would be "+setcpuaffinity +pemap 0-127:32.31 +commap 31-127:32"

For Intel processors, including KNL, where hyperthreads on the same core are not numbered consecutively, hyperthreads may be mapped to consecutive PEs by appending [+span] to a core set, e.g., "+pemap 0-63+64+128+192" to use all threads on a 64-core, 256-thread KNL with threads mapped to PEs as 0,64,128,192,1,65,129,193,...

For an Altix UV or other machines where the queueing system assigns cores to jobs this information must be obtained with numactl –show and passed to NAMD in order to set thread affinity (which will improve performance):

```
namd3 +setcpuaffinity `numactl --show | awk '/^physcpubind/ {printf \
   "+p%d +pemap %d",(NF-1),$2; for(i=3;i<=NF;++i){printf ",%d",$i}}'` ...
```

## 19.10 GPU Acceleration

NAMD supports GPU-accelerated calculations for NVIDIA GPUs using CUDA and AMD GPUs using HIP/ROCm. The "classic" mode of running NAMD with GPU acceleration offloads most of the force calculations to GPU devices (GPU-offload mode) and runs the other calculations (integration, rigid bond constraints, etc.) on the CPU. Version 3.0 of NAMD introduces a GPU-resident mode which calculates the entire dynamics calculations on GPU.

A port of NAMD's CUDA kernels to SYCL/oneAPI is in development to support Intel GPUs (e.g., ALCF Aurora). A source code preview release (version 2.15 alpha 2) providing SYCL support for GPU-offload mode is available on the NAMD website.

### 19.10.1 GPU-Offload Mode

For GPU-offload mode, NAMD does not offload the entire calculation to the GPU, and performance may therefore be limited by the CPU. In general all available CPU cores should be used, with CPU affinity set as described above.

Energy evaluation is slower than calculating forces alone, and the loss is much greater in CUDA-accelerated builds. Therefore you should set outputEnergies to 100 or higher in the simulation config file. Forces evaluated on the GPU differ slightly from a CPU-only calculation, an effect more visible in reported scalar pressure values than in energies.

NAMD now has the entire force calculation offloaded to GPU for conventional MD simulation options. However, not all advanced features are compatible with CUDA-accelerated NAMD builds, in particular, any simulation option that requires modification to the functional form of the nonbonded forces. Note that QM/MM simulation is also disabled for CUDA-accelerated NAMD, because the calculation is bottlenecked by the QM calculation rather than the MM force calculation, so can benefit from CUDA acceleration of the QM part when available. Table 1 lists the parts of NAMD that are accelerated with CUDA-capable GPUs, and Table 2 lists the advanced simulation options that are disabled within a CUDA-accelerated NAMD build.

Table 1: NAMD GPU-offload mode: What is accelerated?

| **Accelerated** | **Not Accelerated** |
|---|---|
| short-range nonbonded | integration |
| PME reciprocal sum | rigid bonds |
| bonded terms | grid forces |
| implicit solvent | collective variables |
| alchemical (FEP and TI) | |

Table 2: NAMD GPU: What features are disabled?

| **Disabled** | **Not Disabled** |
|---|---|
| Locally enhanced sampling | Memory optimized builds |
| Tabulated energies | Conformational free energy |
| Drude (nonbonded Thole) | Collective variables |
| Go forces | Grid forces |
| Pairwise interaction | Steering forces |
| Pressure profile | Almost everything else |
| QM/MM | |

### 19.10.2 GPU-Resident Mode

GPU-resident mode for MD simulation is a feature new to version 3.0. Unlike GPU-offload mode that offloads the force calculation to GPU devices while performing integration and rigid bond constraints on the host CPU, GPU-resident mode also performs integration and rigid bond constraints on the GPU device and, most importantly, maintains the simulation data on the GPU device between time steps. By removing the performance bottleneck resulting from host–device memory transfers and CPU kernel calculations performed at every time step, MD simulation performance on modern GPU hardware is more than doubled.

The new GPU-resident mode is also capable of scaling a simulation across multiple GPUs within a single node as long as the GPUs have peer access to directly read and write each others' local memories, typically configured when all devices are mutually connected by a high-speed networking fabric such as NVLink/NVSwitch for NVIDIA GPUs or Infinity Fabric for AMD GPUs. Representative configurations are found in desktop workstations with an NVLinked pair of NVIDIA GPUs, NVIDIA DGX computers, and the nodes of many supercomputers, e.g., OLCF Frontier (AMD), OLCF Summit (NVIDIA), ALCF Polaris (NVIDIA), and NERSC Perlmutter (NVIDIA).

GPU-resident mode for now exploits shared-memory parallelism, which means that it is limited to multicore and netlrts-smp builds, where the latter supports multiple replica GPU-resident simulations with each replica running as a single process.

Unlike GPU-offload mode which generally benefits from a larger number of CPU cores, GPU-resident mode offers limited host-side work requiring fewer CPU cores. In fact, attempting to run too many CPU cores with GPU-resident mode can slow down performance, due to the increased overhead of synchronizing those cores without enough work to reasonably utilize them. For GPU-resident mode, the number of CPU cores (PEs) to use depends on the size of the system, the features being used, the number of GPU devices used, and the relative performance of the CPU cores compared to the GPU devices. It is recommended to run benchmarks to determine the optimal core count for your hardware. The command `benchmarkTime` can be used with `outputTiming` to easily benchmark a system from the command line without needing to modify the config file. For example,

```
./namd3 +p4 +setcpuaffinity --outputTiming 500 --benchmarkTime 180 <configfile>
```

will terminate the simulation after running for three minutes (180 seconds), detected at the next output from `outputTiming`.

Since GPU-resident mode performs all calculation on the GPU device, advanced features must generally be supported by porting a given feature to the GPU. Several features have been ported to GPU-resident mode but many others still need to be ported. Some of these advanced features are available for multi-GPU scaling and others are single-GPU only. There are also some features that are now provided as GPU-resident-only high-performance alternatives to already existing GPU-offload features. Table 3 lists the features available to GPU-resident mode, indicating support for multi-GPU scaling and what methodologies are replaced, if any.

Essential dynamics includes the standard ensembles (constant energy, constant temperature with Langevin damping or stochastic velocity rescaling, and constant pressure and temperature with Langevin piston) together with rigid bond constraints, multiple time stepping, and PME electrostatics. Note that fixed atoms are not yet supported by GPU-resident, with harmonic restraints recommended as a workaround until support for fixed atoms is finished. The Monte Carlo barostat offers a faster pressure control method than Langevin piston, by avoiding calculation of the pressure virial at every step. Group position restraints is a NAMD-native GPU-resident implementation of Colvars distance restraints, providing much higher performance than Colvars. Both Colvars and TCL forces can be used with GPU-resident mode, but their use might significantly impact performance since either one requires host–device data transfer and CPU host calculations every step. The impact to performance depends on what collective variables have been defined and the number of atoms affected.

Table 3: NAMD GPU-resident mode: What features are supported?

| **Feature** | **Multi-GPU** | **Replacing** |
|---|---|---|
| Essential dynamics | yes | |
| 4-site water models (TIP4P and OPC) | yes | |
| Alchemical (FEP and TI) | yes | |
| Multi-replica | yes | |
| Replica exchange solute scaling (REST2) | yes | |
| Harmonic restraints | no | Fixed atoms |
| External electric field | no | |
| Monte Carlo barostat | no | Langevin piston |
| Group position restraints | no | Colvars distance restraints |
| Colvars | yes | |
| TCL forces | yes | |

Whether or not to use multi-GPU scaling for a simulation depends on the size of the system and the capabilities of the GPU. For example, the 1M-atom STMV benchmark system gets reasonably good scaling efficiency across an 8-GPU NVIDIA DGX-A100. A reasonable rule of thumb seems to be around 100k atoms per GPU for the Ampere series of GPUs and twice that per GPU for Hopper.

Multi-GPU scaling performance for GPU-resident mode is significantly impacted by PME. The issue is that due to the difficulty of scaling the 3D FFT calculations, the long-range (gridded) part of PME is delegated to a single GPU, and NAMD's default work decomposition scheme, to evenly distribute patches to CPU cores and evenly distribute CPU cores to devices will naturally overload the PME device. The workaround is to use task-based parallelism to restrict the amount of "standard" work to the PME device. The approach exploits the existing load balancing performed by NAMD during its startup by simply reducing the number of PEs assigned to the PME device through the new "+pmepes" command line argument. Note that good load balancing should maintain the same number of PEs on the non-PME devices, which means that the overall number of PEs set by "+p" will necessarily be reduced. Setting this argument is best determined by benchmarking the given system on the intended hardware platform, which was done to determine optimal settings for the 1M-atom STMV benchmark system running on DGX-A100, using 8 PEs per device:

```
./namd3 +p8 +setcpuaffinity +devices 0 stmv.namd
./namd3 +p15 +pmepes 7 +setcpuaffinity +devices 0,1 stmv.namd
./namd3 +p29 +pmepes 5 +setcpuaffinity +devices 0,1,2,3 stmv.namd
./namd3 +p57 +pmepes 1 +setcpuaffinity +devices 0,1,2,3,4,5,6,7 stmv.namd
```

Since performance for MD exhibits predominantly linear scaling (up to reasonable size and resource utilization limits), the ratios shown above for STMV can be applied as a starting rule-of-thumb for other systems.

GPU-resident mode can also provide very fast simulation for smaller systems. For example, the AMBER DHFR (23.6k atoms) benchmark, using AMBER force field parameters with 9 Å cutoff, PME, rigid bond constraints, and hydrogen mass repartitioning with 4 fs time step, can be simulated on A100 with over 1 microsecond/day performance. When simulating smaller systems like DHFR, performance is improved by using `twoAwayZ on` to double the patch count, producing a greater number of work units to schedule across the GPU processing units.

Small systems will not achieve good scaling across multiple GPUs. Instead, the most effective way to use multi-GPU architectures is to simulate multi-copy ensembles. Depending on the size of the system and the hardware capability, GPU resources are often most efficiently used by running multiple simulations per GPU, in which performance can be measured as the aggregate number of simulated nanoseconds per day achieved. NVIDIA provides technologies MPS (Multi-Process Service) and MIG (Multi-Instance GPU) that can facilitate running multiple simultaneous NAMD jobs on a single GPU.

### 19.10.3 GPU Hardware Requirements

To benefit from GPU acceleration using NVIDIA GPU hardware you will need a CUDA build of NAMD and a recent NVIDIA video card. CUDA builds will not function without a CUDA-capable GPU and a driver that supports CUDA 9.1. If the installed driver is too old NAMD will exit on startup with the error "CUDA driver version is insufficient for CUDA runtime version." GPUs of compute capability $< 5.0$ are no longer supported and are ignored. GPUs with two or fewer SMs are ignored unless specifically requested with +devices.

Finally, if NAMD was not statically linked against the CUDA runtime then the libcudart.so file included with the binary (copied from the version of CUDA it was built with) must be in a directory in your LD_LIBRARY_PATH before any other libcudart.so libraries. For example, when running a multicore binary (recommended for a single machine):

```
setenv LD_LIBRARY_PATH ".:$LD_LIBRARY_PATH"
(or LD_LIBRARY_PATH=".:$LD_LIBRARY_PATH"; export LD_LIBRARY_PATH)
./namd3 +p8 +setcpuaffinity <configfile>
```

NAMD can be built to run with GPU acceleration on HIP-compatible AMD GPUs. Build instructions can be found in the NAMD distribution notes.txt file. For HIP builds, NAMD has been tested with ROCm 5.4.2 and 5.7.0, and the HIP builds maintain feature parity with CUDA builds.

Each NAMD thread can use only one GPU. Therefore you will need to run at least one thread for each GPU you want to use. Multiple threads in an SMP build of NAMD can share a single GPU, usually with an increase in performance. NAMD will automatically distribute threads equally among the GPUs on a node. Specific GPU device IDs can be requested via the +devices argument on the namd3 command line, for example:

```
./namd3 +p8 +setcpuaffinity +devices 0,2 <configfile>
```

Devices are shared by consecutive threads in a process, so in the above example threads 0–3 will share device 0 and threads 4–7 will share device 2. Repeating a device will cause it to be assigned to multiple master threads, which is allowed only for different processes and is advised against in general but may be faster in certain cases. When running on multiple nodes the +devices specification is applied to each physical node separately and there is no way to provide a unique list for each node.

When running a multi-node parallel job it is recommended to have one process per device to maximize the number of communication threads. If the job launch system enforces device segregation such that not all devices are visible to each process then the +ignoresharing argument must be used to disable the shared-device error message.

When running a multi-copy simulation with both multiple replicas and multiple devices per physical node, the +devicesperreplica <$n$> argument must be used to prevent each replica from binding all of the devices. For example, for 2 replicas per 6-device host use +devicesperreplica 3.

While charmrun with ++local will preserve LD_LIBRARY_PATH, normal charmrun does not. You can use charmrun ++runscript to add the namd3 directory to LD_LIBRARY_PATH with the following executable runscript:

```
#!/bin/csh
setenv LD_LIBRARY_PATH "${1:h}:$LD_LIBRARY_PATH"
$*
```

For example:

```
./charmrun ++runscript ./runscript ++n 4 ./namd3 ++ppn 15 <configfile>
```

An InfiniBand network is highly recommended when running GPU-accelerated NAMD across multiple nodes. You will need either a verbs NAMD binary (available for download) or an MPI NAMD binary (must build Charm++ and NAMD as described above) to make use of the InfiniBand network. The use of SMP binaries is also recommended when running on multiple nodes, with one process per GPU and as many threads as available cores, reserving one core per process for the communication thread.

The CUDA (NVIDIA's graphics processor programming platform) code in NAMD is completely self-contained and does not use any of the CUDA support features in Charm++. When building NAMD with CUDA support you should use the same Charm++ you would use for a non-CUDA build. Do NOT add the cuda option to the Charm++ build command line. The only changes to the build process needed are to add –with-cuda for GPU-offload support or –with-single-node-cuda for GPU-resident support and possibly –cuda-prefix ... to the NAMD config command line.

NAMD can also be built with HIP/ROCm to support compatible AMD GPUs, otherwise matching all features available with CUDA builds. The build configuration options for HIP/ROCm are changed to –with-hip for GPU-offload support or –with-single-node-hip for GPU-resident support and –rocm-prefix ... to specify the library path.

For now, NAMD does not support all available features on GPU. Some keywords have been introduced to give the user better control over GPU calculation. These keywords are relevant only for GPU builds and are ignored if the user is running a CPU-only build.

#### 19.10.4 Keywords

- `GPUresident` < Run dynamics calculations entirely on GPU >
  **Acceptable Values:** "on" or "off"
  **Default Value:** off
  **Description:** GPU-resident enabled builds also require setting the GPUresident keyword in order to run dynamics calculations entirely on GPU. Without setting this keyword, GPU-enabled builds of NAMD will run in GPU-offload mode by default. Replaces the deprecated `CUDASOAintegrate` keyword.

- `GPUAtomMigration` < Perform atom migration on the device >
  **Acceptable Values:** "on" or "off"
  **Default Value:** off
  **Description:** An experimental optimization for GPU-resident mode simulation that performs atom migration on the GPU device rather than the CPU host. Using this can give faster performance, especially for smaller systems, but this feature is still considered experimental. For smaller systems, performance can be further improved by also setting "twoAwayZ on" to double the number of patches, improving GPU utilization by increasing the work units to be scheduled across the device. With `GPUAtomMigration` enabled, the maximum patch size is limited to 2048 atoms; exceeding this limit will cause NAMD to terminate with an error message. If patch sizes are close to this limit, then restarting the simulation with "twoAwayZ on" will reduce the sizes by around one-half. Replaces the deprecated `DeviceMigration` keyword.

- `GPUForceTable` < Always use force table interpolation for nonbonded GPU kernel >
  **Acceptable Values:** "on" or "off"
  **Default Value:** on
  **Description:** An experimental optimization for GPU-resident mode simulation. Setting "GPUForceTable off" will use direct math calculation for non-PME steps (i.e., for multiple time stepping when the error function is *not* being computed). For recent GPUs direct calculation is faster for certain cases than force table lookup and interpolation. Not all nonbonded kernel variants are supported, for example, the standard van der Waals switching function is implemented, but the "vdwForceSwitching" function is not. Similarly, alchemical free energy calculations with FEP and TI are not supported. Replaces the deprecated `CUDAForceTable` keyword.

- `bondedGPU` < 0 to 511 >
  **Acceptable Values:** Integer value between 0 and 511
  **Default Value:** 511
  **Description:** NAMD provides GPU kernels for calculating different bonded force terms. This parameter is irrelevant to GPU-resident mode, since all forces are calculated on the GPU. For GPU-offload mode, the bondedGPU parameter acts as a bitmask indicating whether a particular term is enabled for GPU calculation (bit set) or instead calculated on the CPU (bit unset). Any partial sum of the following values can be used to enable only the specified bonded terms for GPU calculation:

  – bond = 1
  – angle = 2
  – dihedral = 4
  – improper = 8
  – exclusion = 16
  – crossterm = 32
  – screened Coulomb correction of Thole (Drude) = 64
  – anisotropic polarization (Drude) = 128
  – screened Coulomb correction of Thole for 1–4 interactions (Drude) = 256

We provide further clarification for the Drude polarizable force field terms. The screened Coulomb correction of Thole is the dipole-dipole interaction between 1–2 and 1–3 terms that are normally excluded from non-polarizable force fields. These terms exclude charge-dipole and charge-charge interactions. The anisotropic polarization is a spring term that accounts for out-of-plane forces from a polarized atom and its covalently bonded neighbor. The screened Coulomb correction of Thole for 1–4 interactions is a correction term that includes the charge-dipole and charge-charge interactions that are excluded from the nonbonded kernel.

The value of `bondedGPU` as a bitmask is applied additively, as the sum of the terms to be calculated on the GPU. For example, if you want to compute the bond and dihedral terms on CPU while keeping the other bonded terms on GPU in the GPU-offload mode, you can set "`bondedGPU 506`" (where 506 is derived from $511 - 1 - 4$) in your NAMD configuration file. This option replaces the deprecated `bondedCUDA` keyword.

- `usePMEGPU` < Offload entire PME reciprocal sum to GPU? >
  **Acceptable Values:** "on" or "off"
  **Default Value:** on
  **Description:** For GPU-resident mode, the entire PME reciprocal sum is automatically run on a single GPU, so it is not necessary to set this parameter. For GPU-offload mode, whether or not usePMEGPU is enabled depends on how NAMD is run. It is automatically enabled when running on a single node and is disabled when running across multiple nodes. Replaces the deprecated `usePMECUDA` keyword.

- `PMEoffload` < Offload PME gridding/ungridding procedures to GPU? >
  **Acceptable Values:** "on" or "off"
  **Default Value:** off
  **Description:** This parameter is irrelevant for GPU-resident mode. For GPU-offload mode, the gridding and ungridding procedures for calculating the PME reciprocal sum is offloaded to GPUs, with the FFT calculation still performed by CPUs. PMEoffload is enabled by default only for PMEinterpOrder $> 4$.

  For huge systems (10 million atoms and above) where the parallel FFT limits performance, it is desirable to use PMEoffload in conjunction with increased order interpolation and increased grid spacing, in order to decrease the overall communication latency by decreasing the overall grid size by a factor of 8 while maintaining the same accuracy for the calculation.
  **Exemplary use:**
```
PME on
PMEinterpOrder 8
PMEgridSpacing 2.0
PMEoffload on ;# enabled by default for these PME settings
```

## 19.11 Xeon and Zen4 Acceleration

NAMD supports CPU-only optimizations for processors that implement particular AVX-512 vector instruction sets (avx512f and avx512dq). These instruction sets have been available for several years with Intel Xeon and more recently with AMD Zen4 processors. A vectorized kernel replaces the standard nonbonded force kernel, reimplementing the CUDA tiles algorithm for AVX-512 using the *Intrinsics for Intel AVX-512 Instructions*. Use of these optimizations is not compatible with GPU-accelerated builds.

The downloads page offers a multicore AVX-512 binary built using GCC 11.2.1 that is compatible with either Xeon or Zen4. Otherwise, build target `Linux-AVX512-icx` (using a recent Intel oneAPI toolkit) is recommended for Xeon and build target `Linux-AVX512-g++` (using GCC version 11 or later) is recommended for Zen4. The AVX-512 tiles optimization is enabled by default. Since use of this optimized kernel is not compatible with all features, NAMD will automatically disable its use when an incompatibility is detected, falling back to the standard CPU code path. There is also a config file parameter to explicitly disable use of the tiles optimization.

- `useAVXTiles` < Use the AVX-512 tiles optimization >
  **Acceptable Values:** "on" or "off"
  **Default Value:** on
  **Description:** This parameter affects only AVX-512 builds. The AVX-512 tiles optimization for calculating nonbonded forces is enabled by default but can be disabled by setting this parameter "off."

## 19.12 Xeon Phi Acceleration

NAMD supports offloading calculations to Intel Xeon Phi coprocessors. This feature is new and should be considered experimental. Observed speedups are around a factor of two, but parallel scaling is degraded.

The Xeon Phi coprocessor is supported in NAMD similar to CUDA GPUs. Binaries are not provided, so you will need to build from source code (see "Compiling NAMD" below) specifying –with-mic to the config script. As with CUDA, multicore or ibverbs-smp builds are strongly recommended. A recent Intel compiler is obviously required to compile for Xeon Phi.

## 19.13 Memory Usage

NAMD has traditionally used less than 100MB of memory even for systems of 100,000 atoms. With the reintroduction of pairlists in NAMD 2.5, however, memory usage for a 100,000 atom system with a 12A cutoff can approach 300MB, and will grow with the cube of the cutoff. This extra memory is distributed across processors during a parallel run, but a single workstation may run out of physical memory with a large system.

To avoid this, NAMD now provides a pairlistMinProcs config file option that specifies the minimum number of processors that a run must use before pairlists will be enabled (on fewer processors small local pairlists are generated and recycled rather than being saved, the default is "pairlistMinProcs 1"). This is a per-simulation rather than a compile time option because memory usage is molecule-dependent.

Additional information on reducing memory usage may be found at `http://www.ks.uiuc.edu/Research/namd/wiki/index.cgi?NamdMemoryReduction`

## 19.14 Improving Parallel Scaling

While NAMD is designed to be a scalable program, particularly for simulations of 100,000 atoms or more, at some point adding additional processors to a simulation will provide little or no extra performance. If you are lucky enough to have access to a parallel machine you should measure NAMD's parallel speedup for a variety of processor counts when running your particular simulation. The easiest and most accurate way to do this is to look at the "Benchmark time:" lines that are printed after 20 and 25 cycles (usually less than 500 steps). You can monitor performance during the entire simulation by adding "outputTiming *steps*" to your configuration file, but be careful to look at the "wall time" rather than "CPU time" fields on the "TIMING:" output lines produced. For an external measure of performance, you should run simulations of both 25 and 50 cycles (see the stepspercycle parameter) and base your estimate on the additional time needed for the longer simulation in order to exclude startup costs and allow for initial load balancing.

Multicore builds scale well within a single node, but may benefit from setting CPU affinity using the +setcpuaffinity +pemap <map> +commap <map> options described in CPU Affinity above. Experimentation is needed.

We provide standard (UDP), TCP, and ibverbs (InfiniBand) precompiled binaries for Linux clusters. The TCP version may be faster on some networks but the UDP version now performs well on gigabit ethernet. The ibverbs version should be used on any cluster with InfiniBand, and for any other high-speed network you should compile an MPI version.

SMP builds generally do not scale as well across nodes as single-threaded non-SMP builds because the communication thread is both a bottleneck and occupies a core that could otherwise be used for computation. As such they should only be used to reduce memory consumption or if for scaling reasons you are not using all of the cores on a node anyway, and you should run benchmarks to determine the optimal configuration.

Extremely short cycle lengths (less than 10 steps) will limit parallel scaling, since the atom migration at the end of each cycle sends many more messages than a normal force evaluation. Increasing margin from 0 to 1 while doubling stepspercycle and pairlistspercycle may help, but it is important to benchmark. The pairlist distance will adjust automatically, and one pairlist per ten steps is a good ratio.

NAMD should scale very well when the number of patches (multiply the dimensions of the patch grid) is larger or rougly the same as the number of processors. If this is not the case, it may be possible to improve scaling by adding "twoAwayX yes" to the config file, which roughly doubles the number of patches. (Similar options twoAwayY and twoAwayZ also exist, and may be used in combination, but this greatly increases the number of compute objects. twoAwayX has the unique advantage of also improving the scalability of PME.)

Additional performance tuning suggestions and options are described at `http://www.ks.uiuc.edu/Research/namd/wiki/?NamdPerformanceTuning`

# 20 NAMD Availability and Installation

NAMD is distributed freely for non-profit use. NAMD Git-2025-12-04 is based on the Charm++ messaging system and the Converse communication layer (`http://charm.cs.uiuc.edu/`) which have been ported to a wide variety of parallel platforms. This section describes how to obtain and install NAMD Git-2025-12-04.

## 20.1 How to obtain NAMD

NAMD may be downloaded from `http://www.ks.uiuc.edu/Research/namd/`. You will be required to provide minimal registration information and agree to a license before receiving access to the software. Both source and binary distributions are available.

## 20.2 Platforms on which NAMD will currently run

NAMD should be portable to any parallel platform with a modern C++ compiler to which Charm and Converse have been ported. Precompiled NAMD Git-2025-12-04 binaries are available for download for the following platforms:

- Linux on x86_64 processors
- Linux with NVIDIA GPUs (CUDA)
- Mac OS X on Intel and ARM processors
- Windows 10 and 11 supported by running Linux x86_64 builds (both CPU-only and CUDA) using WSL2 (Windows Subsystem for Linux)

Regarding Windows support, we note that community benchmarks have shown faster NAMD performance when using WSL2 versus a native build.

NAMD may be compiled for the following additional platforms:

- Linux or AIX on POWER processors
- Linux on ARM processors
- Linux on ARM or POWER processors with NVIDIA GPUs (CUDA)
- Linux on x86_64 processors with AMD GPUs (HIP/ROCm)
- Linux on x86_64 processors with Intel Xeon Phi coprocessors (MIC)

## 20.3 Installing NAMD

A NAMD binary distribution need only be untarred or unzipped and can be run directly in the resulting directory. When building from source code, "make release" will generate a self-contained directory and .tar.gz or .zip archive that can be moved to the desired installation location. CUDA and HIP builds include .so files that must be in the dynamics library path.

## 20.4 Compiling NAMD

We provide complete and optimized binaries for all common platforms to which NAMD has been ported. It should not be necessary for you to compile NAMD unless you wish to add or modify features or to improve performance by using an MPI library that takes advantage of special networking hardware.

Directions for compiling NAMD are contained in the release notes, which are available from the NAMD web site http://www.ks.uiuc.edu/Research/namd/ and are included in all distributions.

## 20.5 Documentation

All available NAMD documentation is available for download without registration via the NAMD web site http://www.ks.uiuc.edu/Research/namd/.

## References

[1] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, New York, 1987. 14, 15

[2] P. H. Axelsen and D. Li. Improved convergence in dual–topology free energy calculations through use of harmonic restraints. *J. Comput. Chem.*, 19:1278–1283, 1998. 127, 134

[3] C. H. Bennett. Efficient estimation of free energy differences with monte carlo data. *J. Comp. Phys.*, 22:245–268, 1976. 132

[4] F. C. Bernstein, T. F. Koetzle, G. J. B. Williams, J. E. F. Meyer, M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasumi. The protein data bank: A computer-based archival file for macromolecular structures. *J. Mol. Biol.*, 112:535–542, 1977. 15

[5] T. C. Beutler, A. E. Mark, R. C. van Schaik, P. R. Gerber, and W. F. van Gunsteren. Avoiding singularities and numerical instabilities in free energy calculations based on molecular simulations. *Chem. Phys. Lett.*, 222:529–539, 1994. 128, 132

[6] D. L. Beveridge and F. M. DiCapua. Free energy via molecular simulation: Applications to chemical and biomolecular systems. *Annu. Rev. Biophys. Biophys.*, 18:431–492, 1989. 127, 129

[7] L. Biedermannová, Z. Prokop, A. Gora, E. Chovancová, M. Kovács, J. Damborsky, and R. C. Wade. A single mutation in a tunnel to the active site changes the mechanism and kinetics of product release in haloalkane dehalogenase linb. *Journal of Biological Chemistry*, 287(34):29062–29074, 2012. 154

[8] O. Bignucolo, C. Chipot, S. Kellenberger, and B. Roux. Galvani offset potential and constant-pH simulations of membrane proteins. *J. Phys. Chem. B*, 126(36):6868–6877, 2022. PMID: 36049129. 171

[9] A. Bondi. van der Waals volumes and radii. *J. Phys. Chem.*, 68:441–451, 1964. 81

[10] S. Boresch and M. Karplus. The role of bonded terms in free energy simulations: I. theoretical analysis. *J. Phys. Chem. A*, 103:103–118, 1999. 128

[11] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. CHARMM: a program for macromolecular energy, minimization, and dynamics calculations. *J. Comput. Chem.*, 4(2):187–217, 1983. 14, 15, 25

[12] A. T. Brünger. *X-PLOR, Version 3.1, A System for X-ray Crystallography and NMR*. The Howard Hughes Medical Institute and Department of Molecular Biophysics and Biochemistry, Yale University, 1992. 14, 15, 25, 92, 93

[13] G. Bussi, D. Donadio, and M. Parrinello. Canonical sampling through velocity rescaling. *J. Chem. Phys.*, 126:014101, 2007. 93

[14] P. Carlsson, S. Burendahl, and L. Nilsson. Unbinding of retinoic acid from the retinoic acid receptor by random expulsion molecular dynamics. *Biophysical Journal*, 91(9):3151–3161, 2006. 154

[15] Y. Chen and B. Roux. Constant–pH hybrid nonequilibrium molecular dynamics–Monte Carlo simulation method. *J. Chem. Theory Comput.*, 11:3919–3931, 2015. 161

[16] Y. Chen and B. Roux. Generalized Metropolis acceptance criterion for hybrid non-equilibrium molecular dynamics–Monte Carlo simulations. *J. Chem. Phys.*, 142:024101, 2015. 164

[17] C. Chipot and D. A. Pearlman. Free energy calculations. the long and winding gilded road. *Mol. Sim.*, 28:1–12, 2002. 129

[18] C. Chipot and A. Pohorille, editors. *Free energy calculations. Theory and applications in chemistry and biology.* Springer Verlag, 2007. 127, 129, 132, 134

[19] V. Cojocaru, P. J. Winn, and R. C. Wade. Multiple, ligand-dependent routes from the active site of cytochrome P450 2C9. *Current Drug Metabolism*, 13(2):143–154, 2012. 154

[20] Y. Deng and B. Roux. Hydration of amino acid side chains: Nonpolar and electrostatic contributions calculated from staged molecular dynamics free energy simulations with explicit water molecules. *J. Phys. Chem. B*, 108:16567–16576, 2004. 130

[21] D. Frenkel and B. Smit. *Understanding Molecular Simulation From Algorithms to Applications.* Academic Press, California, 2002. 129

[22] J. Gao, K. Kuczera, B. Tidor, and M. Karplus. Hidden thermodynamics of mutant proteins: A molecular dynamics analysis. *Science*, 244:1069–1072, 1989. 127

[23] M. K. Gilson, J. A. Given, B. L. Bush, and J. A. McCammon. The statistical–thermodynamic basis for computation of binding affinities: A critical review. *Biophys. J.*, 72:1047–1069, 1997. 129

[24] D. Hamelberg, C. de Oliveira, and J. McCammon. Sampling of slow diffusive conformational transitions with accelerated molecular dynamics. *J. Chem. Phys.*, 127:155102, 2007. 140

[25] D. Hamelberg, J. Mongan, and J. McCammon. Accelerated molecular dynamics: a promising and efficient simulation method for biomolecules. *J. Chem. Phys.*, 120(24):11919–11929, 2004. 140

[26] E. Harder, V. M. Anisimov, I. V. Vorobyov, P. E. M. Lopes, S. Y. Noskov, A. D. MacKerell, and B. Roux. Atomic level anisotropy in the electrostatic modeling of lone pairs for a polarizable force field based on the classical drude oscillator. *J. Chem. Theory Comput.*, 2(6):1587–1597, 2006. 68

[27] D. J. Hardy, Z. Wu, J. C. Phillips, J. E. Stone, R. D. Skeel, and K. Schulten. Multilevel summation method for electrostatic force evaluation. *J. Chem. Theory Comput.*, 11:766–779, 2015. 62, 63

[28] G. D. Hawkins, C. J. Cramer, and D. G. Truhlar. Parametrized models of aqueous free energies of solvation based on pairwise descreening of solute atomic charges from a dielectric medium. *J. Phys. Chem.*, 100:19824–19839, 1996. 79, 81

[29] J. Huang, J. A. Lemkul, P. K. Eastman, and A. D. MacKerell Jr. Molecular dynamics simulations using the Drude polarizable force field on GPUs with OpenMM: Implementation, validation, and benchmarks. *J. Comput. Chem.*, 39(21):1682–1689, 2018. 68

[30] W. Jiang, C. Chipot, and B. Roux. Computing relative binding affinity of ligands to receptor: An effective hybrid single-dual-topology free-energy perturbation approach in NAMD. *J. Chem. Inf. Model.*, 59(9):3794–3802, 2019. 137

[31] W. Jiang, D. Hardy, J. Phillips, A. MacKerell, K. Schulten, and B. Roux. High-performance scalable molecular dynamics simulations of a polarizable force field based on classical Drude oscillators in NAMD. *J. Phys. Chem. Lett.*, 2:87–92, 2011. 67

[32] S. Jo and W. Jiang. A generic implementation of replica exchange with solute tempering (REST2) algorithm in NAMD for complex biophysical simulations. 197:304–311, 2015. 146

[33] P. M. King. Free energy via molecular simulation: A primer. In W. F. Van Gunsteren, P. K. Weiner, and A. J. Wilkinson, editors, *Computer simulation of biomolecular systems: Theoretical and experimental applications*, volume 2, pages 267–314. ESCOM, Leiden, 1993. 127, 129

[34] J. G. Kirkwood. Statistical mechanics of fluid mixtures. *J. Chem. Phys.*, 3:300–313, 1935. 127, 129

[35] D. B. Kokh, M. Amaral, J. Bomke, U. Grädler, D. Musil, H.-P. Buchstaller, M. K. Dreyer, M. Frech, M. Lowinski, F. Vallee, M. Bianciotto, A. Rak, and R. C. Wade. Estimation of drug-target residence times by $\tau$-random acceleration molecular dynamics simulations. *Journal of Chemical Theory and Computation*, 14(7):3859–3869, 2018. PMID: 29768913. 154

[36] P. A. Kollman. Free energy calculations: Applications to chemical and biochemical phenomena. *Chem. Rev.*, 93:2395–2417, 1993. 129

[37] E. A. Koopman and C. P. Lowe. Advantages of a Lowe-Andersen thermostat in molecular dynamics simulations. *J. Chem. Phys.*, 124:204103, 2006. 96

[38] G. Lamoureux, E. Harder, I. V. Vorobyov, B. Roux, and A. D. MacKerell. A polarizable model of water for molecular dynamics simulations of biomolecules. *Chem. Phys. Lett.*, 418(1-3):245–249, 2006. 67, 69

[39] G. Lamoureux and B. Roux. Modeling induced polarization with classical Drude oscillators: Theory and molecular dynamics simulation algorithm. *J. Chem. Phys.*, 119(6):3025–3039, 2003. 67

[40] J. A. Lemkul, J. Huang, B. Roux, and A. D. MacKerell Jr. An empirical polarizable force field based on the classical Drude oscillator model: Development history and recent applications. *Chem. Rev.*, 116(9):4983–5013, 2016. PMID: 26815602. 68

[41] N. Lu, D. A. Kofke, and T. B. Woolf. Improving the efficiency and reliability of free energy perturbation calculations using overlap sampling methods. *J. Comput. Chem.*, 25:28–39, 2004. 132, 137

[42] S. K. Lüdemann, V. Lounnas, and R. C. Wade. How do substrates enter and products exit the buried active site of cytochrome P450cam? 1. random expulsion molecular dynamics investigation of ligand access channels and mechanisms. *Journal of Molecular Biology*, 303(5):797–811, 2000. 154

[43] Z. M., T. P. Straatsma, and M. J. A. Separation-shifted scaling, a new scaling method for Lennard-Jones interactions in thermodynamic integration. *J. Chem. Phys.*, 100:9025–9031, 1994. 128, 132

[44] J. D. C. Maia, G. A. Urquiza Carvalho, C. P. Mangueira Jr, S. R. Santana, L. A. F. Cabral, and G. B. Rocha. Gpu linear algebra libraries and gpgpu programming for accelerating mopac semiempirical quantum chemistry calculations. *J. Chem. Theory Comput.*, 8(9):3072–3081, 2012. 174

[45] A. E. Mark. Free energy perturbation calculations. In P. v. R. Schleyer, N. L. Allinger, T. Clark, J. Gasteiger, P. A. Kollman, H. F. Schaefer III, and P. R. Schreiner, editors, *Encyclopedia of computational chemistry*, volume 2, pages 1070–1083. Wiley and Sons, Chichester, 1998. 127, 129

[46] S. J. Marrink, A. H. de Vries, and A. E. Mark. Coarse grained model for semiquantitative lipid simulations. *J. Phys. Chem. B*, 108:750–760, 2004. 70

[47] S. J. Marrink, H. J. Risselada, S. Yefimov, D. P. Tieleman, and A. H. de Vries. The martini forcefield: coarse grained model for biomolecular simulations. *J. Phys. Chem. B*, 111:7812–7824, 2007. 70

[48] J. A. McCammon and S. C. Harvey. *Dynamics of Proteins and Nucleic Acids.* Cambridge University Press, Cambridge, 1987. 14

[49] M. Melo, R. Bernardi, T. Rudack, M. Scheurer, C. Riplinger, J. Phillips, J. Maia, G. Rocha, J. Ribeiro, J. Stone, F. Nesse, K. Schulten, and Z. Luthey-Schulten. NAMD goes quantum: An integrative suite for QM/MM simulations. *Nat. Methods*, 15:351–354, 2018. 174

[50] Y. Miao, V. Feher, and J. McCammon. Gaussian accelerated molecular dynamics: Unconstrained enhanced sampling and free energy calculation. *J. Chem. Theory Comput.*, 11:3584–3595, 2015. 142, 143, 144

[51] N. Michaud-Agrawal, E. J. Denning, T. B. Woolf, and O. Beckstein. MDAnalysis: A toolkit for the analysis of molecular dynamics simulations. *J. Comput. Chem.*, 32(10):2319–2327. 116

[52] L. Monticelli, S. Kandasamy, X. Periole, and R. L. D. T. S. Marrink. The martini coarse grained forcefield: extension to proteins. *J. Chem. Theory Comput.*, 4:819–834, 2008. 70

[53] F. Neese. The ORCA program system. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.*, 2:73–78, 2012. 174

[54] J. P. Nilmeier, G. E. Crooks, D. D. L. Minh, and J. D. Chodera. Nonequilibrium candidate Monte Carlo is an efficient tool for equilibrium simulation. *Proc. Natl. Acad. Sci. USA*, 108:E1009–E1018, 2011. 164

[55] J. K. Noel, P. C. Whitford, K. Y. Sanbonmatsu, and J. N. Onuchic. SMOG@ctbp: simplified deployment of structure-based models in GROMACS. *Nucleic Acids Research*, 38:W657–61, 2010. 159

[56] A. Onufriev, D. Bashford, and D. A. Case. Modification of the generalised born model suitable for macromolecules. *J. Phys. Chem.*, 104:3712–3720, 2000. 79

[57] A. Onufriev, D. Bashford, and D. A. Case. Exploring protein native states and large-scale conformational changes with a modified generalized born model. *Proteins: Struct., Func., Gen.*, 55:383–394, 2004. 79, 81, 82

[58] Y. Pang, Y. Miao, Y. Wang, and J. McCammon. Gaussian accelerated molecular dynamics in NAMD. *J. Chem. Theory Comput.*, 13:9–19, 2017. 144

[59] D. A. Pearlman. A comparison of alternative approaches to free energy calculations. *J. Phys. Chem.*, 98:1487–1493, 1994. 127

[60] B. K. Radak, C. Chipot, D. Suh, S. Jo, W. Jiang, J. C. Phillips, K. Schulten, and B. Roux. Constant-pH molecular dynamics simulations for large biomolecular systems. *J. Chem. Theory Comput.*, 13:5933–5944, 2017. 161, 164

[61] B. K. Radak and B. Roux. Efficiency in nonequilibrium molecular dynamics Monte Carlo simulations. *J. Chem. Phys.*, 145:134109, 2016. 164

[62] J. V. Ribeiro, R. C. Bernardi, T. Rudack, J. E. Stone, J. C. Phillips, P. L. Freddolino, and K. Schulten. QwikMD-integrative molecular dynamics toolkit for novices and experts. *Sci. Rep.*, 6:26536, 2016. 174

[63] Richard J. Gowers, Max Linke, Jonathan Barnoud, Tyler J. E. Reddy, Manuel N. Melo, Sean L. Seyler, Jan Domaski, David L. Dotson, Sbastien Buchoux, Ian M. Kenney, and Oliver Beckstein. MDAnalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations. In Sebastian Benthall and Scott Rostrup, editors, *Proceedings of the 15th Python in Science Conference*, pages 98 – 105, 2016. 116

[64] A. Roitberg and R. Elber. Modeling side chains in peptides and proteins: Application of the locally enhanced sampling technique and the simulated annealing methods to find minimum energy conformations. *J. Chem. Phys.*, 95:9277–9287, 1991. 150

[65] M. Schaefer and C. Froemmel. A precise analytical method for calculating the electrostatic energy of macromolecules in aqueous solution. *J. Mol. Biol.*, 216:1045–1066, 1990. 79

[66] K. Schleinkofer, P. J. Winn, S. K. Lüdemann, R. C. Wade, et al. Do mammalian cytochrome P450s show multiple ligand access pathways and ligand channelling? *EMBO Reports*, 6(6):584–589, 2005. 154

[67] H. M. Senn and W. Thiel. Qm/mm methods for biomolecular systems. *Angew. Chem. Int. Ed. Engl.*, 48(7):1198–1229, 2009. 174

[68] M. R. Shirts, D. L. Mobley, J. D. Chodera, and V. S. Pande. Accurate and efficient corrections for missing dispersion interactions in molecular simulations. *J. Phys. Chem. B*, 111(45):13052–13063, 2007. 60

[69] S. Shobana, B. Roux, and O. S. Andersen. Free energy simulations: Thermodynamic reversibility and variability. *J. Phys. Chem. B*, 104(21):5179–5190, 2000. 139

[70] C. Simmerling, T. Fox, and P. A. Kollman. Use of locally enhanced sampling in free energy calculations: Testing and application to the $\alpha \rightarrow \beta$ anomerization of glucose. *J. Am. Chem. Soc.*, 120(23):5771–5782, 1998. 150

[71] C. Simmerling, M. R. Lee, A. R. Ortiz, A. Kolinski, J. Skolnick, and P. A. Kollman. Combining MONSSTER and LES/PME to predict protein structure from amino acid sequence: Application to the small protein CMTI-1. *J. Am. Chem. Soc.*, 122(35):8392–8402, 2000. 150

[72] R. D. Skeel and J. J. Biesiadecki. Symplectic integration with variable stepsize. *Ann. Numer. Math.*, 1:191–198, 1994. 91

[73] J. Srinivasan, M. W. Trevathan, P. Beroza, and D. A. Case. Application of a pairwise generalized born model to proteins and nucleic acids: inclusion of salt effects. *Theor Chem Acc*, 101:426–434, 1999. 79, 81

[74] H. A. Stern. Molecular simulation with variable protonation states at constant pH. *J. Chem. Phys.*, 126:164112, 2007. 161

[75] J. J. Stewart. Mopac: a semiempirical molecular orbital program. *J. Comp.-Aided Mol. Design*, 4(1):1–103, 1990. 174

[76] W. C. Still, A. Tempczyk, R. C. Hawley, and T. Hendrickson. Semianalytical treatment of solvation for molecular mechanics and dynamics. *J. Am. Chem. Soc.*, 112:6127–6129, 1990. 79

[77] T. P. Straatsma and J. A. McCammon. Multiconfiguration thermodynamic integration. *J. Chem. Phys.*, 95:1175–1118, 1991. 129, 130

[78] T. P. Straatsma and J. A. McCammon. Computational alchemy. *Annu. Rev. Phys. Chem.*, 43:407–435, 1992. 129

[79] B. T. Thole. Molecular polarizabilities calculated with a modified dipole interaction. *Chem. Phys.*, 59:341–350, 1981. 68

[80] P. Van Duijnen and M. Swart. Molecular and atomic polarizabilities: Thole's model revisited. *J. Phys. Chem. A*, 102(14):2399–2407, 1998. 68

[81] W. F. van Gunsteren. Methods for calculation of free energies and binding constants: Successes and problems. In W. F. Van Gunsteren and P. K. Weiner, editors, *Computer simulation of biomolecular systems: Theoretical and experimental applications*, pages 27–59. Escom, The Netherlands, 1989. 129

[82] H. Vashisth and C. F. Abrams. Ligand escape pathways and (un) binding free energy calculations for the hexameric insulin-phenol complex. *Biophysical Journal*, 95(9):4193–4204, 2008. 154

[83] L. Wang, R. A. Friesner, and B. J. Berne. Replica exchange with solute scaling: A more efficient version of replica exchange with solute tempering (REST2). *J. Phys. Chem. B*, 115(30):9431–9438, 2011. 146

[84] Y. Wang, C. Harrison, K. Schulten, and J. McCammon. Implementation of accelerated molecular dynamics in NAMD. *"Comp. Sci. Discov."*, 4:015002, 2011. 140

[85] J. Weiser, P. Senkin, and W. C. Still. Approximate atomic surfaces from linear combinations of pairwise overlaps (LCPO). *J. Comput. Chem.*, 20:217–230, 1999. 83

[86] P. C. Whitford, J. K. Noel, S. Gosavi, A. Schug, K. Y. Sanbonmatsu, and J. N. Onuchic. An all-atom structure-based potential for proteins: Bridging minimal models with all-atom empirical forcefields. *Proteins*, 75(2):430–441, 2009. 159

[87] P. C. Whitford, A. Schug, J. Saunders, S. P. Hennelly, J. N. Onuchic, and K. Y. Sanbonmatsu. Nonlocal helix formation is key to understanding s-adenosylmethionine-1 riboswitch function. *Biophysical Journal*, 96(2):L7 – L9, 2009. 159

[88] P. J. Winn, S. K. Lüdemann, R. Gauges, V. Lounnas, and R. C. Wade. Comparison of the dynamics of substrate access channels in three cytochrome P450s reveals different opening mechanisms and a novel functional role for a buried arginine. *Proceedings of the National Academy of Sciences*, 99(8):5361–5366, 2002. 154

[89] R. W. Zwanzig. High–temperature equation of state by a perturbation method. i. nonpolar gases. *J. Chem. Phys.*, 22:1420–1426, 1954. 129

# Index