Griffin Manual

by René Staritzbichler

GRIFFIN calculates realistic lipid conformations even for large TM protein systems with complex topology starting from a pre-equilibrated bilayer. The implicit force field of GRIFFIN guides buried bilayer atoms towards the surface of the protein, while water or lipid atoms outside of the protein volume experience the interaction forces of the protein. While in most cases the force field will be calculated for a protein only, also other molecules like lipids can be included into the implicit force field. After creating the Griffin formatted input files, the workflow consists of three subsequent steps: carving, calculating the implicit force field and optimization. During latter, Griffin needs to be linked to a MD simulation of the bilayer. Additional features include the definition of exclusion volumes, where the interior of geometric objects is kept free of specific molecule types.

Conventions: lines belonging into a file start with '=> ' and lines being part of a command to be executed in a shell or flags are surrounded by ' like: 'COMMAND OPTION'.

Overview

Chapters of this manual:

Appendix:

How to work with this manual:

The actual workflow of Griffin is described in sections 2) to 5). After the installation it is probably most useful to try to follow the tutorial. This should be sufficient for most cases. Most modifications will be adjustments of paths, file names etc.
However, it is necessary for the user to carefully check and potentially tune the output of the carving. The quality of the final result will depend on the carving output. The carver provides many options to control the output.

0) Griffin installation:

Griffin consists of a set of independent executables, which at the moment have to be compiled independently.

0.1) Third-party software/libraries:

a) IPC: The Griffin optimization phase relies heavily on inter-process communication (IPC). More explicitly, message queues are used for the communication between Griffin and the MD package. IPC should exist on most Linux systems. Type:
'ipcs'
to get information about open message queues on your system. I have no experience with IPC on other operating systems. Mac seems to have it. Unfortunately there are slight differences in the syntax already between the different Linux systems we have Griffin running on (SUSE, Redhat).
Remove message queues with
'ipcrm –q QUEUE_ID'
on the node with the queue, or use the cleanup tool:
'griffin_toolbox.exe –cleanup_queues FILE'
FILE contains a list of all nodes on which you want to erase your message queues.
Be sure not to have other processes running on these nodes that depend on message queues.
Obviously this can delete only queues that you own, while the queues belonging other users may still influence your job if they use the same 'channel'. This may be the case if another user war running Griffin on the same node.
Queues will not be cleaned automatically if Griffin crashes.

b) Boost: a commonly used and very powerful C++ library. We use smart pointers and format classes from boost. Should be easy to install. (www.boost.org)
If you install boost in a non standard location (e.g. if you don't have root permissions) you will have to include the path in:
=> CPPFLAGS=-I/PATH_TO_BOOST/

Only for compiling the **parallel** version of the daemon:

c) MPI: if you want to run Griffin on multiple processors and/or machines, you will need the header mpi.h to be located in the c++ include path (/usr/local/include/) or use
=> CPPFLAGS = -I/PATH_TO_MPI_H/
in your makefile.
(more detail below)

0.2) How to compile

For all executables individual makefiles are located in the top griffin directory. Check paths and names.
There are the following makefiles available (see section 1a) all based on calling g++:

makefile_tools
makefile_carver
makefile_forcefield
makefile_messenger
makefile_daemon
makefile_daemon_mpi (ADJUST path to mpicxx !!!)

Clean before building if any macro definition has changed, like when you first compiled all non-mpi executables and then want to build the mpi daemon (or vice versae).

'make -f MAKEFILE clean'

then:

'make –f MAKEFILE'

Call make from the Griffin root directory where all makefiles are located. Most people will work on multi-processor environments and may find the '-j NUMBER_PROCESSORS' flag useful.

NOTE: Griffin is tested with **g++ 4.1.2 and 4.4**, other compilers may need some adjustments.

There are a few debugging options, which require to (clean and) recompile the executables placing one or more of the following flags (as for the mpi version of the daemon) using the '-D' flag when calling g++

=> g++ … -DDEBUG insane amount of output
=> g++ … -DMONITORING for RAM monitoring
=> g++ … -DTIMERS for analyzing grid performance
=> g++ … -DVMD_OUTPUT VMD formatted output for visualization of forces
=> g++ … -DFORCE_COORDINATES prints atom coordinates with forces

0.3) Compiling a MPI executable

Although explained in the previous two sections, a view more words about the MPI version may be helpful.

Here we refer to daemon as a process waiting in the background to be called. Often this involves a fork(), the splitting of a process into two. This is only used for the serial version of Griffin. MPI and fork() are no friends, i.e. when using MPI parallelization the processes are managed by MPI and that does not go together with being a forked daemon.

Compiling with MPI requires several environment variables to be set correctly. Most systems will have a wrapper compiler for that. Make sure to get the makefile adjusted correctly, either calling that wrapper or setting all the variables manually.

In our case the makefile line
=> CXX = g++

has to be replaced by

=> CXX = mpicxx

which in turn is the g++ compiler with all required variables set.

1) Griffin basics:

1.0) The executables:

Griffin contains several executables for the steps involved in the Griffin workflow:

griffin_tools.exe: contains several scripts that are needed for the preparation steps:
-namd2griffin: will create a Griffin formatted input file from NAMD formatted .xpsf and Charmm parameter files (see 2a)
-gmx2griffin: creates Griffin formatted input files from Gromacs dump file and Charmm parameter file (see 2a)
-geometric_objects: allows to quickly visualize definitions in VMD (see 2b)

griffin_carver.exe: erases all waters inside the protein volume and as many lipids to conserve the lipid density (section 3)
griffin_force_grid.exe: calculates the implicit force field grid (section 4)
griffin_daemon.exe: returns forces calculated with the Griffin force field acting on atoms of a molecule (section 5a)
griffin_daemon_mpi.exe: the parallel version of the Griffin daemon (section 5b)
griffin_messenger.exe: is connecting the MD simulation with Griffin (section 5)

Call the executables without any flags to get a list of available options.

1.1) The implicit potential force field

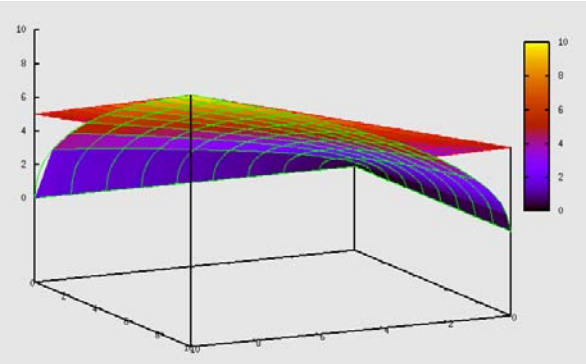
The implicit force field is the 'heart' of Griffin. It stores on a grid the force an atom will experience from the protein. The gridpoint object is the key to the power of the grid to describe very different 'force-compositions' for different gridpoints.

In order to decompose the van der Waals interaction such that a potential force can be calculated, it has to be separated into attractive and repulsive interaction. By applying the geometric average to both … it is straightforward to define the potential force.

We use message queues to communicate with the Griffin daemon, which carries the implicit force field. The force field may be rather large, therefore reading becomes a time issue. Having the force field running as daemon, ready to act upon receiving a message avoid long reading times.

a) van der Waals interaction

The van der Waals interaction may be decomposed in factors from the interaction partners using a geometric rather than a arithmetic one. The figure shows the difference: the arithmetic mean value is planar, while the arithmetic mean has the shape of the lower plot. The two values agree well with other, except for when one of the values is close to zero, while the other isn't.



1.2) Linking Griffin with MD: Gromacs vs NAMD: kcal/Å vs kjoule/nm
While Griffin calculates forces caused by the protein (which itself is non existent during the simulation) it needs to be linked to a MD package for it to calculate the forces between the explicit atoms and to update their coordinates. The requirement to the MD package is to provide an interface for external programs, where MD writes the coordinates of its atoms and reads forces calculated by an external program. NAMD had such an interface and for Gromacs Gerrit Groenhof helped me implementing it. (Thanks Gerrit!!!)

Beside different formats the most crucial difference between Gromacs and Namd is their unit system: NAMD uses kcal and Angstroem, while Gromacs works with kjoule and nm.
The necessary conversions happen at different steps for the different type of interactions.
For the vdW forces, the conversion happens when creating the .gfn files, by adjusting the sigma and epsilon values. This is done automatically when creating the .gfn files.
For the electrostatic force a conversion factor has to be applied when calculating the force grid:
'griffin_force_grid –kjoule_nm_units'
SForces have to be adjusted:
'griffin_daemon.exe –sforce_scale VALUE'

NOTE: SForces of 1.0 for NAMD are equivalent to **0.41868 in GROMACS**.

2.1) Creating .gfn files of protein and bilayer system

Griffin uses its own format for molecules, refered to as .gfn. The .gfn files combine information contained in .pdb, parameter and topolopgy files. See appendix XXX for detailed information about the .gfn format. The griffin_toolbox.exe contains several modes to obtain these files. Griffin is tested currently to work with NAMD and GROMACS. For each of them there is a mode in the toolbox.

A necessary step before creating .gfn files is to align the protein with the membrane. Make sure protein coordinates reflect orientation and burial relative to the lipid bilayer. Check for tools to insert a given protein structure into the membrane:
http://opm.phar.umich.edu/
http://pdbtm.enzim.hu/
(http://expasy.org/tools/ topology prediction section)
http://www.ch.embnet.org/software/TMPRED_form.html
Often we use the band of hydrophobicity on the outside surface of the protein as a guide to the extents of the membrane, orient first along the z-axis, and then laterally translate (e.g. using Charmm to adjust the coordinates).

With **NAMD**:

'griffin_toolbox.exe -namd2griffin NAME.pdb NAME.xpsf CHARMM_PARAMETER_FILE'

With **GROMACS**:

You will have to create a gmx dump file for BOTH the implicit protein as well as for the bilayer system (see a more detailed explanation in Appendix C). To create a .gfn file:

'griffin_toolbox.exe –gmx2griffin NAME.gmx NAME.gfn CHARMM'

The CHARMM file has to contain the [atomtypes] section, in older versions this is
ffcharmm27nb.itp
in more recent versions
ffnb.itp or
ffnonbonded.itp
under
GROMACS_DIR/share/gromacs/top/charmm27.ff/

check consistency of molecule names, e.g. TIP3 vs SOL

There might be shift in the coordinates from NAMD to GROMACS. Gromacs usually uses a box starting at 0,0,0, while Namd has box centered at the origin.
The editconf script from Gromacs allows to perform coordinate translations.

2.2) Geometric Objects

There are two scenarios where the usage of geometric objects is useful:
a) removing cavities within the protein
b) keeping certain or all molecule types from entering a certain volume.
A simple solution is the definition of geometric objects that can be molecule type dependent.
Geometric objects are used during carving and calculating the force grid.

Three kind of objects may be defined (in any combination in the SURF file):
=> Sphere
=> x,y,z,radius
=> Cube
=> x,y,z,xsize,ysize,zsize
=> Cylinder
=> x,y,z,radius,height

The SURF file has to contain first:
NR_OBJECTS (any positive integer)
then for each object:
MOLECULE_TYPE (e.g. “all”, “POPE”, “POPC”, “TIP3”)
and the object block as listed above, e.g.:

This is an example of three objects, the first for POPE, the second for TIP3 and the third for any molecule:
=> 3
=> POPE
=> Sphere
=> 0.0 0.0 0.0 5.0
=> TIP3
=> Cube
=> 5.08 -7.43 5.83 3.0 2.0 4.0
=> all
=> Cylinder
=> 10.0 10.0 0.0 2.0 8.0

The toolbox has a mode
‘-geometric_objects SURF VMD’
which allows to translate a SURF file, containing definitions of geometrical objects, into a set of vmd commands. This allows for fast visual testing of given definitions. One can call it by:
‘vmd PROTEIN.pdb –e VMD’
and examine whether the objects are placed correctly.

3) Carving

The carving erases all solute molecules buried in the protein volume. For lipids the number to be deleted is determined such that the lipid density is conserved.
There is a straightforward basic way to use the carving. However, this is the step where the user needs to check the results. There are several fine-tuning flags that provide full control to the user.
Call executable without further arguments or with ‘-help’ to see a short description of available flags.

NOTE: the Griffin format is explained in appendix and in the toolbox. Call according executable without arguments to see its description.

The basic usage consists of the required flags and specifying either ‘-dimensions’ or ‘-auto_dimensions’. If you want to use Griffin with Gromacs you need also to set ‘-kjoule_nm_units’.

3.1) Required flags:

-membrane FILE
Griffin formatted input (.gfn) of the membrane/solvent system to be carved

-solute FILE
Griffin formatted input (.gfn) of molecule to be embedded in membrane (protein)

-lipid_types RESNAME [RESNAME]
lipid types in membrane, e.g.: POPE POPC, you can provide up to 20 names

-solvent_names RESNAME [RESNAME]
non-lipid molecule types, e.g.: TIP3, NACL. As well here you can provide up to 20 types.

-write_carved FILE
Griffin-formatted output (.gfn) with carved membrane/solvent

3.2) Limits

The carving relies on proper definitions of the bilayer box limits, which in most cases users will be aware of. The dimensions are needed to calculate lipid densities. Most commonly the user will provide box limits by:
-dimensions XMIN XMAX YMIN YMAX ZMIN ZMAX Z_BILAYER_CENTER Z_BILAYER_THICKNESS

Another possibility is to let the program figure out the limits:

-auto_dimensions MODE

MODE can be 'max', calculating maximal values, 'average', using mean values or 'standard-deviation' the mean value plus standard deviation.
‘max’ will usually overestimate the size of the box if periodic boundary conditions are used, because of lipid tails or head groups sticking out the actual box.
‘mean’ uses the maximum values in all dimensions of each lipid and averages over them.
‘standard-deviation’ sums the standard deviation to the ‘mean’ values.

The flag
-write_histograms FILE
allows to analyze the distributions of atoms in the box and to determine accurate dimensions ‘by hand’.

3.3) Surface

Both carving as well as calculating the force grid rely on determining whether a point is within or outside of the protein volume. Reference is the surface of the protein, represented by a point distribution. By default the surface will be smoothed analogous to the rolling probe algorithm, but can be switched to VDW surfaces (plain spheres) by
‘-vdw_surf RADIUS’

The radius of the rolling sphere for the smoothed surf can be altered by
‘-smooth_surf RADIUS’ the default is 1.4Å.

Technically the surface is a calculated using a grid. The distance of gridpoints can be controlled by:
‘-surf_voxel_size VALUE’ with a default of 0.2Å.

Debug options:

The atom (VDW) radii can be altered by applying an general offset and/or a magnification factor:
'-magnify_radii OFFSET FACTOR'

In order to compare the Griffin surfaces with the ones from e.g. VMD generic radii can be used for the atoms:
'-use_generic_radii'

3.4) Geometric objects

One of the advanced features of Griffin is that the user can define any number of geometric objects in which specified molecule types are considered as being within the protein volume.
While it is possible to use the geometric objects only during the calculation of the force field or only during the carving, most useful will be to use them during both. Call:
'-add_surf_objects FILE'
The format of FILE is explained here.

3.5) Output

It is highly recommended to examine both the carved as well as the erased pdbs who complement each other:
'-write_erased_pdb PDB_FILE'
'-write_carved_pdb PDB_FILE'

It is also useful to examine the surface, which carving and force grid calculation use as reference.
'-write_surf_as_pdb PDB_FILE'
Check especially for unwanted cavities within the proteins volume. You may have to switch off the 'autobond' creation in VMD.

3.6) Carving control

The standard setup will be sufficient for most cases. However, there are several means to modify the carving results after examining the output pdbs.

There are two flags allowing to specify lipids that shall be either explicitly kept or removed. If there are molecules that were erased but should be kept:
-keep_molecules POPE 123 POPE 234 …
If molecules were not erased that seem to be useful to remove:
-remove_molecules POPC 21 POPC 43 …

To modify the number of lipids that shall be removed in each layer call:
'-nr_lipids_to_be_removed NUMBER NUMBER'

if '-n' is given, the number calculated automatically is reduced by n,

if '+n' is given, the number is increased by n

if 'n' is given, n lipids will be removed

3.7) Gromacs:

If Gromacs is used, the '-kjoule_nm_units' flag has to be set and all dimensions, probe radii, geometric objects and all .gfn have to be in nm rather then Angstroem! Output pdbs will nonetheless be in Angstroem.

4) Force grid calculation

In this step the implicit force field grid is calculated which will be applied during the optimization. A number of flags are equivalent to the ones in the carving and have to match.

4.1) Required flags

-write_force_grid FILE
output force-field grid file

-solute FILE
Griffin formatted input (.gfn) of protein molecule

4.1) The surface

Like for the carving the surface is used as reference for indicating a point being inside or outside the protein volume and for determining the shortest way to the surface when being inside. Therefore the same surface controlling parameters as in the carving apply.

The bin size of the force and the surface grids are both controlled by:
-voxel_size VALUE (default 0.5)

By default no physical forces are calculated for the surface points. To fill that empty layer with physical forces:
-fill_surf_layer

Offset and optional scaling factor for increasing VdW radius for calculating protein surface:
-magnify_radii OFFSET [FACTOR] (defaults: 1.4, 1.0)

Calculate smooth protein surface (used by default) with variable probe radius:
-smooth_surf PROBE_RADIUS(default: 1.4 Angstroems)

Calculate vdW molecular rather than smooth surface, with variable probe radius:
-vdw_surf PROBE_RADIUS(default: 1.4 Angstroems)

Read Griffin-formatted, pre-calculated protein surface instead of calculating if from the input PDB:
-read_surf FILE

4.2) Geometric objects

One of the advanced features of Griffin is that the user can define any number of geometric objects in which specified molecule types are considered as being within the protein volume.
While it is possible to use the geometric objects only during the calculation of the force field or only during the carving, most useful will be to use them during both. Call:
'-add_surf_objects FILE'
The format of FILE is explained here.

4.3) Force selection

At the moment there are four types of forces and energies which are mounted by default with the following cutoffs:

- surface forces and energies, no cutoff, cannot be modified
- coulomb forces and energies, cutoff: 18Å
- attractive van der Waals forces and energies, cutoff: 12Å
- repulsive van der Waals forces and energies, cutoff: 8Å

If you want to change something from the default force composition and cutoffs you will have to call '-force_file FILE' and provide a file containing:

The surface forces cannot be altered here. When starting the daemon used for optimization, one can rescale the surface forces by calling 'griffin_daemon.exe –sforce_scale VALUE'. Similarly the daemon allows rescaling the interaction forces.

In order to reduce memory one can avoid that force objects are mounted if the stored force vectors are too small:
-minimum_potential_force_magnitude VALUE(default: 1e-11)

4a) Parallel calculation

The calculation of a gridpoint does not depend on any other gridpoint. Therefore the grid can be calculated in separate and independent processes, using:
'-parallel X Y'
where Y^3 is the total number of processes, Y the number of intervals per dimension. X identifies the subprocess and goes from 0 to $Y^3 - 1$.

When calculating the grid in parallel the actual grid has to be recombined from the subgrids into one complete grid using:
-combine_subgrids FILE_PREFIX N
FILE_PREFIX is the subgrid filename (before "_SUBGRID.txt")

4c) Gromacs

When using Gromacs an additional flag has to be set:
'–kjoule_nm_units'

5) The Griffin daemon

The daemon is the 'carrying' the force field. The MD simulation calls the daemon to get the forces for the explicit system calculated.

5.1) Required flags

-force_grid FILE
reads Griffin-formatted force-grid from FILE
1 parameter(s)

-unit_system TYPE
coordinate and energy unit system used in the input .gfn file
TYPE can be 'namd' or 'gromacs'
1 parameter(s)

-lipid_types RESNAME [RESNAME]
list of lipid types in the system (needed for redirection)
from 1 to 9 parameters

5.2) Force control

Upon loading the forces into the daemon, there are several flags allowing to modify them.

Resetting magnitude of surface forces to VALUE:
-sforce_scale VALUE(default: 1)

Multiplying one or more interaction TERMs with VALUE.
-rescale TERM VALUE [TERM VALUE]
TERM can be 'vdw-attractive', 'vdw-repulsive' or 'coulumb'
e.g.: -rescale vdw-attractive 1.5 coulomb 10

Don't calculate surface forces for hydrogen atoms to avoid unreasonable accelerations:
-sforce_exclude_hydrogens
Recalculate zero forces only in specified frequency:
-neighbor_list_update VALUE
This can be very efficient if many atoms are expected to have zero forces.

The surface force acting on a lipid atom inside the protein volume is redirected towards the center of the corresponding molecule if the angle (in degrees) between the surface-force vector and the vector to the molecular center exceeds the threshold VALUE:
-sforce_reset_min_angle VALUE(default: 110 degrees)

6) The Griffin messenger

The messenger is the interface of external programs and the Griffin daemon. It exploits IPC functionality that should be available on most Linux distributions. It has three main modes as needed for controlling Griffin. Additionally it provides two functions that allow one program to wait until another one is ready.

Set up daemon for receiving jobs (by calling '-forces'):
'-setup MOLECULES_INPUT_FILE COORD_INPUT_FILE FORCES_OUTPUT_FILE'

COORD_INPUT_FILE and FORCES_OUTPUT_FILE are used for read/write when calling '-forces'

Call daemon to calculate forces for coordinate file in input directory and to write them into force output file as given in '-setup':
'–forces'

Terminate grid server daemon:
'-terminate'

Embedding the following two calls in submission scripts allows a program to wait for the termination of another one before proceeding:

Wait on ipc channel VALUE until receives a message, then terminates:
'-wait VALUE(default:12342)'

Send empty message on ipc channel VALUE, then terminates:
'-send VALUE'

NOTE:

VALUE in '-wait' and '-send' have to match in order for them to communicate!

7.1) Running Griffin on single processors

7.1.1) PBS

Up till now the Griffin workflow was only tested in conjunction with PBS. Beside providing the program with its requested resources it also allows to access information about the resources, which is used mostly when running in parallel.

7.1.2) General setup

After the PBS settings we set a trap to ensure that the daemon gets properly terminated if e.g. the MD package crashes:
'trap PATH/'griffin_messenger.exe –terminate' 0'
(note: the inner ' need to be typed!)

A trap allows to execute the subsequent command upon the receiving of a signal of the type(s) specified. As soon as one program ends with signal 0 the griffin messenger is called with the termination command.

7.1.3) Running Griffin with NAMD

See tutorial

7.1.4) Running Griffin with GROMACS

Starting a Gromacs simulation involves a couple steps, shortly outlined here and described in detail under
www.gromacs.org/Documentation/How-tos/Steps_to_Perform_a_Simulation
Most of this is done when creating the .gfn files in section 2).

The only Gromacs specific thing in calling Griffin is that the flag
'-coordinate_format gromacs'
has to be set.

The key for starting Gromacs is the .tpr file which contains all necessary information to run the simulation. As explained in XXX it is crucial to have the line
' external_forces = ./wrap.sh'
in the .mdp when creating the .tpr file.
The wrap.sh contains the messenger call, which links Griffin with Gromacs.

Gromacs can be called by:
'mdrun –nt 1 –s EXPLICIT_FILE.tpr >& NAME.log'

where –nt is defining the number of threads (on a single node).

NOTE: the '-nt 1' is essential

7.1.5) Redirection

One of the strengths of Griffin is its ability to resolve conflicting surface forces. The straightforward idea to guide atoms towards the closest point on the protein surface will be what you want in most cases. Consider a lipid tail crossing a protein volume such that parts of it will be pulled in one direction, while other atoms are pulled in opposite direction. This would lead to static oscillations in which the lipid will not actually leave the protein volume.

This is resolved by the redirection of surface forces pointing in such direction. Using the geometric center of the lipid as reference point, it is possible to determine whether a force should be redirected or not.

An minimum angle of 110 degrees is default, other values can be set using the '-sforce_reset_min_angle VALUE' option.

When having lipids included in the implicit force field, the geometry of the complex may get very difficult and redirection may be counterproductive in that case. Visualizing the trajectory of the lipids will show whether this is the case or (most likely) not.

When the lipid box is not much larger than the implicit volume, redirection might become troublesome as well, because it relies on the geometric center of the lipids and periodic boundary conditions may shift atoms to the other side of the box.

7.2) Griffin with MPI support

Setting up a Griffin & NAMD simulation using PBS with MPI parallelization

A common scenario will be that you want to use e.g. 3 nodes with 8 cores in total for the simulation. While there is no problem calling 8 parallel NAMD jobs, according to the grid type this may not be possible for Griffin. That means you will have to request 8 cores per node from your batch system but tell MPI not to start more than e.g. 2 Griffin processes per node.

The submission script will have a line requesting your total nodes/processors

=> #PBS –l nodes=3:ppn=8

If you then wanted to start 3*2=6 Griffin daemons you will have to tell MPI which nodes to use (otherwise 6 processes will be started on one node and none on the other two).

The easiest way would probably the mpiexec flag '-npernode'
but this is not available on our system.
Due to an interaction of PBS and MPI (that might be related to our specific installation) the '--machinefile' flag does also not work.
Also due to this interaction our submission scripts need to have this line:

=> unset OMP_NUM_THREADS

NOTE: the following explanations are about the general mechanism and will be put into a timely order afterwards.

The jobs now can be controlled under PBS using the nodefile that contains all allocated resources. The nodefile is accessible by the environment variable $PBS_NODEFILE. Filtering the nodefile and redirecting the environmental variable enables to specify the exact job-distribution.

Requesting 3 nodes with 8 processors respectively, PBS will assign e.g. nodeX, nodeY and nodeZ to the job. PBS creates a nodefile containing 8 lines with 'nodeX' printed, 8 lines with 'nodeY' and 8 with 'nodeZ'. If one wants to start 2 jobs per node, the nodefile can be filtered and made accessible e.g. like this:

```
=> cat $PBS_NODEFILE > orig_nodefile.txt   (*)
(The next block is actually within call_gfn.sh)
=> cat orig_nodefile.txt |  awk '{ if( $0 != prev){ for( i=0; i<2; i++){ print $0;} prev = $0;}}' > gfn_nodefile.txt
=> export PBS_NODEFILE=gfn_nodefile.txt
```

This allows to call mpiexec without any further options:

=> mpiexec –np 6 ./daemon.sh >& daemon.log

If you wanted 4 instead of 2 processes per node you would have to change the i<2 to i<4 in the awk line and '-np 6' to '-np 12' in the call of mpiexec. The 'daemon.sh' contains the actual program call with all options (which can be pasted directly into the mpiexec call, of course).

After this general explanations we will consider now the actual order of calls after line (*).

In order to start both NAMD (requires seconds) and Griffin (minutes) simultaneously and make sure that none of them executes before the other one is ready, we start by calling a script that contains both some setup information for the simulation as well as the actual call of NAMD:

=> ./call_md.sh &

which needs to be sent to the background ('&') in order for the second process to be started.

Because starting up NAMD is rather fast and also the actual workflow is that NAMD calls Griffin the first step in the call_md.sh script is to start a process that simply waits for a message from Griffin that it is ready to be called. Calling:

=> griffin_messenger.exe -wait 12345

will block the script until a message is received on IPC (inter process communication) channel 12345. The channel has to match the 'griffin_messenger.exe –send 12345' message send by Griffin as confirmation (see below).

After Griffin confirmed that it is ready it will be told which bilayer system to read and the filenames of the coordinates and forces:

=> griffin_messenger.exe -setup carved_bilayer.gfn namdx.tmp namdf.tmp

and then NAMD will be called either without usage of the nodefile if all 8 cores shall be used for NAMD:

=> mpiexec –np 24 NAMD equil.inp > equil.log

or if only the 6 cores shall be used that are not occupied by Griffin:

=> cat original_nodefile.txt | awk '{ if($0 != prev){ for(i=0; i<6; i++){ print $0;} prev = $0;}}' > md_nodefile.txt
=> export PBS_NODEFILE=md_nodefile.txt
=> mpiexec –np 18 NAMD equil.inp > equil.log
As in the non-parallel case the input file (equil.inp) will contain information about the external force calculation called from NAMD, which is the actual linkage between NAMD and Griffin:

=> extForces yes
=> extForcesCommand wrap.sh
=> extCoordName namdx.tmp
=> extForceFilename namdf.tmp

Where wrap.sh contains:

=> PATH/griffin_messenger.exe -forces

Again the command can be pasted directly into the second line, but using a wrapper script allows e.g. copying the coordinate or force file before NAMD swallows them (if that's what you want to do). Another usage of the wrap.sh will be explained in the next section.

After calling './call_md.sh&' the next step is to start Griffin:

=> ./call_gfn.sh

which contains the lines listed after (*) and the actual call of Griffin. The difference to the single processor call is that the confirmation message has to be passed to the
'-system_call ./send.sh' flag, where send.sh contains:

=> griffin_messenger.exe -send 12345

NOTE: all scripts mentioned here have to be executable!

8) After crashes

If Griffin or MD crashes there is a good chance that there open message queues left behind. If they are empty they will cause no problems, but hanging messages may cause a following process relying on message queues to act erratically. Reversely, if a job dies unexpectedly there is a good chance that this was caused by an old termination message.

Therefore it is necessary to clean up message queues on machines a Griffin job crashed. To make this process most convenient there is a mode of the toolbox handling that:
'griffin_toolbox.exe –cleanup_queues NODE_1 ….'
to which you can pass as many nodes as you need.

If you prefer to do it 'by hand': ssh to the node, type 'ipcs' and for each line found under the last block belonging to message queues, you can remove the queue by 'ipcrm –q QUEUE_ID'.

Appendix A: the GFN format

The Griffin format (GFN) is column based, unlike the PDB format which is string position based.
This simplifies the reading/writing. Here is a list of the content of the individual columns:

atom ID
atom type
residue ID
residue name
molecule ID
position - x
position - y
position – z
mass
partial charge
van der Waals radius
Lennard Jones epsilon
temperature (not used yet)

Appendix B: how to use VMD to place exclusion volumes

One way of getting nicely matching objects:
Create a pdb file with a single atom, e.g. by
grep ALA name.pdb | head –n 1 > atom.pdb
Adjust coordinates to a meaningful initial guess (close to an atom where you want the object to be).
Then start vmd with:
vmd protein.pdb –f atom.pdb
Rotate/move the perspective showing the site of interest of the protein (fix it in the main menu – only for the protein).
Go to Graphics->Representations, pick atom.pdb as selected molecule, change drawing method to VDW, adjust radius (increase resolution).
Go to Mouse->Move->Atom and pick the single atom in VDW representation. After a few iterations of moving the dummy atom and adjusting the perspective of site, one may want to change the drawing method of the protein to MSMS, to see how the object matches in a space-filling representation.
Go to File->Save Coordinates, select atom.pdb, select ‘all’ atoms and save new coordinates to a new pdb file.

Appendix C: how to get Gromacs dump files

The steps to create a .tpr file:

i) create a topology file from your pdb:
pdb2gmx –f NAME.pdb –p NAME.top (-o NAME.gro)
in the interactive menu pick the force field you want the simulation to be based upon (the same as for creating the gmx file of the implicit protein when calculating the grid). The .gro file contain the box limits of the system in the last line.

Alternatively, one can create topology files ‘by hand’ by including single molecule topology files:
=> #include "charmm27.ff/forcefield.itp"
=> #include "charmm27.ff/tip3p.itp"
=> #include "pope.itp"
=> [system]
=> ; Name
=> test
=> [molecules]
=> ; Compound #mols
=> POPE 470
=> SOL 34972

This avoids an enormously big topology file. One can also circumvent problems that may appear doing it the first way. For large systems the former can also take quite a while.

ii) this step is necessary only for the explicit bilayer!
In your .mdp file which contains all options and parameters for the simulation add below:
=> userreal = 0
the following lines:
=> (empty line)
=> ; Call external script to add and call forces
=> external_forces = SCRIPT
where SCRIPT is then an executable file containing your commands for force calculation:
=> griffin_messenger.exe -forces

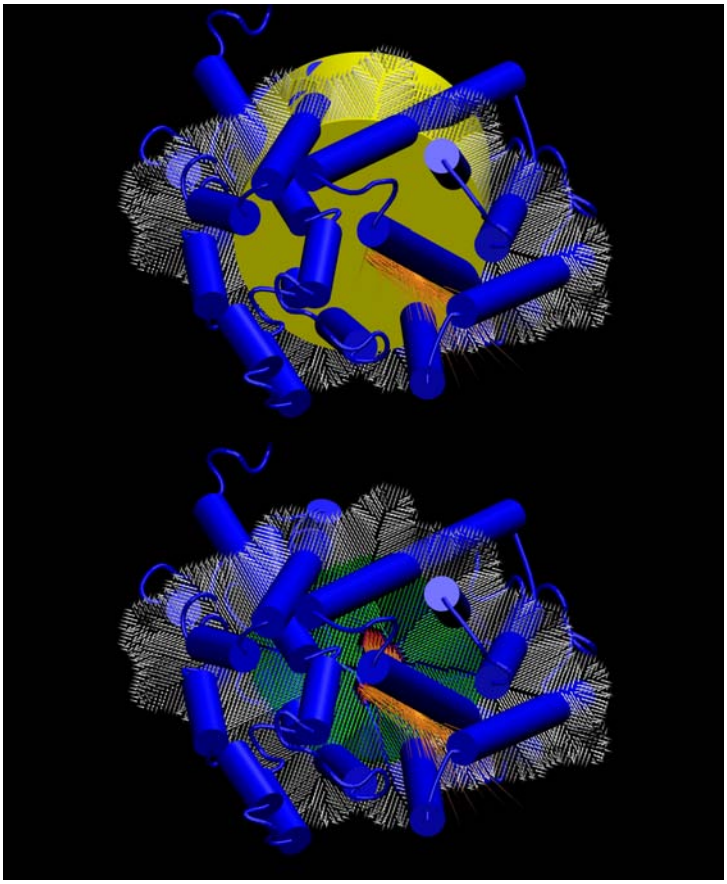
iii) having the .mdp file updated and the topology file created the next step is:
grompp –f NAME.mdp –c TOPOL.tpr -o NEW.tpr
which then allows to call from the PBS script (next section):
mdrun –s NEW.tpr

Binary topology files can be visualized by:
gmxdump -s NEW.tpr > New.gmx

See the Gromacs manual for more explanations.

Appendix D: visualizing the grid

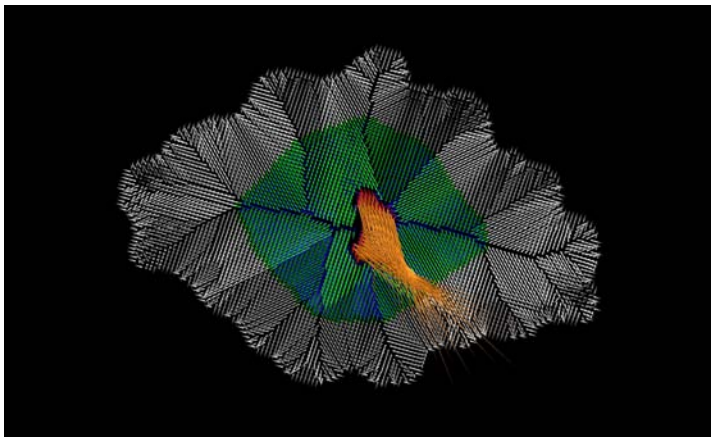
Again VMD is used for visualizing the forces stored in the grid.
There is a script: *GRIFFIN/scripts/visualize_grid.pl* that allows to extract VMD commands from a given grid. Open it and adjust the first block of variables.

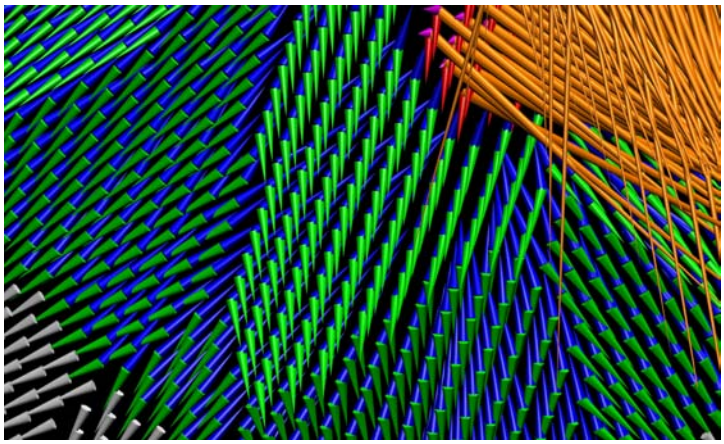


VMD can handle only <10k graphics commands within reasonable time. You will notice easily when the limit is close, the reading becomes exceedingly slow. Adding a low resolution value to the commands may improve this a bit.

Entire grids are too large to be visualized as a total. The script allows to kinds of selections being applied: spatial restrictions and type dependent restrictions. Furthermore, one cannot see much if using more than a single slice of grid points. Slices are created by selecting for one of the dimensions (x,y,z) an interval matching the gridpoint distance e.g. for a grid resolution of 0.5Å one can select:
$z_min = 0;
$z_max = 0.5;
while allowing x and y to cover the grid dimensions.

It is the idea to load one type of vectors e.g. surf forces as one VMD molecule object, which allows to color them differently. The next set of variables controls the type of forces being included:
$plot_const=”true” will plot surf forces
$plot_interaction=”true” will plot interaction forces not affected by geometric objects
$objects_are_defined=”true” hast to be set if geometric objects were used during grid calculation.





The images show a slice as desribed in the z direction through a grid calculated for the protein represented as cartoons (first and second image, in blue). A cylinder object was used to expel lipids from its core region (first picture, in yellow). The interaction forces around the protein were neglected by not setting: $plot_interaction = “true”. Surface forces that were not affected by the object are in white, surface forces acting on lipids are in green, surface forces acting on all other molecules are in blue. Interaction forces acting on non-lipids are orange and magenta. The last picture is a zoom into the same slice of the grid as shown on previous.

Appendix E: visualizing the forces acting on explicit atoms

The most immediate insight into forces calculated by Griffin can be extracted from the input coordinate file and the output force file.

The script:
GRIFFIN/scripts/extract_lines.pl INPUT_COORDS INPUT_FORCES OUTPUT_VMD [FIRST_LINE LAST_LINE]
will create a file with VMD commands visualizing the forces acting on the molecules. One or more blocks can be extracted by specifying first and last lines that are supposed to

One can trace one or more molecules throughout a simulation by
calling the script
GRIFFIN/scripts/wrap.sh
instead of calling the ‘griffin_messenger.exe –forces’
from the equil.inp file (for NAMD).
Using ‘g++ …. –DVMD_OUTPUT’ when compiling the daemon causes vmd commands to be written for all atoms at any step. Therefore this is only reasonable for the initial setup step and maybe a few simulation steps.
It is the idea to load one type of vectors e.g. surf forces as one VMD molecule object, which allows to color them differently and to exclusively follow the evolution of the forces acting on a specific atom.

Appendix F: monitoring memory

Because of its massive usage of RAM it might be useful to monitor the memory demand throughout the simulations.
The script: ‘GRIFFIN/scripts/monitor.pl PID’
will call every 10s a ‘pmap’ command and extract the total usage.
To obtain the PID (process ID) start the process and call ‘top’.
The interval can be adjusted in the script.

There is also a function Monitor() that allows to perform a similar measurement at defined spots in the code. That means rather than performing the measurement at fixed times it is performed when the program reaches a certain point in the code, allowing to trace steps that increase the memory dramatically.
In order for the Memory() function to have any effect the ‘-DMONITORING’ flag has to be provided to the compiler.
By default Memory() writes to the file ‘memory.txt’, calling Memory(“NAME.txt”) allows more complex analysis.

FAQs

1) Surface forces do not work:

did u calc all subgrids with the same flags/options? If one of the subgrids did not have the objects defined the combined grid will lack that ability.

2) Simulations just don't start:

most of the time caused by old 'hanging' message queues with messages (mostly from other users). Check with 'ipcs'.

3) Restarting NAMD/GRIFFIN jobs:

Create equil.2.inp:
set i 2
firsttimestep LAST_STEP_FROM_PREVIOUS_RUN

catdcd –num LASTRUN.dcd
=> N
catdcd -otype xyz –o NAME.xyz –first N –last N –s NAME.pdb –dcd LASTRUN.dcd

4) Lipids are being pulled inside the proteins volume instead of being expelled:

check that your lipid box is large enough. If a some atoms of a lipid are relocated to the other side of the box, the calculation of the geometric center is screwed. An easy fix should be to set:
-sforces_reset_min_angle 180'