Universität Dortmund
Physikalische Chemie IIa
Otto-Hahn Str. 6
D–44221 Dortmund, FRG

Volume,Temperature (V,T) Replica Exchange Molecular Dynamics
with the GROMACS 3.2.1 Simulation Package

RPMDRUN

A mini-HOWTO

Dietmar Paschek

October 1, 2004

Contents

1 Preliminaries

RPMDRUN is a Volume,Temperature (V,T)-Replica Exchange Molecular Dynamics (VTREMD) simulation program. It can be used to perform simulations of an arbitrarily shaped extended ensemble of states in the V,T-plane, such as:

- Sampling along a line of isochoric states.
- Sampling along a line of isobaric states (which have to be identified in a preceding independent simulation run).
- Sampling the entire V,T-plane using a grid of states.
- Multiplexed-Replica Exchange of one of the above.

RPMDRUN is based on the GROMACS 3.2.1 simulation package [1] `http://www.gromacs.org`. For further details concerning the forcefield, the simulation setup, boundary conditions, treatment of long range electrostatics,…*etc*, I would like to refer to the excellent original GROMACS Manual [2]. This little HOWTO just deals with aspects related to VTREMD simulations.

1.1 Contact

Dr. Dietmar Paschek
Physikalische Chemie II a
Universität Dortmund
Otto-Hahn Str. 6
D-44221 Dortmund, Germany
Tel: ++49-231-755-3938
Fax: ++49-231-755-3937
Email: dietmar.paschek@udo.edu

1.2 Disclaimer

1.3 Acknowledgment

RPMDRUN has been developed during a two month stay at Angel García's group (T-10 Division) at Los Alamos National Laboratory (LANL) spring 2004. I would like to thank Angel and his group for their hospitality. Financial support from the Center for Nonlinear Studies at LANL, as well as from the Deutsche Forschungsgemeinschaft (DFG FOR436) is gratefully acknowledged.

2 Installation

2.1 The Sources

I provide the sources and example applications for the RPMDRUN program on my website:

`http://ganter.chemie.uni-dortmund.de/~pas`

At the moment, however, the access is limited and will require a password. The password is available on request. RPMDRUN is based on GROMACS 3.2.1, which can be obtained from the GROMACS web site `http://www.gromacs.org`. Please install the GROMACS package first. Afterward unpack the RPMDRUN package in subdir `gromacs-3.2.1/src/`. Install the GROMACS package as a non-parallel version, otherwise interference with the VTREMD code might be caused. The new directory `replica` contains the RPMDRUN sources. Switch to that directory, and run "make". Eventually the `Makefile` has to be modified to match the requirements of your local installation (I am sorry for the inconvenience).

Please note: In order to run the example applications and any real world production simulations in an efficient manner, you will find it extremely helpful to have the MOSCITO simulation package, as well as the REPLICA simulation package, installed. Both are also available from my website. The packages contain certain programs and Perl-scripts which are used to prepare and monitor the simulation runs. The installation procedure for theses programs is described elsewhere [3]. Since REPLICA is based on MOSCITO, install MOSCITO first.

2.2 The Parallel Environment

A VTREMD simulation obviously requires a parallel environment. The RPMDRUN-code is written in MPI. Since I have made good experiences with the LAM MPI implementation (available from `http://www.lam-mpi.org/`), the given examples will assume using LAM. However, switching to any other MPI implementation will require only very minor modifications.

3 VTREMD-Background

3.1 Motivation

Parallel tempering is a rather new and elegant way to enhance sampling of states which cannot be sampled efficiently otherwise, such as systems exhibiting large barriers or glassy states [4]. Moreover, parallel tempering has also become a useful strategy for bio-molecular simulations (in the form of replica exchange molecular dynamics), e.g. to determine the folding transition of proteins [5]. The methodological advantage is that a protein can explore a different number of possible folding pathways at the same time and can overcome barriers or trapped states at higher temperatures. In general, the sampling is enhanced by a simultaneous consideration of different states and by introduction of temperature swapping moves [6, 7].

However, for the biophysics of many systems and living organisms the pressure is also a relevant parameter. So, many proteins are stable and stay functional only in a limited temperature *and* pressure interval [8]. Therefore it might be fruitful to consider also moves where density changes are involved, thus representing an extended ensemble of states covering a large temperature and pressure range. A successful demonstration involving the reversible temperature and pressure denaturation of a protein fragment has been recently given by us [9].

3.2 VTREMD-Algorithm

In this section a the acceptance rule for parallel tempering state swapping moves involving volumes changes is derived. Let us consider an extended ensemble of n states. Each state i represents a canonical ensemble characterized by a certain temperature T_i and volume V_i. The partition function for the extended ensemble is the product of all individual N, V_i, T_i ensembles

$$\begin{aligned} Q_{\text{ext.}} &= \prod_{i=1}^{n} Q(N, V_i, T_i) \\ &= \prod_{i=1}^{n} \frac{V_i^N}{\Lambda_i^{3N} N!} \int d\vec{s}_i^{\,N} \exp\left[-\beta_i U(\vec{s}_i^{\,N}; L_i)\right] \end{aligned} \tag{1}$$

where $\beta_i = 1/k_B T_i$ and $\vec{s}_N = L^{-1}\,\vec{r}_N$ (with $L = V^{1/3}$ being the length of a hypothetical cubic box) represent the scaled coordinates of particle N. $\vec{s}_i^{\,N}$ represents the set of coordinates of the entire N-particle system belonging to state i. $U(\vec{s}_i^{\,N}; L_i)$ and V_i denote the potential energy and volume of system i, respectively. The probability density $\mathfrak{N}$ to obtain a system $\vec{s}_i^{\,N}$ at a certain state (T_i, V_i) is given by

$$\mathfrak{N}(\vec{s}_i^{\,N}, V_i, \beta_i) = \frac{\exp\left[-\beta_i U(\vec{s}_i^{\,N}; L_i)\right]}{\int d\vec{s}_i^{\,N} \exp\left[-\beta_i U(\vec{s}_i^{\,N}; L_i)\right]} \,. \tag{2}$$

To sample this extended ensemble it is in principle sufficient to to perform NVT simulations of all individual ensembles separately. However, it is also possible (and advantageous) to introduce Monte Carlo moves swapping two states. Let us consider two states i and j, represented by (N, β_i, V_i) and (N, β_j, V_j). The acceptance rule for a swap between two ensembles i and j follows from the condition of detailed balance

$$\begin{aligned} &\mathfrak{N}(\vec{s}_i^{\,N}, V_i, \beta_i)\mathfrak{N}(\vec{s}_j^{\,N}, V_j, \beta_j) \\ &\times\alpha\left[(\vec{s}_i^{\,N}, V_i, \beta_i), (\vec{s}_j^{\,N}, V_j, \beta_j) \to (\vec{s}_j^{\,N}, V_i, \beta_i), (\vec{s}_i^{\,N}, V_j, \beta_j)\right] \\ &\times\text{acc}\left[(\vec{s}_i^{\,N}, V_i, \beta_i), (\vec{s}_j^{\,N}, V_j, \beta_j) \to (\vec{s}_j^{\,N}, V_i, \beta_i), (\vec{s}_i^{\,N}, V_j, \beta_j)\right] \\ &= \\ &\mathfrak{N}(\vec{s}_i^{\,N}, V_j, \beta_j)\mathfrak{N}(\vec{s}_j^{\,N}, V_i, \beta_i) \\ &\times\alpha\left[(\vec{s}_i^{\,N}, V_j, \beta_j), (\vec{s}_j^{\,N}, V_i, \beta_i) \to (\vec{s}_j^{\,N}, V_j, \beta_j), (\vec{s}_i^{\,N}, V_i, \beta_i)\right] \\ &\times\text{acc}\left[(\vec{s}_i^{\,N}, V_j, \beta_j), (\vec{s}_j^{\,N}, V_i, \beta_i) \to (\vec{s}_j^{\,N}, V_j, \beta_j), (\vec{s}_i^{\,N}, V_i, \beta_i)\right] \,. \end{aligned} \tag{3}$$

where $\alpha\,[\ldots]$ represents the *a priori* probability for a given state-swapping move and acc $[\ldots]$ denotes the acceptance ratio. If we construct the Monte Carlo state-swapping move in such a way that the *a priori* probability α [. . .] is equal for all conditions, we obtain as acceptance rule

$$\begin{aligned}
&\frac{\mathrm{acc}\left[(\vec{s}_i^N, V_i, \beta_i), (\vec{s}_j^N, V_j, \beta_j) \to (\vec{s}_j^N, V_i, \beta_i), (\vec{s}_i^N, V_j, \beta_j)\right]}{\mathrm{acc}\left[(\vec{s}_i^N, V_j, \beta_j), (\vec{s}_j^N, V_i, \beta_i) \to (\vec{s}_j^N, V_j, \beta_j), (\vec{s}_i^N, V_i, \beta_i)\right]} \\
&= \\
&\frac{\exp\left[-\beta_i U(\vec{s}_j^N; L_i) - \beta_j U(\vec{s}_i^N; L_j)\right]}{\exp\left[-\beta_i U(\vec{s}_i^N; L_i) - \beta_j U(\vec{s}_j^N; L_j)\right]} \ .
\end{aligned} \tag{4}$$

So the acceptance probability for a state swapping move finally turns up to be

$$\begin{aligned}
P_{acc} &= \min\{1, \exp[\beta_i \left(U(\vec{s}_i^N; L_i) - U(\vec{s}_j^N; L_i)\right) \\
&\quad + \beta_j \left(U(\vec{s}_j^N; L_j) - U(\vec{s}_i^N; L_j)\right)]\} \ .
\end{aligned}$$

The volume change is usually performed in such a way that only intermolecular distances are changing, hence only the center of mass coordinates are scaled. Since we obtain the pressure during the molecular dynamics (MD) simulation routinely and given that we consider only small volume changes, we approximate the energy

$$U(\vec{s}_j^N; L_i) \approx U(\vec{s}_j^N; L_j) - (P_j - \frac{M}{\beta_j' V_j}) \times (V_i - V_j) \tag{5}$$

and

$$U(\vec{s}_i^N; L_j) \approx U(\vec{s}_i^N; L_i) - (P_i - \frac{M}{\beta_i' V_i}) \times (V_j - V_i) \ , \tag{6}$$

where M represents the number of the *molecules* in the simulation box. Here β' represents the *instantaneous* temperature, whereas β denotes the average temperature characterizing the states i and j.

In order to provide equal *a priori* probabilities and to thus fulfill the detailed balance condition, the decision whether a state swapping move, or an MD move is executed, has to be chosen at random.

There is a certain probability that a state swapping move might be reversed in the successive step. Therefore it is important to adjust the energies $U(\vec{s}_j^N; L_i)$ and $U(\vec{s}_i^N; L_j)$ after each successful volume exchange to ensure a proper evaluation of the acceptance probabilities.

4 Running a VTREMD Simulation

The `rpmdrun` program is essentially a VTREMD version of GROMACS' original `mdrun`. However, the VTREMD capabilities require some modifications and extensions. First, there are two additional input files: `replica.gmx` and `replica.mdx`. `replica.gmx` is used to setup the extended ensemble of states. The other one, `replica.mdx`, incorporates an additional concept into GROMACS: Sites that are belonging to a certain molecule (not a residue). Knowing about molecules is important for density scaling, since only intermolecular distances are supposed to change. In addition, for each replica there have to be separate input-

and output-files. Therefore RPMDRUN identifies each replica by its particular replica-*id*. Keep in mind that `rpmdrun` reads from and writes to files that are located in present working directory. Please note, that when using a large number of replicas, the number of files in the working directory might exceed several thousands or even ten-thousands. Please make sure that your file-system supports such a large number of files in one directory. Of course, a huge amount of free disk-space should be available as well.

4.1 The Extended Ensemble and the Replica-Exchange Connectivity Table: replica.gmx and replica.gmx_out

The extended ensemble of considered states and the replica-exchange connectivity table are contained in `replica.gmx` (see Figure 1). The states are defined by their density and reference temperature and the state-exchange table identifies replicas between which state-exchange-moves shall be attempted.

The first line in `replica.gmx` defines the probability to attempt a state-swapping move. Using a value 0.002, on average after every 500'th MD-step, a state-swapping move will be attempted. However, the number of MD-timesteps between two state-swapping attempts might vary, since the decision, whether a state-swapping move is attempted or a MD move is executed, is chosen at random. The value specified in the following line defines how often a state-swapping attempt will be repeated for each try. Usually, one would set this value to one, meaning that there will be only one try per state-swapping move. However, when dealing with a very large number of replicas, it might be advantageous to use a larger number here (in order to try several subsequents attempts), while using a smaller value for the exchange probability. This feature might be also be of use when realizing a multiplexed REMD state-topology [10]. In the next line, the number of replicas n used in the actual simulation is given. The following n lines define the actual replicas by their replica-*id*, their initial temperature (in K) and their density (in $\mathrm{g\,cm^{-3}}$). What follows is the replica connectivity-table.

```
0.002
1
6
0  300.0  1.000
1  305.0  1.000
2  310.0  1.000
3  315.0  1.000
4  320.0  1.000
5  325.0  1.000
5
0 1  1.0
1 2  1.0
2 3  1.0
3 4  1.0
4 5  1.0
```

Figure 1: Example `replica.gmx`–file for an (unrealistically small) extended ensemble of 6 (isochoric) states. For simplicity, the temperature difference between all states has been chosen to be constant. In a realistic simulation one would adjust the temperature differences to maintain a certain state-swapping acceptance ratio.

```
0.002000
1
6
0   300.00   1.0000
1   305.00   1.0000
2   310.00   1.0000
3   320.00   1.0000
4   315.00   1.0000
5   325.00   1.0000
5
0 1 1.000000
1 2 1.000000
2 4 1.000000
4 3 1.000000
3 5 1.000000
```

Figure 2: Example `replica.gmx_out`–file for the extended ensemble of 6 (isochoric) states after one successful state-swapping move. Please note, that 'replica 3' has now a temperature of 320 K, whereas 'replica 4' is at 315 K. Please note also the change in the replica-connectivity table. `replica.gmx_out` has to be renamed to `replica.gmx` in order to be able to continue the simulation successfully.

```
#!/bin/sh
cat >REPLICA.def<<EOF
ATTEMPTS  1
PROB  0.002
TEMP 300.0  # 1
TEMP 305.0  # 2
TEMP 310.0  # 3
TEMP 315.0  # 4
TEMP 320.0  # 5
TEMP 325.0  # 6
RHO 1.000
EOF
r_mkgmx.pl < REPLICA.def > replica.gmx
```

Figure 3: Shown is a small shell-script, creating the `replica.gmx`-file given in Figure 1.

Suppose there are m connectivities, consequently this number m is specified by the first line of the connectivity-table. The following data are the m pairs of replica-*id*s between which state-swapping shall be attempted. The value in the third row specifies an additional execution probability for such an attempt. The idea is to provide a possibility to balance between different kinds of moves such as temperature- and density-swaps. At the moment, however, this feature is disabled.

During the course of the replica simulation, eventually state exchanges between replica occur (it hopefully happens, since state-exchange is what REMD is all about) and therefore the replica-definition and the replica connectivity-table change. In order to be able to continue a simulation run, the final status of the replica-simulation is written to a new file called `replica.gmx_out` (see Figure 2) when closing down the actual simulation run.

Imagine you wish to run a large replica simulation, sampling a large patch of the entire V,T-plane of a liquid solution. A lot of states have to be defined, and particularly setting up the replica-connectivity map by hand might become a tiresome task. Henceforth, a Perl-script `r_mkgmx.pl` is provided, used to create an initial `replica.gmx`-file from a minimal set of

```
#!/bin/sh
cat >REPLICA.def<<EOF
ATTEMPTS  1
PROB  0.002
EOF
r_tempscale -t0 300.0 -dt0 5.0 -dt1 6.0 -n 6 >> REPLICA.def
r_rhoscale -r0  1.0 -dr0 0.01 -dr1 0.015 -n 6 >> REPLICA.def
r_mkgmx.pl < REPLICA.def > replica.gmx
```

Figure 4: Shown is a small shell-script creating the `replica.gmx`-file defining a six (temperature) times six (density) matrix of states with nearest neighbor state-swapping. Temperature sequence: 300.0, 305.0, 310.2, 315.6, 321.2, 327.0. Density sequence: 1.000, 1.010, 1.021, 1.033, 1.046, 1.060.

information defined by a set of keywords contained in a file, typically (but not necessarily) called `REPLICA.def`.

```
r_mkgmx.pl < REPLICA.def > replica.gmx
```

An example is given in Figure 3. `r_mkgmx.pl` always defines a grid of V,T-states (or an isochoric, or isothermal line of states) defined by their temperature (keyword: TEMP) and density (keyword: RHO). The state-swapping attempt probability is given by the PROB keyword and the number of attempts by ATTEMPTS. Please note, that this procedure can, of course, not be employed to attempt sampling along an isochoric line or for Multiplexed-Replica exchange. Perhaps, more elaborate scripts addressing the above mentioned problems, might appear soon.

In order to make creating an initial temperature- and density-tiling for `REPLICA.def` a little bit easier, there are two more Perl-scripts available, generating temperature and density sequences suitable for `REPLICA.def`: `r_rtempscale.pl` and `r_rhoscale.pl` (an example is given in Figure 4).

```
r_tempscale.pl -t0 300.0 -dt0 5.0 -dt1 6.0 -n 6
r_rhoscale.pl -r0  1.0 -dr0 0.01 -dr1 0.015 -n 6
```

Here `-n` defines the number of created temperatures/densities, while `-t0` and `-r0` define the first temperature and density values in the sequence. `-dt0` and `-dr0` define the (starting) temperature/density-increment, whereas `-dt1` and `-dr1` define the final temperature/density-increment. The increment is supposed to change linearly. However, these scripts are probably used just as a first guess, or to prepare a trial simulation. A more elaborate temperature-tiling will require the knowledge of the potential energy-distribution of the particular system as a function of temperature. Typically, the temperature tiling is chosen to ensure sufficiently overlapping energy distributions. Usually this is done in such a way to maintain a certain acceptance ratio of about, say, 20 per cent.

4.2 The Site-Molecule Definition: replica.mdx

`replica.mdx` introduces an additional concept to GROMACS: Sites that are belonging to a certain molecule (not residue). Knowing about molecules is important for density scaling, since only intermolecular distances are supposed to change. The chosen file-format is actually extremely simple. The first line contains the number of *atoms* in the actual simulation, whereas the following line denotes the number of *molecules*. For each atom in the simulation there is a new line and each line contains an integer number specifying the actual molecule to which the atom belongs. Creating such a file by hand might be a tedious and error-prone task. Therefore a small Perl-script `r_mkmdx.pl` is provided:

```
r_mkmdx.pl -mols0 1 -sites0 156 -mols1 2644 -sites1 3 > replica.mdx
```

The example given above represents a simulation of a single protein (with 156 sites) dissolved by 2644 water molecules (each represented by three sites). Additional molecules might be introduced by further `-sites2` and `-mols2`, . . . *etc.*, options. The specifications given above have to be tailored to every new simulation. However, the `rpmdrun`-program will check for inconsistencies with respect to the definitions given in the topology-file.

4.3 Addressing Input-/Output-Files by their Replica-*id*

For each replica there have to be separate input and output-files. RPMDRUN reads and writes data directly to the present working directory. Therefore, to avoid file-name conflicts, each replica adds automatically six characters to the beginning of each file-name: A 'R', followed by the replica-*id* in a *%00004d*-format, followed by a '_'. The rest of the file names can (and should) be defined when calling `rpmdrun`. An example is given in Figure 5. The input-/output-files can be discussed just briefly here. For more detailed information, please take a look at the GROMACS manual [2]. The thermodynamical properties for each replica are stored in the `.edr`-files, and the system trajectory (atom-coordinates as a function of time) are stored in the `.xtc`-files. [1] The GROMACS program `g_energy` might be used to extract

[1] `.edr`- and `.xtc`-files are based the XDRF-library [11], which stores molecular structure data in a very efficient manner by making use of correlations between the atoms and by limiting the accuracy to about 0.5 pm. An entire coordinate triple (x,y,z) consumes thus just about 3.5 to 4 bytes on average. The xtc-format is supported by a number of programs, such as the recent versions of VMD [12] and MOSCITO [3]. The binary data in

```
>mpirun -c 6 rpmdrun -x sim1.xtc -o sim1.trr -c sim1.gro -e sim1.edr -g sim1.log
>ls R*.sim1.*
R0000_sim1.edr
R0000_sim1.edr_pte
R0000_sim1.gro
R0000_sim1.log
R0000_sim1.trr
R0000_sim1.xtc
R0000_sim1.xtc_pte
R0000_sim1out.mdp
R0001_sim1.edr
R0001_sim1.edr_pte
R0001_sim1.gro
R0001_sim1.log
R0001_sim1.trr
R0001_sim1.xtc
R0001_sim1.xtc_pte
  .
  .
  .
R0005_sim1.edr
R0005_sim1.edr_pte
R0005_sim1.gro
R0005_sim1.log
R0005_sim1.trr
R0005_sim1.xtc
R0005_sim1.xtc_pte
```

Figure 5: Example of a 6-replica simulation run as obtained by calling `rpmdrun`. The output-filenames are specified at the command-line. The `rpmdrun`-STDOUT-output has been discarded for clarity.

```
>grompp -f SIM.mdp -p SIM.top -c R0000_sim0.gro -po R0000_sim1.mdp_out -o R0000_STRT.tpr
>grompp -f SIM.mdp -p SIM.top -c R0001_sim0.gro -po R0001_sim1.mdp_out -o R0001_STRT.tpr
>grompp -f SIM.mdp -p SIM.top -c R0002_sim0.gro -po R0002_sim1.mdp_out -o R0002_STRT.tpr
>grompp -f SIM.mdp -p SIM.top -c R0003_sim0.gro -po R0003_sim1.mdp_out -o R0003_STRT.tpr
>grompp -f SIM.mdp -p SIM.top -c R0004_sim0.gro -po R0004_sim1.mdp_out -o R0004_STRT.tpr
>grompp -f SIM.mdp -p SIM.top -c R0005_sim0.gro -po R0005_sim1.mdp_out -o R0005_STRT.tpr
>ls R*.tpr
R0000_STRT.tpr
R0001_STRT.tpr
R0002_STRT.tpr
R0003_STRT.tpr
R0004_STRT.tpr
R0005_STRT.tpr
```

Figure 6: Each replica requires its own particular startup-file (`.tpr`-file) which contains the definition for an entire (replica) simulation run, including topology/forcefield information, initial coordinates and velocities, as well as the simulation run setup. The binary `.tpr`-file has to be created by using the conventional GROMACS preprocessor `grompp`. `grompp` uses three different input sources: (1) The `.mdp`-file, containing the run-setup. (2) The `.gro`-file (the native Gromos87 format), holding the coordinates and velocities of the final configuration of a preceding simulation run. (3) The `.top`-file, containing the topology and forcefield. The startup-file for each replica-*id* requires (!) the fixed format name: `R####_STRT.tpr`. The `grompp`-STDOUT-output has been discarded for clarity.

data from an `.edr`-file. The `.gro`- and `.trr`-files contain the atom coordinates and velocities for the final configuration. The `.gro`-format is the generic Gromos87 ASCII-format, whereas the `.trr`-file has a GROMACS specific binary format. Usually ,I prefer to restart a simulation from a `.gro`-file, since all counters etc. are then automatically set to zero. Finally, the `.log`-file contains information on the simulation run, such as averages and timings in a human readable form.

Whereas the names for the output-files can be specified at the command line, the name of the input-file is fixed. Each replica reads the configuration of the entire simulation run, including the run-setup, the forcefield and molecule-topologies, as well as the initial coordinates and velocities from one particular binary input-file called `R####_STRT.tpr` (it is not a typo, it is really STRT). Therefore, before being able to start the VTREMD-run, such an input-file has to be created for each replica. For this purpose, GROMACS provides the GROMACS-preprocessor `grompp`. `grompp` is intermingling three different sources by processing a *topology* `.top`-file, containing the forcefield, a restart configuration (typically contained in a `.gro`-file), and a `.mdp`-file, holding the setup (time-step, Ewald-setup, output-frequency) for the entire MD-run. Since all replica-runs should, of course, be configured identically, there should be just one `.top`- and `.mdp`-file. Figure 6 exemplifies the input-file preparation for a 6-replica simulation run. `grompp` will produce considerable output, and eventually warnings and error-messages. In addition, an extensively detailed, processed version of the `.mdp`-file is written to the file specified by the `-po`-option.

4.4 Replica-State Trajectory-Protocol Files: .xtc_pte and .edr_pte

In order to provide a protocol for the VTREMD-simulation and to be able to assign the stored configurations and thermodynamical/energetical data of each of the individual replicas to their corresponding states, `rpmdrun` always creates two additional so-called ’PTE’-files. (PTE

XDRF-files is stored in a hardware independent way and allows interchange of data between big- and little-endian machines.

```
 1000 296.42111206 -106067.67187500 20302.11523438 -56.05250931  T 300.0 R 0.960
 2000 304.36569214 -106193.09375000 20846.24609375 -39.49655533  T 300.0 R 0.960
 3000 295.61325073 -105746.89843750 20246.78320312 -74.47006989  T 300.0 R 0.960
 4000 295.22518921 -106338.10156250 20220.20507812 -44.83750534  T 300.0 R 0.980
 5000 296.44931030 -106217.46093750 20304.04687500 -2.97017622  T 300.0 R 0.980
 6000 304.13015747 -106413.78906250 20830.11328125 -13.73261642  T 300.0 R 0.980
 7000 295.76913452 -106426.27343750 20257.46093750 -47.76762390  T 300.0 R 0.980
 8000 296.79498291 -107209.78125000 20327.72070312 -55.09703445  T 300.0 R 0.980
 9000 299.71136475 -106004.05468750 20527.46679688 -15.42340279  T 300.0 R 0.980
10000 302.69570923 -106578.67187500 20731.86718750 -12.33014297  T 300.0 R 0.980
11000 298.35388184 -106640.51562500 20434.49218750 3.01580453  T 300.0 R 0.980
12000 303.08145142 -106655.57812500 20758.28710938 -17.05986977  T 300.0 R 0.980
13000 305.81127930 -105773.40625000 20945.25390625 10.96167374  T 304.5 R 0.980
13000 305.81127930 -105773.40625000 20945.25390625 10.96167374  T 304.5 R 0.980
14000 301.16677856 -106106.89062500 20627.15039062 16.52543259  T 304.5 R 0.980
15000 303.30520630 -106170.78125000 20773.61132812 -17.04383850  T 304.5 R 0.980
16000 301.62588501 -105919.75781250 20658.59375000 -15.34104824  T 304.5 R 0.980
17000 304.10519409 -106192.64062500 20828.40429688 24.53301430  T 304.5 R 0.980
18000 303.90145874 -106735.03125000 20814.44921875 41.20051193  T 304.5 R 1.000
19000 314.30288696 -105446.15625000 21526.85156250 20.68351364  T 313.7 R 1.000
20000 314.05596924 -105031.68750000 21509.93945312 -4.26713705  T 313.7 R 1.000
21000 314.82305908 -106141.62500000 21562.47851562 21.09751892  T 313.7 R 1.000
22000 316.67568970 -105588.36718750 21689.36718750 68.60745239  T 313.7 R 1.000
```

Figure 7: Excerpt from a PTE-file. The first column specifies the actual time-step. The second column holds the instantaneous temperature (in K). The third column specifies the configurational (potential) energy (in kJ mol^{-1}). The fourth column contains the kinetic energy (in kJ mol^{-1}) and the fifth column contains the pressure (in MPa). The last columns define actually the state by its target temperature (in K) and density (in g cm^{-3}) with a preceding 'T ' and 'R ', respectively.

stands for pressure, temperature, energy). For each configuration written to the `.xtc`- or `.edr`-file the current state is written to the corresponding PTE-file. thus that the data in the both files can be reassigned to the proper state. The PTE-filename is alway specified by adding the sub-extension '`_pte`' to filenames of the corresponding `.xtc` and `.edr`-files. An example PTE-file is shown in Figure 7. Further detailed information on the exact file-format and the contained data is given in the corresponding figure-caption.

4.5 Generating Start Configurations for All Replicas

Creating all the start-configurations by hand might be an error-prone procedure when a large number of replicas is involved and the density varies. Therefore a small Perl-script `r_mkgro.pl` is provided

```
r_mkgro.pl -gro startconf.gro < REPLICA.def
```

that creates a start configuration with the correct density for each replica. The information is

```
>r_mkgro.pl -gro startconf.gro < REPLICA.def
>ls R*.START.*
R0000_START.gro
R0001_START.gro
R0002_START.gro
R0003_START.gro
R0004_START.gro
R0005_START.gro
```

Figure 8: `r_mkgro.pl` prepares the start configuration for all replica according to the dfinition given in `REPLICA.def`.

read from a corresponding `REPLICA.def`-file. In addition, a seed-configuration is needed, which is specified by the `-gro`-option. In order to provide a correct density scaling, the seed-configuration should exhibit a density of exactly 1.0 $\mathrm{g\,cm^{-3}}$. The created starting configurations are written to files `R####_START.gro`. Moreover, `r_mkgro.pl` writes a protocol, containing replica-*id*, temperature and density to STDOUT. An example application is given in Figure 8.

4.6 Recommendations and Suggestions for .mdp-Files When Using RPMDRUN

An example `.mdp`-file specifying the MD-setup for a typical VTREMD-simulation used by us is given in Figure 9. I would like to emphasize on some obvious and some probably not so obvious points:

1. Unless you perform just a short equilibration simulation, coordinates and energies should be written only to `.xtc`- and `.edr`-files, since just for them the corresponding PTE-files are available. Please, set nstxout to zero and use in any case the more compact xtc-format (nstxtcout).

2. A timesteps of 2.0 fs are usually feasible for the considered typical temperature range between 280 K to 550 K. When flexible molecules with fixed-hydrogens are involved, it

```
;       Input file
;
title               =  Yo                     ; a string
cpp                 =  /lib/cpp               ; c-preprocessor
dt                  =  0.002                  ; time step
nsteps              =  250000                 ; number of steps
nstcomm             =  1                      ; reset c.o.m. motion
nstxout             =  0                      ; write no coords
nstxtcout           =  1000                   ; write compat coords to xtc-file
nstvout             =  0                      ; write no velocities
nstlog              =  50000                  ; print to logfile
nstenergy           =  100                    ; print energies
nstlist             =  6                      ; update pairlist
ns_type             =  grid                   ; pairlist method
coulombtype         =  pme                    ; use particle-mesh Ewald
fourier_nx          = 36                      ; use a fixed rec.-space lattice
fourier_ny          = 36
fourier_nz          = 36
pmeorder            =  4
ewald_rtol          =  1.0e-5
DispCorr            =  EnerPres
constraint_algorithm=  shake
shake_tol           =  1.0e-4
rlist               =  0.9                     ; cut-off for ns
rvdw                =  0.9                     ; cut-off for vdw
rcoulomb            =  0.9                     ; cut-off for coulomb
Tcoupl              =  nose-hoover             ; temperature coupling: Use Nose-Hoover
ref_t               =  300.0
tc-grps             =  System
tau_t               =  0.5
Pcoupl              = berendsen                ; Enable pressure bath swith although not used
Pcoupltype          =  isotropic               ; pressure geometry is necessary
tau_p               =  4.0                     ; p-coupoling time is necessary
compressibility     =  4.5e-5                  ;
ref_p               =  1.0                     ;
gen_vel             =  no                      ; generate initial velocities
gen_temp            =  300                     ; initial temperature
gen_seed            =  1993                    ; random seed
```

Figure 9: An example `.mdp`-file as used recently for a VTREMD simulation.

is sometimes useful to consider the `-heavyh`-option when preparing the protein topology.

3. When density-swaps are possible (VTREMD), choose a setup for the PME-reciprocal lattice that achieves an appropriate accuracy for the whole considered density range. Please note: Define the PME-lattice explicitly by the `fourier_nx...` keywords. Otherwise `grompp` might probably choose different lattice setups just based on the box-dimensions of the initial configurations. It is certainly more consistent to provide fully identical conditions for all replicas.

4. The pressure-coupling MUST be enabled (!!!!) although each replica simulation is in fact conducted in the NVT-ensemble. However, it is important for GROMACS to know, that box sizes (e.g. when replicas are exchanging their states) might vary.

5. I strongly recommend the use of the Nosé-Hoover thermostat, since it generates a proper canonical ensemble of states. The VTREMD-algorithm is basically derived assuming a canonical distribution. Moreover, the Berendsen-algorithm tends to introduce a coupling-time dependence to the width of the configurational energy distribution. This might be particularly inconvenient when setting up a VTREMD-simulation, and might influence the physics as well.

5 An Example Application: The AK-peptide

This part provides some details on the preconfigured simulation runs that are available from my website http://ganter.chemie.uni-dortmund.de/~pas/rpmdrun_demo.tgz. In this section, however, I will focus on just one particular application: the REMD simulation of the helical 20-residue AK-peptide in an explicit solvent. More information on the AK-peptide can be found in a recent paper by Gnanakaran et al. [13].

5.1 Creating the Forcefield and the Initial Configuration

The subdir `RPMDRUN_DEMO/AK_DEMO/build` contains actually three shell scripts, shown in Figures 10 and 11. The scripts can be used to setup the topology-files and initial configurations of the solvated α-helical AK-peptide.

`mkpdb.sh` uses the `tleap` program (which is part of the AMBER simulation package and not free available) to generate a pdf-file for the 20-residue AK-peptide in an α-helical conformation. In case that `tleap` is not available, a copy of the pdb-file can be found in

```
#!/bin/sh
LEAPROOT=~/LEAP
cat > leaprc <<EOF
#
parm94 = loadAmberParams "parm94.dat"
loadOff all_amino94.lib
loadOff all_aminont94.lib
loadOff all_aminoct94.lib
#
U=sequence { ALA ALA LYS ALA ALA }
AKpeptide=sequence { ALA ALA U U U ALA ALA TYR  }
impose AKpeptide { { 1 999 } } { { N CA C N -40.0 } { C N CA C -60.0 } }
savepdb AKpeptide AKpeptide.pdb
quit
EOF
tleap
```

Figure 10: `mkpdb.sh` uses Leap (which is part of the AMBER simulation package) to create a pdb-file for the 20 residue AK peptide with sequence $AA-(AAKAA)_3AAY$ in an α-helical conformation.

```
#!/bin/sh
echo 1 1 | pdb2gmx -ff G43a1 -ignh -ter -water spc -n akp.ndx \
           -o akp.gro -p akpaq.top -i akp.itp -f AKpeptide.pdb -q akp.pdb
editconf -f akp.gro -o akpc.gro -box 4.35 4.35 4.35
genbox -box 4.35 4.35 4.35 -cp akpc.gro -cs spc216.gro -p akpaq.top -o akpaq.gro
```

```
#!/bin/sh
echo 1 1 | pdb2gmx -ff oplsaa -ignh -ter -water tip3p -n akp.ndx \
           -o akp.gro -p akpaq.top -i akp.itp -f AKpeptide.pdb -q akp.pdb
editconf -f akp.gro -o akpc.gro -box 4.35 4.35 4.35
genbox -box 4.35 4.35 4.35 -cp akpc.gro -cs spc216.gro -p akpaq.top -o akpaq.gro
```

Figure 11: Top: `mktop_g96.sh`. Bottom: `mktop_opls.sh`. Both shell scripts generate topology files for the solvated AK-peptide located in a 4.35 nm simulation box. Roughly 2650 water molecules are represent the solvent phase. Top: The peptide is represented by Gromos96-Forcefield using the SPC-model for water. Bottom: The peptide is represented by OPLS(AA)-Forcefield using the TIP3P-model for water.

RPMDRUN_DEMO/AK_DEMO/pdb. The file AKpeptide.pdb is holding the entire peptide configuration.

In order to prepare the topology-files and the start-configuration, preconfigured scripts are available for either generating an OPLS all-atom model of the AK-peptide solvated in TIP3P water, or an Gromos96-model of the peptide using the SPC-model as solvent. The two shell-scripts are shown in Figure 11. Detailed information on the individual programs is given in the GROMACS manual [2]. Basically two files are created: akpaq.top contains the complete peptide/solvent topology-information, whereas akpag.gro holds the initial structure of the solvated protein in the generic Gromos87 format. akpag.gro can be loaded into VMD [12] to visualize the configuration. In addition, the file ak.gro contains just the peptide configuration without any solvent.

5.2 Running an Entire REMD-Simulation

Before attempting to run one the shell-scripts, make sure you have installed all the MOSCITO, REPLICA, GROMACS and RPMDRUN programs and that the executables are available in your actual PATH. Before starting the shell-scripts in the RPMDRUN_DEMO/AK_DEMO/run subdirectory, add the names of the actual nodes, you wish to run the programs on, to run/machine.pas. Remember to log to one of the nodes before starting any jobs. A complete entire simulation run will be executed by calling RUNSIM.sh. So, in principle you can just start 'RUNSIM.sh', contained in this directory. Nevertheless, I would encourage you to execute the commands listed in RUNSIM.sh step by step in order to see what the shell-scripts and programs are doing.

5.2.1 Defining States and Preparing the Simulation

The REMD simulation run is actually prepared by calling two shell-scripts: mk_def.sh and init_gmx.sh. Both are shown in Figures 13 and 14. When calling mk_def.sh, the state-swapping attempt probability is passed to the shell-script as an argument. In addition, the number of REPLICAS for the present simulation is defined here. It is explicitly defined by calling

```
./r_tempscale_AK_TIP3P.pl -t0 300 -n 2 >> REPLICA.def
```

For the very first small test case, I have limited the number of replicas just to two, which might be sufficient to check whether the program works. In case you have more computer power (nodes) available, you might enlarge this number. Therefore you have to edit

```
#!/bin/sh
./mk_def.sh 0.0   # No replica exchange
./init_gmx.sh     # Prepare startup files
lamboot -v machines.pas   #  Boot LAM
./do_min.sh       # Energy minimization
./do_eq0.sh       # Equilibration run without replica exchange
./mk_def.sh 0.01  # Attempt exchange moves with a probability of 0.01
r_mkgmx.pl < REPLICA.def > replica.gmx_out
./do_sim.sh 1 2   # Two successive simulation runs
wipe -v machines.pas   # Shutdown LAM
```

Figure 12: RUNSIM.sh performs the entire replica-exchange simulation run, including an initial energy minimization, an equilibration run, and two successive production runs.

```
#!/bin/sh
echo 'ATTEMPTS  1' > REPLICA.def
echo 'PROB ' $1   >> REPLICA.def
#./r_tempscale_AK_SPC.pl -t0 300 -n 6 >> REPLICA.def
./r_tempscale_AK_TIP3P.pl -t0 300 -n 2 >> REPLICA.def
#r_tempscale.pl -t0 300 -dt0 5 -dt1 5 -n 6 >> REPLICA.def
#r_rhoscale.pl -r0 0.960 -dr0 0.02 -dr1 0.02 -n 11 >> REPLICA.def
echo 'RHO 1.000' >> REPLICA.def
```

Figure 13: `mk_def.sh` creates a file `REPLICA.def` used to prepare the `replica.gmx`-file for the initial RPMDRUN simulation. `mk_def.sh` has to be called with a value defining the probability to attempt state-exchange moves.

```
#!/bin/sh

cat ../build/akpaq.gro | changedens_gmx 1.004707053 > akpaq.gro
/bin/cp ../build/akpaq.top .

r_mkgmx.pl < REPLICA.def > replica.gmx
r_mkgro.pl -gro akpaq.gro < REPLICA.def

nsites0=`cat ../build/akp.gro | head -2 | tail -1`
nmols1=`cat akpaq.gro | grep SOL | grep OW | wc -l`
r_mkmdx.pl -mols0 1 -sites0 $nsites0 -mols1 $nmols1 -sites1 3 > replica.mdx
```

Figure 14: `init_gmx.sh` prepares the entire simulation run by (1) copying the topology-file to the actual directory; (2) preparing `replica.gmx` and `replica.mdx`; (3) creating the initial start-configurations for each replica.

`mkdef.sh` (see Figure 13). The number of REPLICAS and the indicated lowest temperature define the actual temperature range. The scripts `r_tempscale_AK_TIP3P.pl` and `r_tempscale_AK_SPC.pl` have been fitted to a simulations of the AK-peptide in 2666 water (SPC, or TIP3P) molecules. In the following step, `init_gmx.sh` creates the input files for the `rpmdrun` program. You have probably noticed that in `RUNSIM.sh` (Figure 12), `mk_def.sh` is called with '0.0' as argument, indicating that replica-exchange is basically switched off. The reason for this is that `do_min.sh` is calling an energy minimization run. The steepest descent energy-minimization is actually performed by `rpmdrun`. The state-exchange wouldn't make any sense in this stage.a Before `rpmdrun` can be started, the entire parallel environment has to be set up, which is done by calling the `lamboot` command, as shown in Figure 12. The setup of the energy minimization run is shown in Figure 15.[2] Please note the out-commented call of `mdrun` in Figure 15. decommenting this line and out-commenting the `rpmdrun` call will execute the energy minimization run sequentially for each replica.

5.2.2 Equilibration and Production Run

After a having successfully called `do_min.sh`, the energy minimized start-configurations are stored in `R####_MIN.gro`. These configurations are loaded, when starting the equilibration run by calling `do_eq0.sh`. Please note that the same unchanged `replica.gmx` file is used here, since no replica exchange was supposed to happen. Again, also for the equilibration run,

[2] When preparing a VTREMD-simulation run involving large density changes, I found it useful to to employ a flexible water model just for the energy minimization step. Another strategy would be to avoid large stress in the initial configurations by successively increasing the density step by step, while performing a short equilibration run after each density change.

no replica exchanges are attempted The equilibration simulation is done with a rather tight-coupled ($\tau_T = -0.2\,\mathrm{ps}$) Berendsen-thermostat. For an equilibration run, there is no need of generating a proper canonical distribution. Instead, the Berendsen-method might be helpful, when states are considered that are far away from the equilibrium, since it safely shifts the system toward the desired-state. Once the equilibration run has been completed, the entire

```
#!/bin/sh

cat > START.mdp <<EOF

;       Input file
;
Integrator          = steep
emstep              0.001
emtol               100.0
title               =  Yo                       ; a string
cpp                 =  /lib/cpp                 ; c-preprocessor
dt                  =  0.002                    ; time step
nsteps              =  500                   ; number of steps
nstcomm             =  1                        ; reset c.o.m. motion
nstxout             =  0                        ; write coords
nstxtcout           =  100
nstvout             =  0                        ; write velocities
nstlog              =  50                       ; print to logfile
nstenergy           =  1                      ; print energies
nstlist             =  2                        ; update pairlist
ns_type             =  grid                     ; pairlist method
coulombtype         =  pme
;fourierspacing      =  0.1
fourier_nx          = 32
fourier_ny          = 32
fourier_nz          = 32
pmeorder            =  4
;optimize_fft        =  yes
ewald_rtol          =  1.0e-5
DispCorr            =  EnerPres
constraint_algorithm=  shake
shake_tol           =  1.0e-4
rlist               =  0.9                      ; cut-off for ns
rvdw                =  0.9                      ; cut-off for vdw
rcoulomb            =  0.9                      ; cut-off for coulomb
;Tcoupl              =  no
;Tcoupl              =  berendsen                ; temperature coupling
;ref_t               =  300
;tc-grps             =  System
;tau_t               =  0.1
;Pcoupl              =  berendsen                 ; pressure bath
;Pcoupltype          =  isotropic               ; pressure geometry
;tau_p               =  1.0                     ; p-coupoling time
;compressibility     =  4.5e-5
;ref_p               =  1.0
;gen_vel             =  no                      ; generate initial velocities
;gen_temp            =  300                      ; initial temperature
gen_seed            =  1993                     ; random seed

EOF

nodes=‘cat replica.gmx | head -3 | tail -1‘

let stop=$nodes
let stop=stop-1
let i=0
while [ $i -le $stop ]
do
label=‘echo $i | awk ’{printf " R%00004d", $1}’‘
echo $label

grompp -f START.mdp -p akpaq.top -po ${label}_STARTout.mdp -c ${label}_START.gro -o STRT.tpr
/bin/mv STRT.tpr ${label}_STRT.tpr

#mdrun -s ${label}_STRT.tpr -e ${label}_MIN.edr -g ${label}_MIN.log  \
#       -x ${label}_MIN.xtc -o ${label}_MIN.trr -c ${label}_MIN.gro

let i=i+1
done

mpirun -c $nodes rpmdrun -e MIN.edr -g MIN.log -x MIN.xtc -o MIN.trr -c MIN.gro
```

Figure 15: `do_min.sh` performs an energy minimization. This procedure is intended to overcome a possible unfavorable initial configuration.

production run is started by calling `do_sim.sh` (shown in Figure 17). `do_sim.sh` is called with two arguments. The start-index and the stop-index of a consecutive series of simulation runs. ′`do_sim.sh 1 10` ′ would produce a series of 10 production runs starting from the configurations produced by the initial equilibration run.

```
#!/bin/sh

cat > START.mdp <<EOF

;       Input file
;
title               =  Yo                        ; a string
cpp                 =  /lib/cpp                  ; c-preprocessor
dt                  =  0.002                     ; time step
nsteps              =  5000                      ; number of steps : Increase number of steps here !!
nstcomm             =  1                         ; reset c.o.m. motion
nstxout             =  0                         ; write coords
nstxtcout           =  0
nstvout             =  0                         ; write velocities
nstlog              =  50                        ; print to logfile
nstenergy           =  10                        ; print energies
nstlist             =  6                         ; update pairlist
ns_type             =  grid                      ; pairlist method
coulombtype         =  pme
fourier_nx          = 36
fourier_ny          = 36
fourier_nz          = 36
pmeorder            =  4
;optimize_fft        =  yes
ewald_rtol          =  1.0e-5
DispCorr            =  EnerPres
constraint_algorithm=  shake
shake_tol           =  1.0e-4
rlist               =  0.9                       ; cut-off for ns
rvdw                =  0.9                       ; cut-off for vdw
rcoulomb            =  0.9                       ; cut-off for coulomb
;Tcoupl              =  no
Tcoupl              =  berendsen                 ; temperature coupling : Berensen thermostat used here
ref_t               =  300 300
tc-grps             =  Protein SOL
tau_t               =  0.2 0.2                   ; tight initial berendsen coupling
Pcoupl              =  berendsen                 ; pressure bath
Pcoupltype          =  isotropic                 ; pressure geometry
tau_p               =  1.0                       ; p-coupoling time
compressibility     =  4.5e-5
ref_p               =  1.0
gen_vel             =  no                        ; generate initial velocities
gen_temp            =  300                       ; initial temperature
gen_seed            =  1993                      ; random seed

EOF

nodes=‘cat replica.gmx | head -3 | tail -1‘

let stop=$nodes
let stop=stop-1
let i=0
while [ $i -le $stop ]
do
label=‘echo $i | awk ’{printf " R%00004d", $1}’‘
echo $label
/bin/rm -f ${label}_STARTout.mdp
grompp -f START.mdp -p akpaq.top -po ${label}_STARTout.mdp -c ${label}_MIN.gro -o STRT.tpr
/bin/mv STRT.tpr ${label}_STRT.tpr
let i=i+1
done
sleep 5

mpirun -ger -c $nodes rpmdrun -e EQU0.edr -g EQU0.log -x EQU0.xtc -o EQU0.trr -c EQU0.gro
```

Figure 16: `do_eq0.sh` executes the equilibration run. Please note that the run is rather short (5000 timesteps). Usually one would use a longer simulation here, of, say 100 ps (50000 timesteps). Please note also the use of the Berendsen thermostat here. The Berendsen thermostat appears to be better suited for systems being far from the equilibrium state. For the initial equilibration period, generating a proper canonical distribution is probably of minor importance.

```
#!/bin/sh
start=$1
stop=$2

cat > SIMXX.mdp <<EOF
;       Input file
title               =  Yo                     ; a string
cpp                 =  /lib/cpp               ; c-preprocessor
dt                  =  0.002                  ; time step
nsteps              =  2500                   ; number of steps: Increase number of steps here !!
nstcomm             =  1                      ; reset c.o.m. motion
nstxout             =  0                      ; write coords
nstxtcout           =  1000
nstvout             =  0                      ; write velocities
nstlog              =  50000                  ; print to logfile
nstenergy           =  100                    ; print energies
nstlist             =  6                      ; update pairlist
ns_type             =  grid                   ; pairlist method
coulombtype         =  pme
fourier_nx          = 36
fourier_ny          = 36
fourier_nz          = 36
pmeorder            =  4
ewald_rtol          =  1.0e-5
DispCorr            =  EnerPres
constraint_algorithm=  shake
shake_tol           =  1.0e-4
rlist               =  0.9                     ; cut-off for ns
rvdw                =  0.9                     ; cut-off for vdw
rcoulomb            =  0.9                     ; cut-off for coulomb
Tcoupl              =  nose-hoover             ; temperature coupling
ref_t               =  300.0
tc-grps             =  System
tau_t               =  0.5
Pcoupl              = berendsen                ; pressure bath
;Pcoupl              = Parrinello-Rahman               ; pressure bath
Pcoupltype          =  isotropic               ; pressure geometry
tau_p               =  4.0                     ; p-coupoling time
compressibility     =  4.5e-5
ref_p               =  1.0
gen_vel             =  no                      ; generate initial velocities
gen_temp            =  300                     ; initial temperature
gen_seed            =  1993                    ; random seed
EOF

nodes=‘cat replica.gmx | head -3 | tail -1‘
let i=$start

while [ $i -le $stop ]
do

let j=i-1
infile=sim${j}.gro
if [ $i -eq 1 ]; then
   infile=EQU0.gro
fi
mdpoutfile=sim${i}out.mdp
tprfile=STRT.tpr
xtcfile=sim${i}.xtc
trrfile=sim${i}.trr
edrfile=sim${i}.edr
logfile=sim${i}.log
outfile=sim${i}.gro
/bin/rm -f STRT.tpr *STRT.tpr *out.mdp
/bin/cp replica.gmx_out replica.gmx

let nstop=$nodes
let nstop=nstop-1
let j=0
while [ $j -le $nstop ]
do
   label=‘echo $j | awk ’{printf "R%00004d", $1}’‘
   echo $label
   center_gmx 1 20 < ${label}_$infile > STRT.gro
   grompp -f SIMXX.mdp -p akpaq.top -po ${label}_$mdpoutfile -c STRT.gro -o STRT.tpr
   /bin/mv STRT.tpr ${label}_STRT.tpr
   let j=j+1
done

mpirun -c $nodes rpmdrun  -x $xtcfile -o $trrfile -c $outfile -e $edrfile -g $logfile
/bin/cp replica.gmx_out sim${i}.gmx_out

let i=i+1
done
```

Figure 17: `do_sim.sh` executes subsequent chunks of the production run. Please note that each run is rather short here (2500 timesteps). Usually one would use a longer simulation if $0.5 - 1.0$ ns (250000-500000 timesteps).

For the production run the Nosé-Hoover thermostat is used, since the VTREMD-algorithm is basically derived assuming a canonical distribution. I would like to emphasize the use of the `center_gmx` command here. It is used to translate the protein to the very center of the simulation box (and of course all the solvent molecules are translated as well). Both arguments specify the range of residues, which belong to the protein. Please note that this might have to be changed when studying a different protein.

5.3 Some Analysis

5.3.1 PVT-Data

The shell-script `do_pvt.sh`, located in the `RPMDRUN_DEMO/AK_DEMO/PVT` subdirectory, calculates the average potential energy, temperature and pressures, as well as the second moment of their distribution functions. This is basically done by the `scanavdata_gmx`-program, which comes with the REPLICA-package. `scanavdata_gmx` needs a `replica.gmx`-file in order to know about all V, T-states. I prefer to create it directly from `REPLICA.def` here, since this ensures that all states will be nicely ordered. `scanavdata_gmx` generally expects a data-stream coming in the PTE-format and extracts and accumulates the data, corresponding to each of the states. Please note that by selecting only particular PTE-files, one might restrict the averaging to certain parts of the total simulation run (e.g. the second half,...*etc*.).

```
#!/bin/sh

r_mkgmx.pl < ../run/REPLICA.def > replica.gmx

ptefiles=‘ls -t ../run/R*sim*.edr_pte‘
#ptefiles=‘ls -t ../run/R*sim[5-8].edr_pte‘

cat $ptefiles | scanavdata_gmx > PVT.dat
```

Figure 18: `do_pvt.sh` calculates the average temperature, potential energy, and pressure, as well as the second moment of their distribution functions for each state of the extended ensemble. The ouy-commented line illustrates how to restrict the averaging only to a certain part of simulation run.

5.3.2 Extracting Protein-Coordinates

Figure 19 contains the shell-script `extract.sh`, located in the `RPMDRUN_DEMO/AK_DEMO/extract` subdirectory. It extracts only the protein coordinates for each replica, which are usually distributed over a number of `.xtc`-files (that contain, in addition, also the solvent molecules), while creating one new `.xtc`-file for each replica, which is finally holding the complete entire trajectory of just the protein. The program `protextrx` is basically doing the job by cutting out a range of atoms of the original `.xtc`-files (Here: starting from atom 1 to atom 259). Please note that only every n'th configuration is written to the new file, which can be specified at the `-gap`-option. For completeness, also a corresponding PTE-file is created. Here. the Perl-script `./select_nth_time.pl` just passes through every n'th line of the incoming data-stream. In combination with pdb-file for the peptide without solvent, `akp.pdb`, contained in `RPMDRUN_DEMO/AK_DEMO/build`, the new `.xtc`-files are ready to be viewed with VMD. The Option `-ignore_step0` is provides

```
#!/bin/sh

nodes=‘cat ../run/replica.gmx | head -3 | tail -1‘

let nstop=$nodes
let nstop=nstop-1

GAP=10

let j=0

while [ $j -le $nstop ]
do

  label=‘echo $j | awk '{printf "R%00004d", $1}'‘

  xtcfiles=‘ls -t ../run/${label}_sim*.xtc‘
  ptefiles=‘ls -t ../run/${label}_sim*.xtc_pte‘
  xtcoutfile="${label}_prot.xtc"
  pteoutfile="${label}_prot.xtc_pte"

  echo $label

  protextrx -ignore_step0 -gap $GAP -xtc $xtcfiles -cnull 0 -ncsites 259 -xtcout $xtcoutfile
  cat $ptefiles | ./select_nth_time.pl -n $GAP > $pteoutfile

  let j=j+1
done
```

Figure 19: `extract.sh` Extracts just the protein coordinates contained in all `.xtc`-files belinging to one particular replica and writes them to a new `.xtc`-file. This might be used to watch the folding/unfolding of the individual replicas. Here just every 10'th configuration is extracted.

an (unfortunately) necessary workaround, since the initial configuration (configuration '0', if you like) is written to the `.xtc`-files, but not to the PTE-files. Using `-ignore_step0`, `protextrx` actually will ignore this initial configuration.

5.3.3 Learning More About the Peptide-Configuration: The Radius of Gyration

In the `RPMDRUN_DEMO/AK_DEMO/gyr` directory it is actually demonstrated, how to extract a particular structural (or any other) information about the peptide as a function of the considered states of an entire extended ensemble. Here we discuss only properties that can be extracted by using one of the (standard) programs that come with the GROMACS distribution. In particular, only one application is discussed here: The calculation of the radius of gyration, as obtained by calling the `g_gyrate`-program [2]. However, any other property might be obtained in a straightforward fashion by calling other programs. sThe calculation is done in three stages: First, the radius of gyration is calculated for each replica as a function of simulation time. This is achieved by the shell-script `get_simx.sh`, shown in Figure 20. Please note, that the range of considered parts of simulation runs has to be specified in the script by assigning values to the variable `$min` and `$max`. The calculated property can thus be restricted to certain parts of the total simulation run. Now, for each available protein-configuration the radius of gyration will be calculated and the `g_gyrate`-output is written to `gyrate.xvg`. After removing some irrelevant information, the content is added to a file `R####_gyr.dat`, which finally contains the radius of gyration for each of the replicas over the entire discussed part of the simulation. Please note that `g_gyrate` is called with the '`-b 1.0`'-option. The intention is basically, that the first configuration in each of the `.xtc`-files is skipped, as it has been discussed (more in detail) in the previous section. To calculate the average radius of gyration as a function of the states, the `scanavdata_gmx`-program is used

```
#!/bin/sh

nodes=‘cat ../run/replica.gmx | head -3 | tail -1‘

let nstop=$nodes
let nstop=nstop-1

let min=1
let max=2

let j=0

while [ $j -le $nstop ]
  do

  let i=$min

  label=‘echo $j | awk ’{printf "R%00004d", $1}’‘
  outfile=${label}_gyr.dat

  /bin/rm -f $outfile

  while [ $i -le $max ]
    do

    xtcfile=‘ls ../run/${label}_sim${i}.xtc‘

    echo 4 | g_gyrate -b 1.0 -s ../run/R0000_MIN.gro -f $xtcfile
    echo $label

    cat gyrate.xvg | grep -v ’#’ | grep -v ’@’ | awk ’{print $2}’ >> $outfile
    /bin/rm -f gyrate.xvg

    let i=i+1
  done
  let j=j+1
done
```

Figure 20: `get_simx.sh` calculates the radius of gyration for all replicas. For each replica a file `R####_gyr.dat` is created which contains the radius of gyration for each of the available configurations. For each configuration there will be a new line. The are written in plain ASCII format.

```
#!/bin/sh

nodes=‘cat ../run/replica.gmx | head -3 | tail -1‘
let nstop=$nodes
let nstop=nstop-1

let j=0
while [ $j -le $nstop ]
  do

  label=‘echo $j | awk ’{printf "R%00004d", $1}’‘

  echo $label
  ptefiles=‘ls ../run/${label}_sim[1-2].xtc_pte‘

  cat $ptefiles | cut -c 1-9 > a.dat
  cat $ptefiles | cut -c 22- > b.dat

  paste a.dat ${label}_gyr.dat b.dat > ${label}_mix.dat

  let j=j+1
done
```

Figure 21: `do_mix.sh` intermixes the `R####_gyr.dat`-files and the PTE-files in order to be able to assign each obtained value to its corresponding state-point.

the same way, as it is shown done in section 5.3.1. Therefore the corresponding PTE-files and `R####_gyr.dat`-files have to be *mixed*, which is done by `do_mix.sh`, as shown in Figure 21. Here, the PTE-files are basically splitted vertically and the second column of each PTE-file is cut out. Finally, the two halfs of the PTE-file and the `R####_gyr.dat`-file are pasted

```
#!/bin/sh

r_mkgmx.pl < REPLICA.def > replica.gmx

cat R*mix.dat | scanavdata_gmx > GYR.dat
```

Figure 22: `do_average.sh` calculates averages for each of the state-points of the entire extended ensemble as it has been shown in section 5.3.1. However, instead of calculating the average temperature, now the average radius of gyration is obtained.

together to form a new file, which has actually a PTE-format, but its second column contains now the actual radius of gyration. `do_average.sh` passes the content of these files through `scanavdata_gmx`, which comes finally up with the average radius of gyration as a function of each of the considered state-points.

6 A (Preliminary) Concluding Remark

A lot of work has still to be done, basically I will try to add a section on the incorporation of the AMBER-forcefield into GROMACS as soon as possible. However, in order to make the GROMACS/AMBER-stuff convenient to work with, some additional programs still need to be written. Basically, the data-extraction step has to be improved. Finally, I hope this little HOWTO is not messed up too much and *at least* helpful in a sense.

References

[1] E. Lindahl, B. Hess and D. van der Spoel, "Gromacs 3.0: A package for molecular simulation and trajectory analysis", *J. Mol. Mod.*, **7**, 306–317, (2001). 1

[2] D. van der Spoel, E. Lindahl, B. Hess, A. R. van Buuren, E. Apol, P. J. Meulenhoff, D. P. Tieleman, A. L. T. M. Sijbers, K. A. Feenstra, R. van Drunen and H. J. C. Berendsen. *Gromacs User Manual version 3.2*. www.gromcs.org, (2004). `ftp://ftp.gromacs.org/pub/manual/3.2/manual-3.2.pdf`. 1, 8, 14, 20

[3] D. Paschek and A. Geiger. *MOSCITO 4*. Department of Physical Chemistry, University of Dortmund, (2002). `http://ganter.chemie.uni-dortmund.de/MOSCITO`. 2, 8

[4] D. Frenkel and B. Smit. *Understanding Molecular Simulation — From Algorithms to Applications*. Academic Press, San Diego, 2nd Ausgabe, (2002). 2

[5] Y. Sugita and Y. Okamoto, "Replica exchange molecular dynamics method for protein folding", *Chem. Phys. Lett.*, **314**, 141–151, (1999). 2

[6] S. Gnanakaran, H. Nymeyer, J. Portman, K. Y. Sanbonmatsu and A. E. García, "Peptide folding simulations", *Curr. Opin. Struct. Bio.*, **13**, 168–174, (2003). 2

[7] C. Lin, C. Hu and U. Hansmann, "Parallel Tempering Simulations of HP-36", *Proteins: Struct., Funct.,Genet.*, **52**, 436–445, (2003). 2

[8] C. A. Royer, "Revisiting volume changes in pressure-induced protein unfolding", *Biochim. Biophys. Acta-Protein Struct. Molec. Enzym.*, **1595**, 201–209, (2002). 2

[9] D. Paschek and A. E. García, "Reversible temperature and pressure denaturation of a protein fragment: A replica exchange molecular dynamics simulation study", *Phys. Rev. Lett.*, (2004). in press. 3

[10] Y. M. Rhee and V. S. Pande, "Multiplexed-Replica Exchange Molecular Dyanmics Method for Protein Folding Simulation", *Biophys. J.*, **84**, 775–786, (2003). 5

[11] F. v. Hoesel. "XRD-library for data compression", (1995). `http://hpcv100.rc.rug.nl/~hoesel/xdrf.html`. 8

[12] W. Humphrey, A. Dalke and K. Schulten, "VMD – Visual Molecular Dynamics", *J. Molec. Graphics*, **14**, 33–38, (1996). `http://www.ks.uiuc.edu/Research/vmd/`. 8, 14

[13] S. Gnanakaran, R. M. Hochstrasser and A. E. García, "Nature of structural inhomogeneities on folding a helix and their influence on spectral measurements", *Proc. Natl. Acad. Sci. USA*, **101**, 9229–9234, (2004). 13