

# ParallelReax Manual

by Hasan Metin Aktulga  
Feb 15, 2010

## 1 Input Files

ParallelReax expects 3 input files: geometry, force field and control files.

### 1.1 Geometry File

Geometry file tells about the types and initial positions of the atoms in the system. ParallelReax supports geometry files in *pdb* and custom formats. It is also possible to restart from an earlier simulation using a restart file (which can be either in ascii or binary format).

#### 1.1.1 *pdb* format

For more information on *pdb* format, please visit <http://www.wwpdb.org/docs.html>. Input files of various other formats can easily be converted to *pdb* using the freely available OpenBabel software ([http://openbabel.sourceforge.net/wiki/Main\\_Page](http://openbabel.sourceforge.net/wiki/Main_Page)).

*pdb* format limits the number of atom serial digits just to 5, therefore the maximum number of atoms that can be input using the *pdb* format is only 100000.

#### 1.1.2 custom format

ParallelReax features a custom geometry format as well. The format of the custom geometry file is very simple. The first line describes the simulation box, the second line gives the total number of atoms. Then there needs to be a single line for each atom describing it in detail.

```
BOXGEO x_len y_len z_len alpha beta gamma
N
1 ele1 name1 x1 y1 z1
```

```

2 ele2  name2  x2 y2 z2
.
.
.
N eleN  nameN  xN yN zN

```

First three floating point numbers on the first line give the length of the simulation box in x, y, z dimensions, the remaining ones are for the angles between them. Currently, *ParallelReax* works only with an orthogonal box meaning all angles need to be 90.0 degrees. There is no limit by the format on the number of atoms that can be input. Each atom line consists of 6 fields:

- an integer denoting the atom serial
- a string for the chemical symbol of the element (2 characters max, case insensitive)
- a string for the atom name (7 characters max)
- 3 floating point numbers describing the position in cartesian coordinates

## 1.2 Force Field File

Force field file contains the *ReaxFF* parameters to be used during the simulation.

## 1.3 Control File

Parameters in the control file allow the user to tune various simulation options. Parameter names are case-sensitive but their order is not important (except that *ensemble\_type* needs to precede *p\_mass* and *pressure*). Described below are the fields that you might use in a control file. If a parameter is missing from the control file, its default value will be assumed.

```
simulation_name test_parallel
```

Output files produced by *ParallelReax* will be in *simulation\_name.some\_extension* format. They will be discussed in more detail in section 3. Default value is “simulate”.

```
ensemble_type    1    ! 0:NVE 1:bNVT 2:nhNVT 3:sNPT 4:iNPT 5:NPT
```

*ensemble\_type* denotes the type of ensemble to be produced by *ParallelReax*. Supported ensemble types are as follows:

- 0: NVE
- 1: bNVT - NVT with Berendsen thermostat
- 2: nhNVT - NVT with Nose-Hoover thermostat (under testing)
- 3: sNPT - semiisotropic NPT with Berendsen's coupling
- 4: iNPT - isotropic NPT with Berendsen's coupling
- 5: NPT - anisotropic NPT with Parrinello-Rehman coupling (under development)

*ensemble\_type* is NVE by default.

```
nsteps          1000    ! number of simulation steps
dt              0.25    ! time step in fs
```

*nsteps* controls the total number of steps to be simulated and *dt* controls the length of each time step (measured in *femtoseconds*). Number of steps is 0 by default and timestep length is 0.25 fs.

```
proc_by_dim    1 1 3    ! decomposition of the simulation box to processors
```

ParallelReax uses the domain decomposition technique to distribute the load among processors, it currently does not have dynamic load balancing. *proc\_by\_dim* denotes the desired decomposition of the simulation box into subboxes (first integer is the number of equal-length partitions in x dimension, second integer is for y dimension and the last one is for z dimension). Each subbox is subsequently assigned to a processor. ParallelReax constructs a periodic 3D mesh based on *proc\_by\_dim* parameter. The default is to use a single processor without any decomposition.

```
geo_format     0      ! 0:custom 1:pdb 2:ASCII restart 3:binary restart
```

*geo\_format* parameter informs *ParallelReax* about the format of the geometry file to read. *pdb* and custom formats were already discussed at section 1.1. The default input format is the custom one.

Other options include resuming from an older run by setting *geo\_format* to “ASCII” or “binary” restart and providing the name of the restart file as an argument to *ParallelReax*. Then *ParallelReax* will read the box geometry, positions and velocities for all atoms in the system from the restart file and continue execution thereon.

```
tabulate_long_range 10000 ! granularity of tables, 0 no tabulation
```

When set to  $m$  (must be a positive integer), this option turns on the tabulation optimization for computing electrostatics and van der Waals interactions. The range  $[0, \text{cutoff}]$  is sampled at  $m$  equally spaced points; energy and forces due to long range interactions between each atom type in the system are computed at each of these sample points and stored in a table. Then for each interval, we compute the coefficients of its cubic spline interpolation function. During the simulation when we need to compute the long range interactions between any two atoms, we locate the corresponding interpolation function and compute the energy and forces between them by interpolating. This method gives huge speed-up compared to computing everything from scratch each time and with only 10000 sample points it is able to provide accuracies at the machine precision level. The default is no tabulation.

```
energy_update_freq 10
```

This option controls the frequency of writes into output files (besides the trajectory and restart files which are controlled by some other parameters explained later) described in section 3. The default value for this parameter is 0, meaning there will not be any energies and performance logs output.

```
remove_CoM_vel 500 ! removal of CoM trans&rot vel freq
```

Removal of translational and rotational velocities around the center of mass needs to be done for *NVT* and *NPT* type simulations to remove the unphysical effects of scaling velocities. In case of *NVE*, this is not necessary and is not done regardless of the value of *remove\_CoM\_vel*. The default value is to remove translational and rotational velocities at every 250 steps.

```

nbrhood_cutoff 5.0      ! bonded intr cutoff in Å
thb_cutoff      0.001    ! bond strength threshold for 3-body intrs
hbond_cutoff    7.50     ! cutoff distance (H--Z) for hydrogen bonds

```

These cutoff parameters are crucial for the correctness and efficiency of *ParallelReax*. Normally, bonded interactions are truncated after 4-5 Å in *ReaxFF* and this is controlled by the *nbrhood\_cutoff* parameter (default value is 4 Å).

*thb\_cutoff* sets the bond strength threshold for valence angle interactions. Bonds which are weaker than *thb\_cutoff* will not be included in valence angle interactions (default *thb\_cutoff* is 0.001).

Currently, *hbond\_cutoff* is set pretty conservatively to 7.5 Å in *ReaxFF*. It should be okay to lower it to 6 Å for the newer force field files. If *hbond\_cutoff* is set to 0, hydrogen bond interactions will be turned off completely (could be very useful for increasing the performance of simulations where it is apriori known that there are no hydrogen bonding interactions) and this is the default behaviour.

```

reneighbor      10      ! in steps
vlist_buffer    2       ! in angstroms

```

*ParallelReax* features delayed neighbor generation by using Verlet lists. *reneighbor* controls the reneighboring frequency and *vlist\_buffer* controls the buffer space beyond the nonbonded interaction cutoff. By default, *vlist\_buffer* is set to 0 and reneighboring is done at every step.

```

q_err           1e-6    ! norm of the relative residual in QEq solve
qeq_freq        1       ! frequency to update charges with QEq

```

In *ParallelReax*, we use a CG solver (with a diagonal preconditioner) for the charge equilibration problem. *q\_err* denotes the stopping criteria for the CG solver. A lower threshold would yield more accurate charge equilibration at the expense of increased computational time. A threshold of  $10^{-6}$  should be good enough for most cases and this is the default value.

*qeq\_freq* can be used to perform charge equilibration at every few steps instead of the default behaviour of performing it at every step. Although doing QEq less frequently would save important computational time, it is not recommended. Because this might cause wild fluctuations in energies.

```

temp_init      0.0      ! desired init T
temp_final     300.0    ! desired final T
t_mass         0.1666   ! thermal inertia in fs

```

Temperature coupling parameters are effective in all types of ensembles except for *NVE*. Initial temperature is controlled via the *temp\_init* parameter including the *NVE* ensemble. 0 K is the default value for *temp\_init* and 300 K is the default value for *temp\_final*.

*ParallelReax* features both Berendsen and Nose-Hoover type thermostats. *t\_mass* of 500.0 should be okay (and is the default) for the Berendsen thermostat and 0.166 should be okay for the Nose-Hoover thermostat in most systems to get a good convergence rate.

*Important note: Nose-Hoover thermostat is still under testing.*

```

pressure       0.000101 0.000101 0.000101 ! ext pressure in GPa
p_mass         5000.0   5000.0   5000.0   ! pressure inertia in fs

```

Pressure coupling parameters are needed only when working with *NPT*-type ensembles. Currently *iNPT* (isotropic NPT) and *sNPT* (semi-isotropic NPT) are the available pressure coupling ensembles. Berendsen thermostats and barostats are used in both methods [2]. To have a stable system, using a *p\_mass* value around 5000.0 (which is the default already) together with a *t\_mass* of 500.0 should be fine.

For *iNPT* ensemble, *pressure* parameter expects a single floating number (in case there are more, they will simply be ignored) to control pressure. For *sNPT* ensemble, *pressure* parameter expects 3 floating point numbers to control pressure on each side. Same things apply for *p\_mass* as well.

```

write_freq     100     ! write traj at every 'this' many steps
traj_method    1       ! 0: simple parallel I/O, 1: MPI I/O

```

Trajectory of the simulation will be output to the trajectory file at every *write\_freq* steps. For making analysis easier, the trajectory file is written as an ASCII file. By default, no trajectory file is written.

*ParallelReax* can output trajectories either using simple MPI send/receives (option 0 which is the default) or using MPI I/O calls (option 1) which are part of the MPI-2 standard. The latter option is supposed to be more efficient (not verified by tests though) but may not be available in some MPI implementations.

```

traj_title      NVT_SIMULATION_OF_WATER
atom_info       1      ! 1: print basic atom info
atom_forces     1      ! 1: print the force on each atom
atom_velocities 1      ! 1: print atom velocities
bond_info       0      ! 1: print bonds
angle_info      0      ! 1: print angles

```

Currently ParallelReax only outputs trajectories in its custom trajectory format. This custom format starts with a trajectory header detailing the trajectory title and control parameters used for the simulation. A brief description of atoms follows the trajectory header with atom serial numbers and what element each atom is.

Then at each *write\_freq* steps (including step 0), a trajectory frame is appended to the trajectory file. The frame header which gives information about various potential energies, temperature, pressure and box geometry is standard. However, the latter parts of the frame can be customized using *atom\_info*, *atom\_forces*, *atom\_velocities*, *bond\_info* and *angle\_info* parameters which are already self-explanatory. The ordering is atoms section, bonds section and angles section assuming they are all present. By default, all atom, bond and angle information outputting is turned off.

One nice property of the custom trajectory format is that each part of the trajectory is prepended by a number that can be used to skip that part. For example, the trajectory header is prepended by an integer giving the number of characters to skip the control parameters section. The initial atom descriptions is prepended by the number of characters to skip the initial descriptions part and another one that tells the number of atom description lines. Similar numbers are found at the start of each section within a trajectory frame as well. So the general layout of our custom trajectory format is as follows (assuming all trajectory options are turned on):

```

CHARS_TO_SKIP_SECTION
trajectory header
CHARS_TO_SKIP_ATOM_DESCS NUM_LINES
atom descriptions
CHARS_TO_SKIP_FRAME_HEADER
frame1 header
CHARS_TO_SKIP_ATOM_LINES NUM_ATOM_LINES
frame1 atom info
CHARS_TO_SKIP_BOND_LINES NUM_BOND_LINES

```

```

frame1 bond info
CHARS_TO_SKIP_ANGLE_LINES NUM_ANGLE_LINES
frame1 angle info
.
.
.
CHARS_TO_SKIP_FRAME_HEADER
frameN header
CHARS_TO_SKIP_ATOM_LINES NUM_ATOM_LINES
frameN atom info
CHARS_TO_SKIP_BOND_LINES NUM_BOND_LINES
frameN bond info
CHARS_TO_SKIP_ANGLE_LINES NUM_ANGLE_LINES
frameN angle info

restart_format    1 ! 0: restarts in ASCII  1: restarts in binary
restart_freq      0 ! output a restart file at every 'this many' steps

```

ParallelReax can output restart files both in ASCII and binary formats. While ASCII format is good for portability, binary restart files are much more compact and does not cause any loss of information due to truncation of floating point numbers. For this reason binary restarts is the default.

There will not be any restart files output unless *restart\_freq* parameter is set to a positive integer. A restart file will carry the name *simulation\_name.resS* where *S* denotes the step that the restart file is written).

## 2 How to Compile and Run

When you extract the *ParallelReax.tar.gz* file with the command

```
gtar xvzf ParallelReax.tar.gz
```

a new directory, *ParallelReax*, will appear in your working directory. It contains the source code directory (*src*) along with a directory for sample systems (*examples*).

You can compile *ParallelReax* by switching to the source directory and typing *make*. This will produce the executable file, *ParallelReax*, inside the source directory. The Makefile that comes in the distribution assumes OpenMPI as the default MPI



implementation and *mpicc* as the default MPI compiler. In case you have a different MPI implementation, please set your MPI compiler in the Makefile appropriately.

ParallelReax requires 3 input files as mentioned in section 1. For example, the command to run *ParallelReax* with OpenMPI is as follows:

```
mpirun -np num_procs -machinefile machines ParallelReax geo ffield control
```

### 3 Output

*ParallelReax* writes its output files into the directory where it is run. There are a number of output files all of which have the *simulation\_name* as the first part of their names followed by its unique extension:

- **.out** contains a summary of the simulation progress. Its format is:

```
Step    Total Energy    Potential    Kinetic    T (in K) Volume(in A^3) P(in GP)
```

- **.pot** contains detailed information regarding various types of energies that comprise the total potential energy:

```
Step    Bonds    OverCoor+UnderCoor    LonePair
Angle+Penalty    3-body Coalition Hydrogen Bonds    Torsion    4-body Conjugation
vander Waals    Coulomb    Polarization
```

- **.log** is intended for performance tracking purposes. It displays the total time per step and what parts of code take up how much time to compute.
- **.prs** is output only when pressure coupling is on. It displays detailed information regarding the pressure and box dimensions.
- **.trj** is the trajectory file. Atom positions are written into this file at every *write\_freq* steps using the desired format as explained before. Each frame is concatenated one below the other.

Apart from these, there might be some text printed to *stderr* for debugging purposes. If you encounter some problems with the code (like a segmentation fault or unexpected termination of the code), please contact me with the error message printed to *stderr*.

## 4 Tested Architectures

ParallelReax has been tested on a variety of platforms including:

- OpenMPI on Intel i7 processor
- OpenMPI on Intel Quadcore cluster
- OpenMPI on AMD Opteron cluster (hera @ LLNL)
- MPICH2 with Intel C compiler on AMD Opteron cluster (hera @ LLNL)

## 5 Performance

ParallelReax inherits many features from SerialReax algorithmically, i.e. its general code structure, memory management, neighbor generation and force computation routines are almost identical to those of SerialReax except where parallelization needs are to be addressed. One significant difference, though, is the linear solver used for the QEq problem. We use a GMRES solver with an ILU-based preconditioner in SerialReax. However, neither GMRES nor ILU factorization algorithms cannot be made parallel easily and efficiently. Therefore in ParallelReax, we have substituted the GMRES solver with a CG solver and the ILU-based preconditioner with a simple diagonal preconditioner. The resulting QEq solver is less efficient compared to the one in SerialReax. This and other overheads due to parallelization cause ParallelReax on a single processor to be slower than SerialReax. The performance loss can be up to 40-50% depending on the system and simulation parameters.

### 5.1 Strong Scaling Test: Bilayer System

We have performed the strong scaling test (keep the system size constant, measure parallel efficiency while increasing the number of processors) on a 56800 atom bilayer-water system.

During our tests, we have set the values of simulation parameters that are critical for the performance of ParallelReax as follows:

- ensemble = 1 - Berendsen NVT
- nsteps = 1000

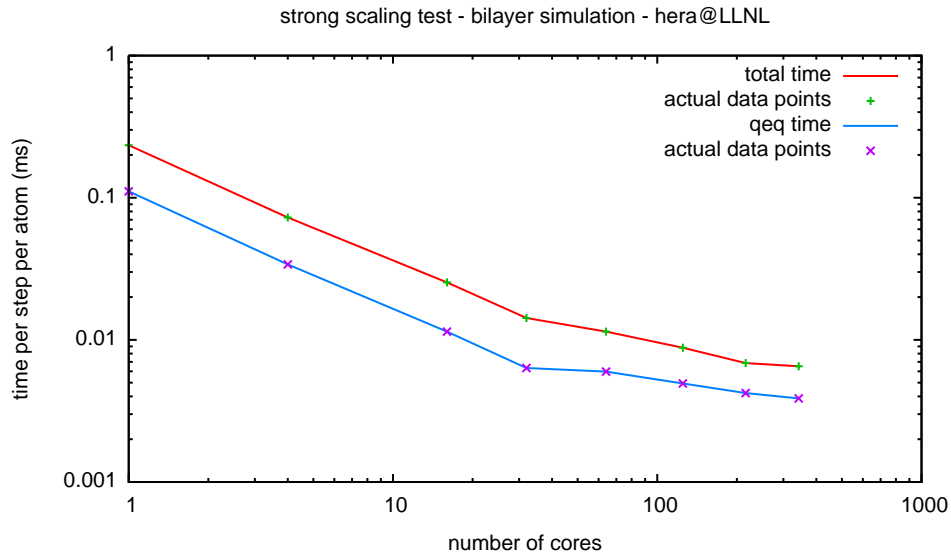
- `dt = 0.25`
- `tabulate_long_range = 10000`
- `energy_update_freq = 10`
- `remove_CoM_vel = 500`
- `reneighbor = 1`
- `vlist_buffer = 0`
- `nrhood_cutoff = 5.0`
- `hbond_cutoff = 7.5`
- `thb_cutoff = 0.001`
- `qeq_freq = 1`
- `q_err = 1e-6`

We have compiled the source code using the Intel C compiler with MVAPICH2 (mpiicc) and the following optimization flags:

```
-O3 -funroll-loops -fstrict-aliasing
```

We have used the Hera cluster at LLNL for our experiments. Hera has 800 batch nodes, each with 4 AMD Opteron Quadcore CPUs at 2.3 GHz and 32 GBs of memory. Nodes are interconnected with InfiniBand switches and work on the CHAOS 4.2 operating system. Table 5.1 and figure 5.1 show how ParallelReax scales with the increasing number of processors on the bilayer system.

executable	num cores	time per step(sec)	QEq time per step(sec)
SerialReax (icc -fast)	1	7.76	1.34
SerialReax (gcc -O3)	1	9.95	1.35
ParallelReax	1	13.30	6.30
ParallelReax	4	4.13	1.93
ParallelReax	16	1.44	0.65
ParallelReax	32	0.81	0.36
ParallelReax	64	0.65	0.34
ParallelReax	125	0.50	0.28
ParallelReax	216	0.39	0.24
ParallelReax	343	0.37	0.22



Poor performance of ParallelReax compared to SerialReax can largely be attributed to the much slower QEq solver used in ParallelReax.

## 5.2 Weak Scaling Test: Bulk Water System

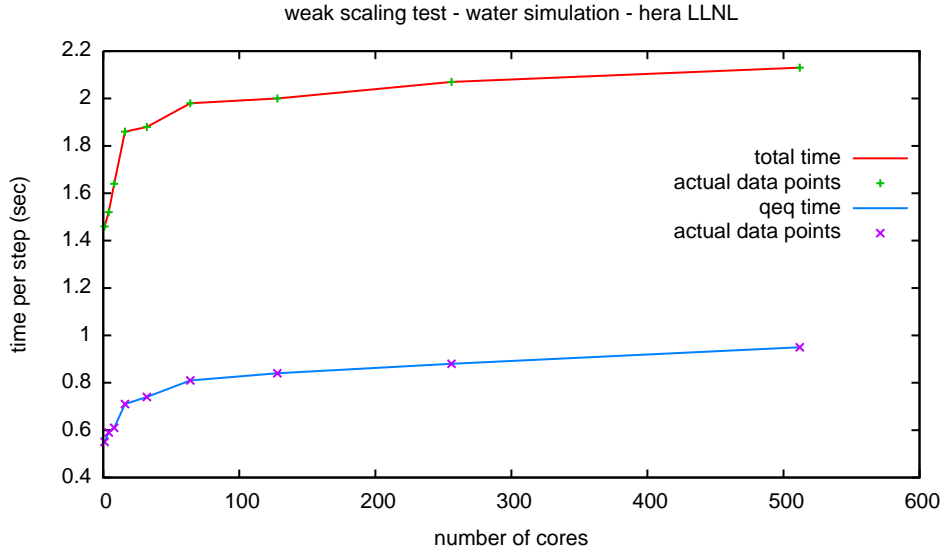
We have performed the weak scaling test (increase both the problem size and the number of processors at the same rate to keep the size of the problem assigned to a processor constant), on a 6540 atom bulk water system.

During our tests, we have set the values of simulation parameters that are critical for the performance of ParallelReax as follows:

- ensemble = 1 - Berendsen NVT
- nsteps = 1000
- dt = 0.25
- tabulate\_long\_range = 10000
- energy\_update\_freq = 10
- remove\_CoM\_vel = 500
- reneighbor = 1
- vlist\_buffer = 0
- nbrhood\_cutoff = 4.5
- hbond\_cutoff = 7.5
- thb\_cutoff = 0.001
- qeq\_freq = 1
- q\_err = 1e-6

Compilation of the parallel code and the platform we have tested it on are the same as in section 5.1. Table 5.2 and figure 5.2 show the results of weak scaling experiments.

executable	num cores	time per step(sec)	QEq time per step(sec)
SerialReax (icc -fast)	1	0.74	0.12
SerialReax (gcc -O3)	1	0.96	0.13
ParallelReax	1	1.46	0.55
ParallelReax	4	1.52	0.59
ParallelReax	8	1.64	0.61
ParallelReax	16	1.86	0.71
ParallelReax	32	1.88	0.74
ParallelReax	64	1.98	0.81
ParallelReax	128	2.00	0.84
ParallelReax	256	2.07	0.88
ParallelReax	512	2.13	0.95



Weak scaling numbers look satisfactory in general (if we consider in the context of ParallelReax only). However, there is a sudden jump in the per step running time when we go from 1 to 16 processors. We believe this is mostly due to caching/loading issues in the 4 Quadcore processors packed cluster nodes. So weak scaling numbers from 16 processors to 512 processors look nice despite the intensive communications required for QEq (credit goes to the very well constructed Hera cluster).

## References

- [1] Glenn J. Martyna, Douglas J. Tobias, and Michael L. Klein. “Constant pressure molecular dynamics algorithms.” *The Journal of Chemical Physics* 101, 4177 (1994).
- [2] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, and J. R. Haak. “Molecular dynamics with coupling to an external bath.” *The Journal of Chemical Physics* 81, 3684-3690 (1984).