

ESPResSo

Mesoscopic Molecular Modeling

Analysis, Checkpointing, and the TestSuite

*What do you
want to sim
today?*



The Chemical Company

Analysis

What do you want
to sim today?

 **BASF**
The Chemical Company

```
acinclude.m4      communication.h  dihedral.h      global.h         ljcos2.h        mpi.h            rattle.c        sub_tlj.h
aclocal.m4       config          doc              GPL.TXT         ljcos.h         nemd.c          rattle.h        tab.h
angle.h          config.c       domain_decomposition.c  grid.c         lj.h            nemd.h          README         test.sh
archconfig.h.in config.cgp.h   domain_decomposition.h  grid.h         Maestro3       nsquare.c      RELEASE_NOTES  testsuite
autogen.sh       config.h       doxygen_config    harmonic.h     maggs.c        nsquare.h      renice.tcl     thermostat.c
autom4te.cache   config.log     elc.c            imd.c          maggs.h        obj-Xeon-pc-linux  rotation.c     thermostat.h
binary_file.c    config.old.h   elc.h            imd.h          main.c         p3m.c          rotation.h     topology.c
binary_file.h    config.status  energy.c         initialize.c   Makefile       p3m.h          samples        topology.h
bin.c            configure     energy.h         initialize.h   Makefile.in    packages       scripts        trace_memory.tcl
bin.h            configure.ac   errorhandling.c  INSTALL        Makefile.Xeon-pc-linux  parser.c      simulation.tcl  tuning.c
blockfile.c      configure.ac.raw  errorhandling.h  integrate.c   mmm1d.c        particle_data.c  soft_sphere.h  tuning.h
blockfile.h      constraint.c   Espresso        interaction_data.c  mmm1d.h        particle_data.h  specfunc.c     utils.h
blockfile_tcl.c  constraint.h   fene.h          interaction_data.h  mmm2d.c        polymer.c       specfunc.h     uwerr.c
blockfile_tcl.h  copyright.sh  fft.h           internal        mmm2d.h        polymer.h       statistics.c    uwerr.h
blockfile_test.c CVS           forces.c        layered.c       mmm-common.c   polynom.h      statistics_chain.c  verlet.c
buckingham.h    cvs_ssh.sh    forces.h        layered.h       mmm-common.h   pr             statistics_cluster.c  verlet.h
cells.c          darwinlink.sh gb.h            lb.c           modes.c         pressure.c      statistics_cluster.h  version.h
cells.h          debug.c       ghosts.c        lb.h           modes.h         pressure.h      statistics.h     vmdsock.c
comfixed.h      debug.h       ghosts.h        LICENSE.TXT    morse.h        random.c       statistics_molecule.c  vmdsock.h
comforce.h      debye_hueckel.h  global.c       Linux          mpi.c          random.h       statistics_molecule.h
```

- access using `analyze <what> [<options>]` from the Tcl-Level, or by invoking the corresponding C-function from within the C-Level
- each feature shall consist of two parts:
 - `parser_parse_NAME(interp, argc-2, argv+2)` for I/O to Tcl
 - main function `NAME(<arguments>)` to perform actual analysis, returning results as argument, pointer, or directly

Analysis

What do you want
to sim today?



- enter here in `statistics.c` from Tcl:

```
/*
 *                               main parser for analyze
 */
int analyze(ClientData data, Tcl_Interp *interp, int argc, char **argv)
{
    int err = TCL_OK;
    if (argc < 2) {
        Tcl_AppendResult(interp, "Wrong # of args! Usage: analyze <what> ...", (char *)NULL);
        return (TCL_ERROR);
    }

    if (ARG1_IS_S("set"))
        err = parse_analyze_set_topology(interp, argc - 2, argv + 2);
    else if (ARG1_IS_S("get_folded_positions"))
        err = parse_get_folded_positions(interp, argc - 2, argv + 2);
    else if (ARG1_IS_S("configs"))
        err = parse_configs(interp, argc - 2, argv + 2);
    else {
        /* the default */
        Tcl_ResetResult(interp);
        Tcl_AppendResult(interp, "The operation \"", argv[1],
            "\" you requested is not implemented.", (char *)NULL);
        err = (TCL_ERROR);
    }
}
```

Analysis

What do you want
to sim today?



- enter here in `statistics.c` from Tcl
- continue to parser / functions in
 - file `statistics.c`:
 - `double mindist(IntList *set1, IntList *set2)`
 - `int aggregation(...)`
 - `void centermass(int type, double *com)`
 - `void momentofinertiamatrix(...)`
 - `void nbhood(..., IntList *il, int planedims[3])`
 - `double distto(double p[3], int pid)`
 - `void calc_cell_gpb(..., double *result)`
 - `void calc_part_distribution(..., double *dist)`
 - `void calc_rdf(..., double *rdf)`
 - `void calc_rdf_av(..., double *rdf, ...)`
 - `void calc_rdf_intermol_av(...)`
 - `void analyze_structurefactor(..., double **_ff)`
 - `int analyze_radial_density_map (...)`

- enter here in `statistics.c` from Tcl
- continue to parser / functions in
 - file `statistics.c`
 - file `statistics_chain.c`:
 - `void calc_re{_av}(double **re);`
 - `void calc_rg{_av}(double **rg);`
 - `void calc_rh{_av}(double **rh);`
 - `void calc_internal_dist{_av}(double **idf);`
 - `void calc_bond_l{_av}(double **bond_l);`
 - `void calc_bond_dist{_av}(double **bdf, int ind_n);`
 - `void calc_g123(double *g1, double *g2, double *g3);`
 - `void calc_g{1|2|3}{_av}(double **g{1|2|3})`
 - `void analyze_formfactor{_av}(..., double **_ff);`
 - `void analyze_rdfchain(..., double **_rdf, ...);`

Analysis

What do you want
to sim today?



- enter here in `statistics.c` from Tcl
- continue to parser / functions in
 - file `statistics.c`:
 - file `statistics_chain.c`,
which require chain information / topology to be set via
`analyze set chain <chain_start> <n_chains> <chain_length>`:
 - `int print_chain_structure_info(...);`
 - `int parse_chain_structure_info(...);`
 - `void update_mol_ids_setchains();`
 - `int check_and_parse_chain_structure_info(...);`
 - in the long run, chain information should instead use `mol_id` of particles

Analysis

*What do you want
to sim today?*



- enter here in `statistics.c` from Tcl
- continue to parser / functions in
 - file `statistics.c`:
 - file `statistics_chain.c`
 - file `statistics_molecule.c`:
 - `int analyze_fold_molecules(...);`
 - `void calc_mol_center_of_mass(Molecule mol, ...);`
 - `void mol_center_of_mass_(Molecule mol, ...);`

Analysis

*What do you want
to sim today?*



- enter here in `statistics.c` from Tcl
- continue to parser / functions in
 - file `statistics.c`:
 - file `statistics_chain.c`
 - file `statistics_molecule.c`
 - file `statistics_cluster.c`:
 - `int parse_necklace_analyzation(...)`

Analysis

*What do you want
to sim today?*



- enter here in `statistics.c` from Tcl
- continue to parser / functions in
 - file `statistics.c`:
 - file `statistics_chain.c`
 - file `statistics_molecule.c`
 - file `statistics_cluster.c`
 - file `pressure.c`:
 - `void pressure_calc(double *result, int v_comp)`
 - `void add_non_bonded_pair_virials(...)`
 - `void add_bonded_virials(Particle *p1)`
 - `void add_kinetic_virials(Particle *p1, int v_comp)`
 - `int parse_and_print_pressure(..., int v_comp)`

 - `int parse_bins(...);`
 - `int parse_and_print_p_IK1(...);`

Analysis

*What do you want
to sim today?*



- enter here in `statistics.c` from Tcl
- continue to parser / functions in
 - file `statistics.c`:
 - file `statistics_chain.c`
 - file `statistics_molecule.c`
 - file `statistics_cluster.c`
 - file `pressure.c`
 - file `energy.c`:
 - (same as for the pressure)

Analysis

What do you want
to sim today?



- enter here in `statistics.c` from Tcl
 - continue to parser / functions in
 - file `statistics.c`
 - file `statistics_chain.c`
 - file `statistics_molecule.c`
 - file `statistics_cluster.c`
 - file `pressure.c`
 - file `energy.c`
 - advantages of current implementation:
 - easy access from Tcl and from C
 - very straightforward to enhance
 - simple structures allow convenient processing of particle data
 - disadvantages:
 - non-parallel code is slow
 - double data storage can quickly become big problem (e.g. trajectories)
 - only partial particle informations available
- master-node
 - copy of particle data (RAM!)
 - parallel (→ integrate 0)

Checkpointing

What do you want
to sim today?



- want to be able to restart at any point, reproducing uninterrupted trajectory
- use Tcl-functions in `scripts/auxiliary.tcl`, i.e.:
 - `checkpoint_set { destination { cnt "all" } { tclvar "all" } { ia "all" } { var "all" } { ran "all" } { COMPACT_CHK 0 } ... }`
 - `checkpoint_read { origin { read_all_chks 1 } { write 0 } { name "anim" } { pdb_sfx 5 }`
- current state being saved using `polyBlockWriteAll` with everything on
- simple textfile contains consecutive list of blockfiles building the trajectory, where suffix of given filename is removed to construct its name
 - e.g. use `checkpoint_set my_sim.[format %03d $i].gz` to get list of (gzipped) blockfiles `my_sim.000.gz`, `my_sim.001.gz`,... and blockfile list `my_sim.chk` in same path
 - simply use `checkpoint_read my_sim.chk` to read all stuff back in
- *everything* can be saved, incl. *internal* state of random number generator
- particle distribution to nodes depends on communication → possible problem

TestSuite

*What do you want
to sim today?*

 **BASF**
The Chemical Company

- built-in self-consistency check for basic functionality
- supposed to be easily usable to ensure frequent employment:
 - `gmake test`
- therefore, output should be self-explanatory
- the following tests are currently implemented
 - `madelung.tcl`, `kinetic.tcl`, `thermostat.tcl`
 - `nve_pe.tcl`, `npt.tcl`, `intpbc.tcl`, `intppbc.tcl`
 - `layered.tcl`, `nsquare.tcl`
 - `lj.tcl`, `lj-cos.tcl`, `fene.tcl`, `harm.tcl`, `gb.tcl`
 - `dh.tcl`, `p3m.tcl`, `mmm1d.tcl`, `el2d.tcl`,
 - `comfixed.tcl`, `comforce.tcl`, `tabulated.tcl`
 - `analysis.tcl`, `pe_micelle.tcl`
 - `constraints.tcl`, `rotation.tcl`, `uwerr_test.tcl`
- tests are automatically blacklisted if unapplicable due to compilation state
- if all tests succeed, ESPResSo is not necessarily bug-free, but if just one of them fails, it's definitely having a problem

SampleSuite

What do you want
to sim today?



- set of “real” test scenarios, preferably reproducing well-known “real” systems (e.g. published long ago, heavily cited, never doubted) with automated analysis for simple comparison to original data
- also serves as set of examples and system showcase
- difference TestSuite ↔ SampleSuite:
 - running time (seconds vs. hours / days / weeks)
 - background (artificial and purely technical setup vs. “real” experiment)
 - employment (use as little as possible vs. use whatever you like)
 - coverage (check few features vs. evaluate entire scenarios)
- if a test case fails, you know exactly where to look (*well, almost always*), if a sample case fails, you just know that *something's* wrong; but: since sample cases aren't artificial, you're more likely to find errors!