ESPResSo

Mesoscopic Molecular Modeling

Analysis, Checkpointing, and the TestSuite

What do you want to sim today?

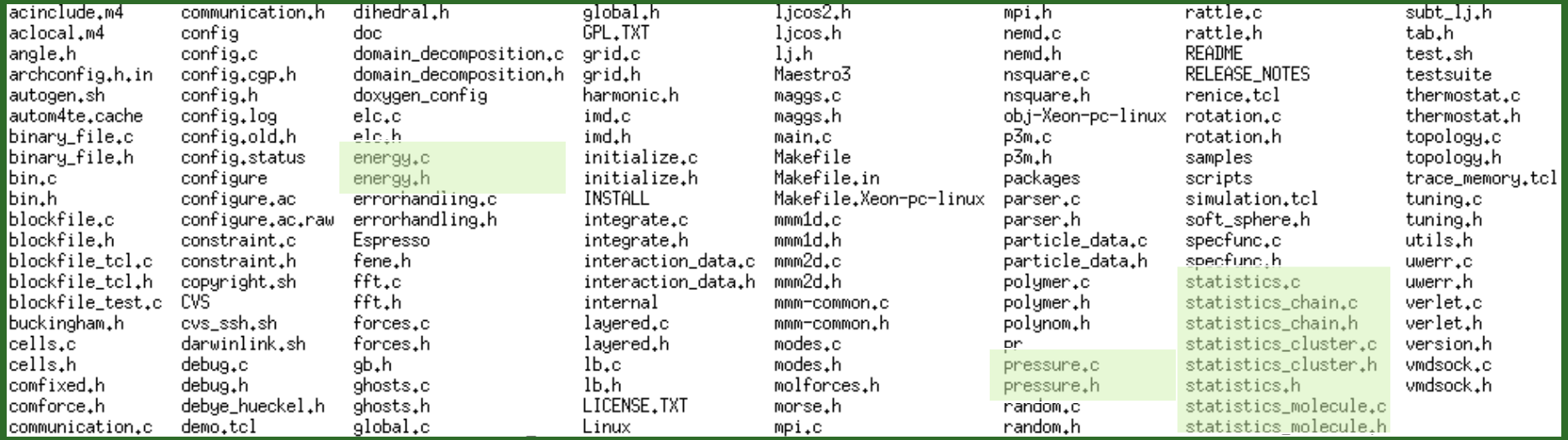
BASF

The Chemical Company

Analysis

What do you want to sim today?

BASF
The Chemical Company



- 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?

BASF
The Chemical Company

- 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

- 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 (...)`

Analysis

- 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

- 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

- 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?

BASF
The Chemical Company

- 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

- 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

- 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

- 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`

  → master-node, copy of particle data (RAM!)
  - file `pressure.c`
  - file `energy.c`

  → parallel (→ `integrate 0`)
- 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

Checkpointing

- 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

- 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

- 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!