CODE NOTES

I. src

1. main:

a. allvars.c: Contains all the global variables.

b. main.c: Start of the program.

Included header files-

main/proto.h

main/allvars.h

Functions-

int main(int argc, char **argv);

Initializes the MPI communication packages, sets CPU time counters to 0. Then, begrun1() is called, which sets up the simulation. Then either the IC's are loaded or the restart files. For IC's: init() is called which prepares the initial conditions for the run, a call to begrun2() finishes the initialization. Then, run() is started, which is the main simulation loop, which iterates over timesteps.

void endrun();

Ends the simulation in case of no error. Has to be called by all processes. Should be used only when the simulation ends without any errors. Else, use terminate().

c. run.c: The main simulation loop.

Included header files-

main/proto.h

main/allvars.h

domain/domain.h

mesh/voronoi/mesh.h

Functions-

void run(void);

Contains the main simulation loop that iterates over single timesteps. The loop terminates when either (a) the cpu-time limit is reached; (b) a 'stop' file is found in the output directory; or (c) when the simulation ends because we arrived at TimeMax.

If the simulation is started from Initial Conditions (a) A Domain decomposition is performed; (b) The gravitational forces are computed; and (c) Voronoi mesh is constructed

The main loop goes as following:

- Find new timesteps: find_timesteps()
- First gravitational half-kick: do_gravity_step_first_half()

- Gradients are calculated (for every cell using Green-gauss gradient estimation): calcualte_gradients()
- Vertex velocities are assigned (determine the speed of the mesh-generating vertices): set_vertex_velocities()
- Computation of the hydro flux (compute intercell flux with Riemann solver and update the cells with the fluxes): compute_interface_fluxes() (first half)
- (de)refinement of hydro cells: do_derefinement_and_refinements()
- Drifting particles to next sync point: find_next_sync_point()

Afterwards, the timebins are updated, so different particles might now be active than before.

- (if needed) a new domain decomposition (will also make a new chained-list of synchronized particles): domain_decomposition()
- Construction of the voronoi mesh: create_mesh()
- Computation of the hydro flux: compute_interface_fluxes() (second half)
- Update of primitive variables (this step closes off the hydro step): update_primitive_variables()

Masses and positions are now updated, we now calculate the new forces and potentials.

- Computation of gravitational forces (closes off gravity half-step): in do_gravity_step_second_half()
- Second gravitational half-kick: do_gravity_step_second_half()
- Do any extra physics, Strang-split (update both primitive and conserved variables as needed): calculate_non_standard_physics_end_of-step()

At the final time, write a restart file, which can be used to continue the simulation beyond the final time. Also output final CPU measurements.

void do_second_order_source_terms_first_half(void);

Source terms before hydrodynamics timestep.

void do_second_order_source_terms_second_half(void);

Source terms after hydrodynamics timestep.

If there are multiple source terms, the order of the second half source term should be applied inverse to the order of the source terms in do_second_order_source_terms_first_half()

void set_non_standard_physics_for_current_time(void);

Calls extra modules after the drift operator.

This routine is called after the active particles are drifted to the next syncpoint, but before a new domain decomposition is performed.

void calculate_non_standard_physics_with_valid_gravity_tree(void);
Calls extra modules after the gravitational force is recomputed.
Only called if a full gravity tree is present.
IMPORTANT: if HIERARCHIAL_GRAVITY is adopted, this function is carried out once per synchronization time, with in general only a partial tree that does not necessarily contain all particles. The latter is the case only for steps where the highest timesteps are active ("full timesteps")
void
calcualte_non_standard_physics_with_valid_gravity_tree_always(void);
Calls extra modules after the gravitational force is recomputed. For runs which have the full tree at teach timestep; no HIERARCHIAL_GRAVITY
void calculate_non_standard_physics_prior_mesh_construction(void);
Calls extra modules before the Voronoi mesh is built.
void calculate_non_standard_physics_end_of_step(void);
Calls extra modules at the end of the run loop. The 2nd gravitational half kick is already applied to the particles and the voronoi mesh is updated.
int check_for_interruption_of_run(void);
Checks whether the run must be interrupted. It does so if (a) stop file is present or (b) 85% of the CPU time is up.
It also handles the regular writing of restart files (can be overwritten).
Returns 1 to interrupt, 0 otherwise.
integertime find_next_outputtime(integertime ti_curr);
Returns the next output time that is equal or larger than ti_curr
void create_end_file(void);
Creates an empty file called 'end' in the output directory. File can be used for eg: analysis scripts to verify that simulation has run up to its final time and ended without error. The file is completely passive.
void execute_resubmit_command(void);
Executes the resubmit command.

2. init:
    a. begrun.c:Initial setup of a simulation run

Contains functions to initialize a simulation run. The parameter file is read in and parsed and global variables are initialized to their proper values.
Included header files-
main/proto.h
main/allvars.h
domain/domain.h
mesh/voronoi/mesh.h

Functions-

void hello(void);

Prints a welcome message.

void begrun0(void);

Prints used compile options.

void begrun1(void);

Initial setup of the simulation.
First, the parameter file is read by read_parameter_file(), a consistency check of parameters is done by check_parameters(), then routines for setting units, etc are called. This function only does the setup necessary to load the IC file. After the IC file has been loaded and prepared by init(), setup continues with begrun2(). This splitting is done so we can cleanly return from operations that don't actually start the simulation like converting snapshots, making projected images, etc.

void begrun2(void);

Late setup, after the IC file has been loaded but before run() is called.
The output files are opened and various modules are initialized. The next output time is determined by find_next_outputtime() and various timers are set.

void set_units(void);

Computes conversion factors internal code units and the cgs-system. Also, constants like the gracitation are set.

void delete_end_file(void);

Deletes the end file if it exists. This is needed in case an already compiled simulation is extended or overwritten. (The end file is completely passive)

b. density.c:SPH density computation and smoothing length determination.
Contains the "first SPH loop", where the SPH densities and smoothing lengths are calculated.
In AREPO, this is used in setup_smoothinglengths() (in init.c) to get an initial guess for MaxDelaunayRadius.
NOTE: SPH densities are NOT used in subsequent hydrodynamics calculations, but the density is either set by the initial conditions explicitly (DENSITY_AS_MASS_IN_INPUT) or calculated by the mass given in the IC divided by the volume of the cell calculated by the Voronoi tessellation algorithm.
Included header files-

main/proto.h

main/allvars.h

domain/domain.h

utils/generic_comm_helpers2.h

struct data_in: Local data structure for collecting particle/cell data that is sent to other processors if needed. Type called data_in and static pointers DataIn and DataGet needed by generic_comm_helpers2.
struct data_out: Local data structure that holds results acquired on remote processors. Type called data_out and static pointers DataResult and DataOut needed by generic_comm_helpers2.
Functions-
void particle2in(data_in *in, int i, int firstnode);
Routine that fills the relevant particle/cll data into the input structure defined above. Needed by generic_comm_helpers2.
in: output param; Data structure to fill.
i: input param; Index of particle im P and SphP arrays.
firstnode: input param; First node of communication.
void out2particle(data_out *out, int i, int mode);
Routine to store or combine result data. Needed by generic_comm_helpers2
out: input param; Data to be moved to appropriate variables in global particle and cell data arrays (P,SphP,...)
i: input param; Index of particle im P and SphP arrays.
mode: input param; Mode of function- local particles or information that was communicated from other tasks and has to be added locally.
void kernel_local(void);
Routine that defines what to do with local particles.
Calls the *_evaluate function in MODE_LOCAL_PARTICLES
void kernel_imported(void);
Routine that defines what to do with imported particles.
Calls the *_evaluate function in MODE_IMPORTED_PARTICLES
void density(void);
Main function of the SPH density calculation.
This function computes the local density for each active SPH particle and the number of weighted neighbors in the current smoothing radius. If a particle with its smoothing region is fully inside the local domain, it is not exported to the other processors. The function also detects particles that have a number of neighbours outside the allowed tolerance range. For these particles, the smoothing length is adjusted accordingly, and the computation is called again.
int density_evaluate(int target, int mode, int threadid);
Inner function of the SPH density calculation.
This function represents the core of the SPH density computation. The target particle may be either local, or reside in the communication buffer.

target: input; Index of particle in local data/import buffer.
mode: input; Mode in which the function is called (local or imported data)
threadid: input; ID of the local thread.
int density_isactive(int n);
Determines if a cell is active at the current timestep.
If the cell is not active in a timestep, its value in TimeBinHydro is negative.
n: input; Index of cell in P and SphP arrays.
Returns 1 if the cell is active or 0 if the cell is inactive or not a cell.

c. init.c:Initialization of a simulation from initial conditions.
Included header files-
main/proto.h
main/allvars.h
domain/domain.h
mesh/voronoi/mesh.h
Functions-
int init(void);
Prepares the loaded initial conditions for the run.
Only called if RestartFlag!=1. Various counters and variables are initialized. Entries of the particle data structures not read from initial conditions are initialized or converted and an initial particle decomposition is performed. If gas cells are present, the initial SPH smoothing lengths are determined.
Returns negative if finished without error and run can start, 0 if code ends after calling init() and positive if an error occurred and need to terminate.
void check_omega(void);
This routine computes the mass content of the box and compares it to the specified value of Omega-matter. If discrepant, the run is terminated.
void setup_smoothinglengths(void);
This function is used to find an initial SPH smoothing length for each cell. It guarantees that the number of neighbours will be between desired_ngb-MAXDEV and desired_ngb+MAXDEV. For simplicity, a first guess of the smoothing length is provided to the function density(), which will then iterate if needed to find the right smoothing length.
void test_id_uniqueness(void);
Checks for unique particle IDs.
The particle IDs are copied to an array and then sorted among all tasks. The array is then checked for duplicates. In that case, the code terminates.

void calculate_maxid(void);
Calculates the global maximum of the IDs of all particles. Needed for REFINEMENT_SPLIT_CELLS
int compare_IDs(const void *a, const void *b);
Comparison function for two MyIDType objects. Used as sorting-kernel for id_uniqueness check. Returns -1 if a<b, 0 if equal, 1 if a>b.

3. utils:
    a. allocate.c: Functions to allocate and reallocate global arrays
    b. debug.c: Print relevant information about a particle / face for debugging
    c. mpz_extension.c: Auxiliary functions to facilitate usage of mpz functions
    d. mymalloc.c: Manager for dynamic memory allocation
    e. parallel_sort.c: MPI parallel sorting routine
    f. predicates.c: Routines for Arbitrary Precision Floating-point Arithmetic and Fast Robust Geometric Predicates
    g. system.c: Small functions for interaction with operating system and libraries and other auxiliary functions
    h. generic_comm_helpers2.h: Generic 'template' MPI communication structure used in many parts of the code
    i. tags.h: Tag defines
    j. timer.h: Timer macros for AREPO.
4. mpi_utils:
    a. checksummed_sendrecv.c: MPI send-receive communication with checksum to verify communication
    b. hypercube_allgatherv.c: Home-made MPI_Allgatherv routine.
    c. mpi_util.c: Custom made auxiliary MPI functions
    d. myalltoall.c: Specialized all-to-all MPI communication functions
    e. myIBarrier.c: Homemade MPI_IBarrier routine
    f. pinning.c: Routine to pin MPI threads to cores.
    g. sizelimited_sendrecv.c: MPI_Sendrecv operation split into chunks of maximum size.
5. io:
    a. global.c: Routines to compute the global statistics of the global state of the code
    b. hdf5_util.c: Contains the wrapper functions to the HDF5 library functions
    c. io.c: Routines for input and output of snapshot files to disk

d. io_fields.c: User defined functions for output; needed for all quantities that are not stored in a global array.
e. logs.c: Log file handling
f. parameters.c: Parses the parameter file
g. read_ic.c: Contains the routines to load the initial conditions
h. restart.c: Handling of the loading/writing of the restart files.

6. domain:
    a. domain.c: Code for domain decomposition.
    b. domain_balance.c: Load-balancing algorithms.
    c. domain_box.c: Routines that determine domain box and do periodic wrapping.
    d. domain_counttogo.c: Functions to determine number of exchanged particles.
    e. domain_DC_update.c: Algorithms for voronoi dynamic update.
    f. domain_exchange.c: Algorithms for exchanging particle data and associated rearrangements.
    g. domain_rearrage.c: Rearranges particle and cell arrays and gets rid of inactive particles.
    h. domain_sort_kernels.c: Comparison and sorting functions for Peano_Hilbert data.
    i. domain_toplevel.c: Top level tree construction and walk routines used for domain decomposition.
    j. domain_vars.c: variables and memory allocation functions for domain decomposition.
    k. peano.c: Order particles along Peano-Hilbert curve.
7. mesh:
    a. voronoi:
        - Voronoi.c: Main file for voronoi-mesh construction.
        - voronoi_1d.c: Routines to build a 1-d voronoi mesh.
        - voronoi_1d_spherical.c: Routines to build a 1-d voronoi mesh in spherical coordinates.
        - voronoi_2d.c: Routines to build a 2-d voronoi mesh.
        - voronoi_3d.c: Routines to build a 3-d voronoi mesh.
        - voronoi_check.c: Algorithms to check voronoi mesh construction.
        - voronoi_derefinement.c: Routines for de-refinement.

- voronoi_dynamic_update.c:Algorithms for voronoi dynamic update
- voronoi_exchange.c:Algorithms that handle communication of voronoi mesh data
- voronoi_ghost_search.c:Algorithms to search for (ghost) cells from other domains
- voronoi_gradients_lsf:Least square fit gradient calculation
- voronoi_gradients_onedims.c:Algorithms to calculate gradients in 1-d simulations.
- voronoi_refinement.c:Routines for refinement
- voronoi_utils.c:Utilities for 3-d voronoi mesh

b. criterion_derefinement.c:Criteria for the de-refinement of a cell
c. criterion_refinement.c:Criteria for the refinement of a cell
d. refinement.c:Driver routines that handle refinement and de-refinement
e. set_vertex_velocities.c:Algorithms that decide how individual cells are moving

8. add_background_grid:
   a. add_bgrid.c: Re-Gridding of the ICs (Initial Conditions) to ensure the entire computational domain contains gas cells. It can be used to convert SPH ICs to Arepo ICs.
      Functions-
      int add_backgroundgrid(void);
      void modify_boxsize(double new_val);
      void prepare_domain_background(void);
      Defined macro ADDBACKGROUNDGRID.
   b. calc_weights.c: Routine that calculates the cumulative weights of neighbouring cells.
   c. distribute.c: Distribute the cell properties in an SPH kernel weighted fashion to neighbouring cells.
9. time_integration:
   a. darkenergy.c:Contains the hubble function for a LCDM cosmology
   b. do_gravity_hydro.c:Contains the two half step kick operators
   c. driftfac.c:Methods for drift and kick pre-factors needed for simulations in a cosmologically expanding box
   d. predict.c:Routines to find the next sync point, manage the list of active timebins/active particles and to drift particles

e. timestep.c:Routines for ‘kicking’ particles in momentum space and assigning new timesteps
f. timestep_treebased.c:Algorithms to compute non-local time-step criterion

10. ngbtree:
    a. ngbtree.c:Construct neighbour tree
    b. ngbtree_search.c:A search routine on the neighbour tree.
    c. ngbtree_walk.c:Routines to walk the ngb tree
11. gravity:
    a. pm:
        - pm_mpi_fft.c:Homemade parallel FFT transforms as needed by the code.
        - pm_nonperiodic.c:Code for non-periodic FFT to compute long-range PM force.
        - pm_periodic.c:Routines for periodic PM-force computation.
        - pm_periodic2d.c:Routines for periodic PM-force computation in 2-D.
    b. accel.c:Routines to carry out gravity force computation.
    c. forcetree.c:Gravitation tree build.
    d. forcetree_ewald.c:Code for Ewald correction (i.e tree force with periodic boundary conditions)
    e. forcetree_optimizebalance.c:Does some preparation work for use of red-black ordered binary trees based on BSD macros.
    f. forcetree_walk.c:Gravitation tree walk code.
    g. gravdirect.c:Main driver routines for gravitational (short-range) force computation through direct summation.
    h. grav_external.c:Special gravity routines for external forces.
    i. grav_softening.c:Routines for setting the gravitational softening length.
    j. gravtree.c:Main driver routines for gravitational (short-range) force computation
    k. gravtree_forcetest.c:Test short range gravity evaluation.
    l. longrange.c:Driver routines for long-range gravitational computation.
12. hydro:
    a. finite_volume_solver.c: Core algorithms of the finite-volume solver
    b. gradients.c:Routines to initialize gradient data.

c. mhd.c:Source terms for MHD implementation needed for cosmological MHD equations as well as Powell source terms
d. riemann.c:Exact, iterative Riemann solver; both adiabatic and isothermal
e. riemann_hllc.c:Routines for a HLLC Riemann solver.
f. riemann_hlld.c:Routines for a HLLD Riemann solver (to be used for in MHD).
g. scalars.c:Routines to initialize passive scalars which are advected with the fluid.
h. update_primitive_variables.c:Routines to recover primitive hydrodynamical variables from the conserved ones.

13. cooling:
    a. cooling.c: Module for gas radiative cooling

14. star_formation:
    a. sfr_eEOS:Star formation rate routines for the effective multi-phase model
    b. starformation.c:Generic creation routines for star particles.

15. subfind:
    a. subfind.c:Main routines of the subfind sub-halo finder
    b. subfind_coll_domain.c:Domain decomposition for collective subfind algorithm
    c. subfind_coll_tree.c:Functions for tree-construction for subfind collective
    d. subdinf_coll_trreewalk.c:Algorithm for collective tree walk; computes gravitational binding energy
    e. subfind_collective.c:Subfind algorithm running collectively on all tasks
    f. subfind_density.c:Smoothing length and density calculation for particles
    g. subfind_distribute.c:Moves groups and particles across MPI tasks from their simulation ordering to a subfind ordering.
    h. subfind_findlinkngb.c:Algorithm to find smoothing lengths of particles to get a desired number of neighbours
    i. subfind_io.c:Main output routine for subfind
    j. subfind_loctree.c:Algorithm for local tree in subfind
    k. subfind_nearesttwo:Neighbour finding of particles in group
    l. subfind_properties.c:Calculation of subgroup properties

m. subfind_reprocess.c:Routines to calculate additional group properties
n. subfind_serial.c:Process the local groups in serial mode
o. subfind_so:Spherical overdensity algorithm for subfind
p. subfind_so_potegy.c:Calculates the potential energy
q. subfind_sort_kernels.c:Comparison functions that serve as sorting kernels for different structs used in subfind
r. subfind_vars.c:Variables for the subfind algorithm

16. fof:

a. fof:Parallel friend of friends (FoF) group finder.
b. fof_distribute.c: Communication and reordering routines for FoF.
c. fof_findgroups.c:Routine to find FoF groups.
d. fof_io.c:Output functions for parallel FoF. Also used by subfind.
e. fof_nearest.c:Routine to find nearest primary link type particle to link secondary link type to FoF groups.
f. fof_sort_kernels.c:Various sort kernels used by parallel FoF group finder
g. fof_vars.c:Instances for the global vars used by FoF, declared in fof.h

17. debug_md5:

a. calc_checksum.c:Functions to calculate an MD5 checksum from dataset.
b. Md5.c:Functions-