Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

PEPC – Code Structure

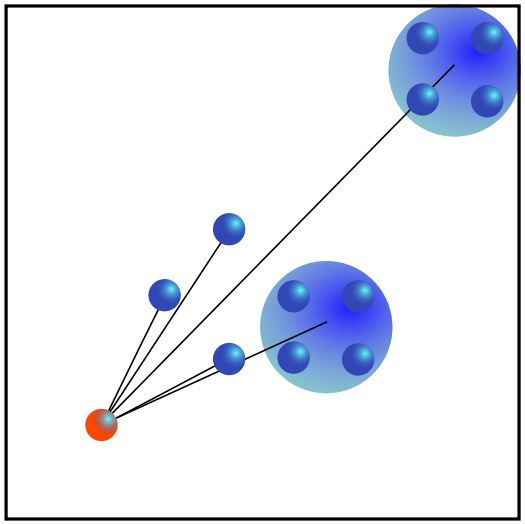
An Overview

April 27, 2012 | Mathias Winkel – Jülich Supercomputing Centre

Barnes-Hut Tree Code Algorithm for the N-body problem

$$\phi_i(\vec{r}_i, t) = \sum_{j \neq i} q_j \frac{1}{|\vec{r}_i - \vec{r}_j|} \qquad \vec{F}(\vec{r}_i, t) = -\sum_{j \neq i} q_j \frac{\vec{r}_i - \vec{r}_j}{|\vec{r}_i - \vec{r}_j|^3}$$

Treecode



$\mathcal{O}(N \log N)$

- straightforward load-balancing for dynamic simulations
- extensibility
- exchange of interaction kernel

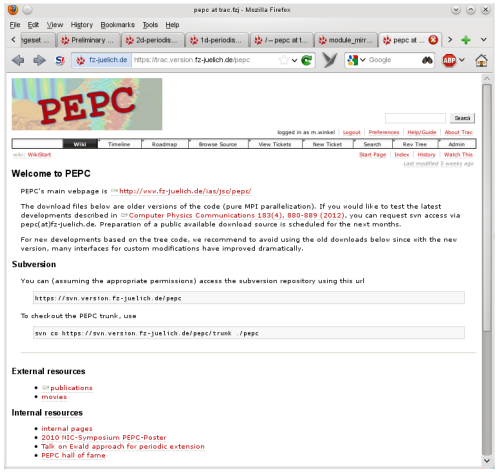
Source Code Access

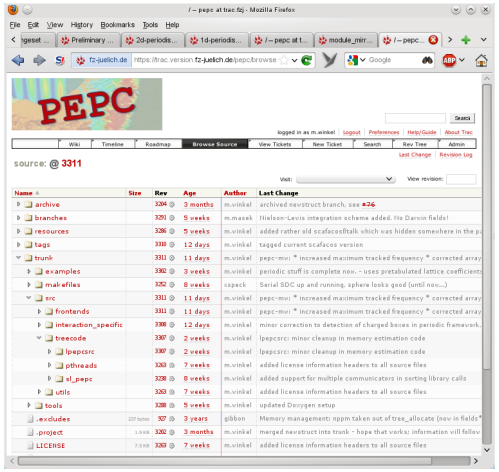
SVN & TRAC

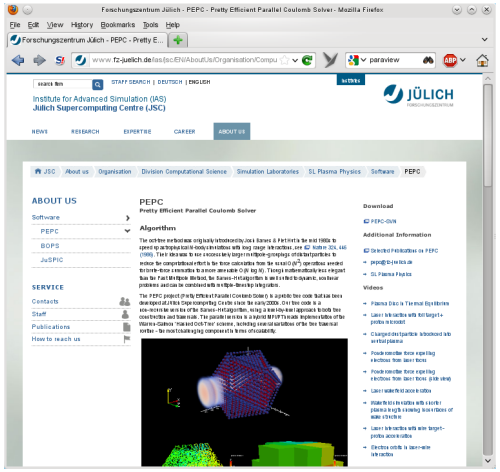
- source code access via SVN:

```
svn co https://svn.version.fz-juelich.de/pepc/trunk ./pepc
```

- account data from Paul / Lukas / Mathias
- trac wiki pages: `https://trac.version.fz-juelich.de/pepc`

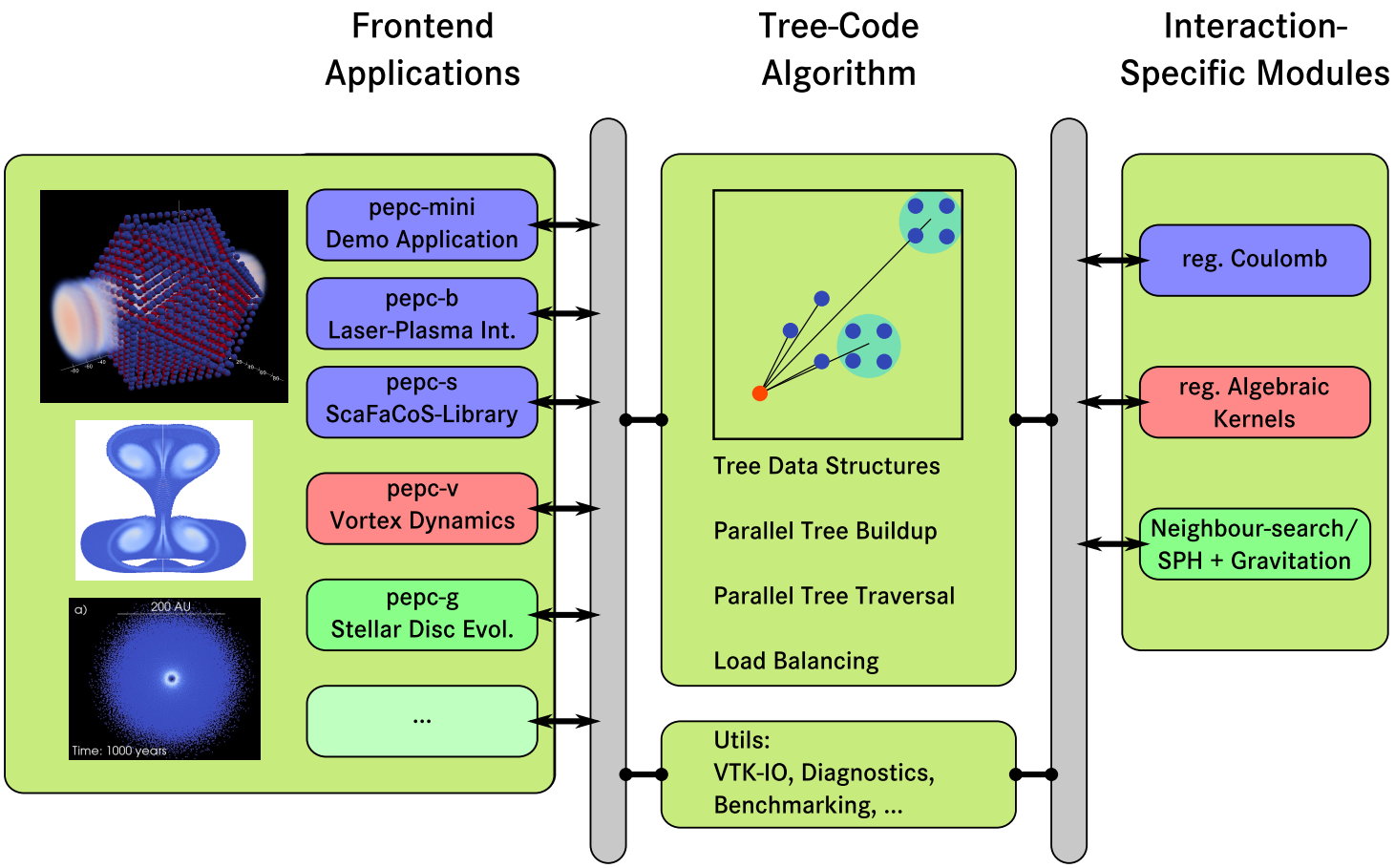






- mailing list for reaching all developers: `pepc@fz-juelich.de`
- PEPC-Website: `www.fz-juelich.de/ias/jsc/pepc` – take a special look at publication links at the bottom of the page

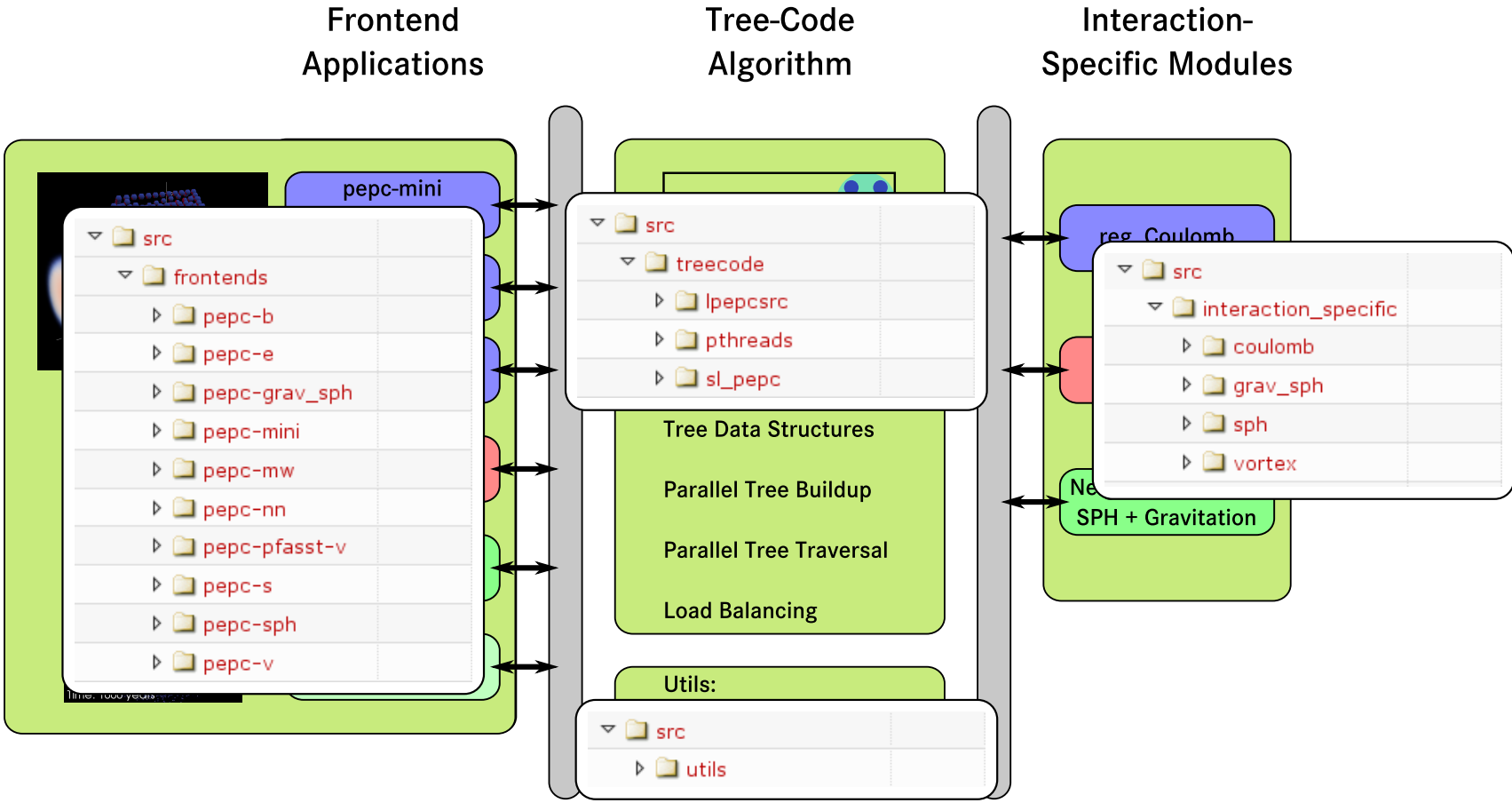
PEPC interface concept



Pretty Efficient Parallel Coulomb Solver: http://www.fz-juelich.de/ias/jsc/pepc

PEPC interface concept

Source code directories



Interaction specific data

Coulomb interaction

/src/interaction_specific/coulomb/module_interaction_specific_types_XYZQ.f90

```
!> Data structure for storing interaction-specific particle data
type t_particle_data
   real*8 :: q
end type t_particle_data

!> Data structure for shipping results
type t_particle_results
   real*8, dimension(3) :: e
   real*8 :: pot
end type t_particle_results

!> Data structure for storing multiple moments of tree nodes
type t_tree_node_interaction_data
  real*8 :: coc(3)      ! centre of charge
  real*8 :: charge      ! net charge sum
  real*8 :: abs_charge  !  absolute charge sum
  real*8 :: dip(3)      ! dipole moment
  real*8 :: quad(3)     ! diagonal quadrupole moments
  real*8 :: xyquad      ! other quadrupole moments
  real*8 :: yzquad
  real*8 :: zxquad
  real*8 :: bmax
end type t_tree_node_interaction_data
```

Registration of MPI types for interaction specific data

Coulomb interaction

/src/interaction_specific/coulomb/module_interaction_specific_types_XYZQ.f90

```
!> Creates and registers interaction-specific MPI-types
!> is automatically called from register_libpepc_mpi_types()

subroutine register_interaction_specific_mpi_types(mpi_type_particle_data,
      MPI_TYPE_tree_node_interaction_data, mpi_type_particle_results)
...
  call MPI_TYPE_STRUCT( nprops_particle_data, blocklengths, displacements,
        types, mpi_type_particle_data, ierr )
  call MPI_TYPE_COMMIT( mpi_type_particle_data, ierr)
...
  call MPI_TYPE_STRUCT( nprops_particle_results, blocklengths, displacements,
        types, mpi_type_particle_results, ierr )
  call MPI_TYPE_COMMIT( mpi_type_particle_results, ierr)
...
  call MPI_TYPE_STRUCT( nprops_tree_node_interaction_data, blocklengths,
        displacements, types, MPI_TYPE_tree_node_interaction_data, ierr )
  call MPI_TYPE_COMMIT( MPI_TYPE_tree_node_interaction_data, ierr)
end subroutine
```

Interaction-specific routines

Coulomb interaction

/src/interaction_specific/coulomb/module_interaction_specific.f90

```
! currently, all public fucns in module_interaction_specific are obligatory
!> Computes multipole properties of a single particle
public multipole_from_particle
!> Accumulates multipole properties of child nodes to parent node
public shift_multipoles_up
!> sums two variables of type(t_particle_result)
public results_add
!> calculation of a single particle-node interaction
public calc_force_per_interaction
!> wrapper for additional external (e.g. periodic) forces
public calc_force_per_particle
!> Multipole Acceptance Criterion
public mac
!> clears result in t_particle datatype
public particleresults_clear
!> Parameter I/O and initialization/finalization
public calc_force_read_parameters
public calc_force_write_parameters
public calc_force_finalize
public calc_force_prepare
public get_number_of_interactions_per_particle
```

How is this used in PEPC itself?

Automatic inclusion of interaction-specific types

src/treecode/lpepcsrc/module_pepc_types.f90

```
!> Data structure for shipping single particles
type t_particle
   real*8 :: x(1:3) !< coordinates
   ...
   integer :: label !< particle label, can be used freely by the
         frontend
   ...
   type(t_particle_data) :: data        !< real physics (charge, etc.)
   type(t_particle_results) :: results !< results of calc_force_etc and
         companions
end type t_particle

! Data structure for shipping multiple moments of child nodes
type t_tree_node_transport_package
   ...
   type(t_tree_node_interaction_data) :: m ! real physics
end type t_tree_node_transport_package
```

- interaction-specific routines are called respectively

Adding own particle data that is only being shipped but not needed for interaction

/src/interaction_specific/coulomb/module_interaction_specific_types_XYZQVM.f90

```
!> Data structure for storing interaction-specific particle data
type t_particle_data
   real*8 :: q
   real*8 :: v(3)
   real*8 :: m
end type t_particle_data

...

! must also modify MPI type registration in
subroutine register_interaction_specific_mpi_types(mpi_type_particle_data,
      MPI_TYPE_tree_node_interaction_data, mpi_type_particle_results)
```

Other interactions need some more modification

/src/interaction_specific/vortex/module_interaction_specific_types.f90

```
!> Data structure for storing interaction-specific particle data
type t_particle_data
   real*8 :: alpha(3)     ! vorticity or better: alpha = vorticity *
         volume
   real*8 :: x_rk(3)      ! position temp array for Runge-Kutta
   real*8 :: alpha_rk(3)  ! vorticity temp array for Runge-Kutta
   real*8 :: u_rk(3)      ! velocity temp array for Runge-Kutta
   real*8 :: af_rk(3)     ! vorticity RHS temp array for Runge-Kutta
end type t_particle_data

!> Data structure for shipping results
type t_particle_results
   real*8, dimension(3) :: u    ! velocities
   real*8, dimension(3) :: af   ! RHS for vorticity/alpha ODE
   real*8 :: div                ! divergence
end type t_particle_results

!> Data structure for storing multiple moments of tree nodes
type t_tree_node_interaction_data
  real*8 :: coc(3)       ! centre of charge
  real*8 :: abs_charge   ! absolute charge sum
  real*8 :: chargex      ! 3D monopole = 3 entries
  real*8 :: chargey
  real*8 :: chargez
  real*8 :: xdip1        ! 3D dipole = 3*3 entries
  real*8 :: ydip1
  real*8 :: zdip1
  real*8 :: xdip2
```

How to invoke the tree code

The complete interface resides in `module_pepc.f90`

`src/treecode/lpepcsrc/module_pepc.f90`

```
public pepc_initialize                  !< mandatory, once per simulation

public pepc_prepare                     !< mandatory, once per simulation
      or after changing internal parameters

public pepc_particleresults_clear       ! usually once per timestep

public pepc_grow_tree                   !< mandatory, once per timestep
public pepc_traverse_tree               !< mandatory, several times per
      timestep with different particles possible
public pepc_statistics                  !< once or never per timestep
public pepc_restore_particles           !< once or never per timestep
public pepc_timber_tree                 !< once or never per timestep

public pepc_grow_and_traverse           !< once per timestep, calls
      pepc_grow_tree, pepc_traverse_tree, pepc_statistics,
      pepc_restore_particles, pepc_timber_tree

public pepc_finalize                    !< mandatory, once per simulation
```

Tree construction

src/treecode/lpepcsrc/module_pepc.f90

```
!> Builds the tree from the given particles, redistributes particles
!> to other MPI ranks if necessary (i.e. reallocates particles and changes
    np_local)

subroutine pepc_grow_tree(np_local, npart_total, particles)
  use module_pepc_types
  use module_libpepc_main
  implicit none
  integer, intent(inout) :: np_local     !< number of particles on this CPU,
        i.e. number of particles in particles-array
  integer, intent(in) :: npart_total !< total number of simulation particles
        (sum over np_local over all MPI ranks)
  type(t_particle), allocatable, intent(inout) :: particles(:) !< input
        particle data, initializes %x, %data, %work appropriately (and
        optionally set %label) before calling this function
...
```

Tree Traversal and Force Evaluation

src/treecode/lpepcsrc/module_pepc.f90

```
!> Traverses the complete tree for the given particles, i.e. computes
!> the field values at their positions. Although missing information
!> is automatically requested from remote MPI ranks, it is important
!> that the particle coordinates fit to the local MPI ranks domain
!> to avoid excessive communication
!> It makes sense to provide the same particles as given/returned
!> from to pepc_grow_tree()

subroutine pepc_traverse_tree(nparticles, particles)
  use module_pepc_types
  use module_libpepc_main
  implicit none
  integer, intent(in) :: nparticles     !< number of particles on this CPU,
        i.e. number of particles in particles-array
  type(t_particle), allocatable, intent(inout) :: particles(:) !< input
        particle data, initializes %x, %data, %work appropriately (and
        optionally set %label) before calling this function
...
```

Restoring particle distribution

src/treecode/lpepcsrc/module_pepc.f90

```
!> Restores the initial particle distribution (before calling
     pepc_grow_tree() ).

subroutine pepc_restore_particles(np_local, particles)
  use module_pepc_types
  use module_libpepc_main
  implicit none
  integer, intent(inout) :: np_local     !< number of particles on this CPU,
        i.e. number of particles in particles-array
  type(t_particle), allocatable, intent(inout) :: particles(:) !< input
        particle data on local MPI rank - is replaced by original particle
        data that was given before calling pepc_grow_tree()
...
```

Invoking PEPC from the frontend application

src/frontends/pepc-mini/pepc.f90

```
program pepc
  ! pepc modules
  use module_pepc
  use module_pepc_types

  type(t_particle), allocatable :: particles(:)

  ! initialize pepc library and MPI
  call pepc_initialize("pepc-mini", my_rank, n_ranks, .true.)

  allocate(particles(1:n_local_particles))
  particles(:)%x(1:3) = [x(:), y(:), z(:)]

  do step=0, n_timesteps
    call pepc_particleresults_clear(particles, n_local_particles)
    call pepc_grow_tree(n_local_particles, n_total_particles, particles)
    call pepc_traverse_tree(n_local_particles, particles)

    ! frontend-defined integrator
    call push_particles(particles)
  end do

  ! cleanup pepc and MPI
  call pepc_finalize()

end program pepc
```

PEPC-e/b – Wrappers for older code

src/interaction_specific/coulomb/module_pepc_wrappers.f90

```
subroutine pepc_fields_coulomb_wrapper(np_local,npart_total,p_x, p_y, p_z,
      p_q, p_w, p_label, p_Ex, p_Ey, p_Ez, p_pot, itime, no_dealloc,
      no_restore, force_const)
    allocate(particles(1:np_local))
   ...
    do i=1,np_local
        particles(i) = t_particle( [p_x(i), p_y(i), p_z(i)], ... )
    end do

    call pepc_particleresults_clear(particles, np_local)
    call pepc_grow_and_traverse(np_local, npart_total, particles, itime,
         no_dealloc, no_restore)

    ! read data from particle_coordinates, particle_results, ...
    do i=1,np_local
      p_ex(i)  = force_const*particles(i)%results%e(1)
                 ...
    end do

end subroutine
```

- do not use this for newly developed frontends

Other Frontends/Backends

Coulomb – Coulomb/Plummer potential for electrostatics/gravitation

PEPC-mini	purely electrostatic, toy example programme, convenient basis for newly developed frontends
PEPC-b	includes ext. magnetic fields, laser-plasma interaction, P. Gibbon
PEPC-mw	purely electrostatic, laser-plasma interaction, M. Winkel
PEPC-s	library version for inclusion in ScaFaCoS
PEPC-e	purely electrostatic, deprecated benchmark version

Vortex – Regularized kernels for fluid simulations

PEPC-v	Vortex-Particle-Method, R. Speck
PEPC-pfasst-v	Vortex-Particle-Method, Parallel-In-Time integration, R. Speck

SPH/Grav_SPH – Nearest-Neighbour Search (+ Gravitation)

PEPC-nn	nearest neighbour search, M. Winkel, A. Breslau
PEPC-sph	Smooth Particle Hydrodynamics (SPH), A. Breslau
PEPC-grav_sph	SPH and gravitation, A. Breslau

Compiling PEPC

- Download source code

```
> svn co https://svn.version.fz-juelich.de/pepc/trunk ./pepc
```

- Prepare build system (choose appropriate `makefile.defs`, adapt it as necessary)

```
> cd ./pepc
> ln -sf makefiles/makefile.defs.SYSTEM ./makefile.defs
```

- Build PEPC-mini (parallel build: `-j`)

```
> make -j pepc-mini
```

- Build other/all frontends, cleanup build environment for treecode, cleanup build even for support libraries

```
> make -j pepc-XYZ
> make -j all
> make clean
> make cleanall
```

- Additional information on currently selected `makefile.defs`

```
> make help
======== make info
==== target architecture : IBM-BGP-jugene
==== code version        : 3305:3311M
==== pepc directory      : /home5/m.winkel/workspace/pepc.newstruct
==== available frontends : pepc-b pepc-e pepc-grav_sph pepc-mini pepc-mw \
     pepc-nn pepc-pfasst-v pepc-s pepc-sph pepc-v
 # Makefile for JuGene
 ...
```

Adding a new frontend/backend

Frontend

- make a copy of the `pepc-mini` subdirectory and name it `pepc-WHATEVER`.
- edit the file `makefile.include` therein appropriately when adding/removing source code files
- the Coulomb-backend is automatically used, for choosing another one, take a look at the `BACKEND`-variable, e.g. in `src/frontends/pepc-v/makefile.include`

Frontend

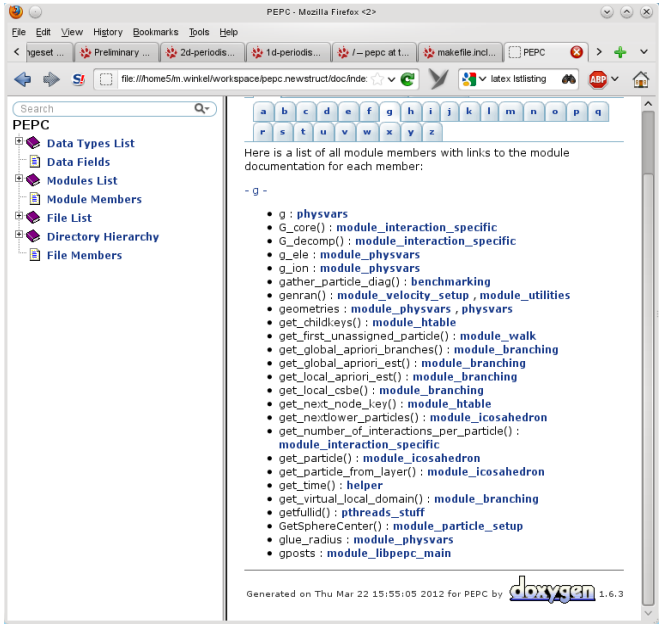
- if you only want to add some particle properties, take a look at `BACKENDTYPE`-variable in `src/interaction_specific/coulomb/makefile.include` and `src/frontends/pepc-mini/makefile.include`
- for even modifying interaction-specific routines:
  - create a copy of the `coulomb`-subdir
  - name it appropriately and perform your modifications therein
  - then select this backend using the `BACKEND` variable in your frontends `makefile.include`

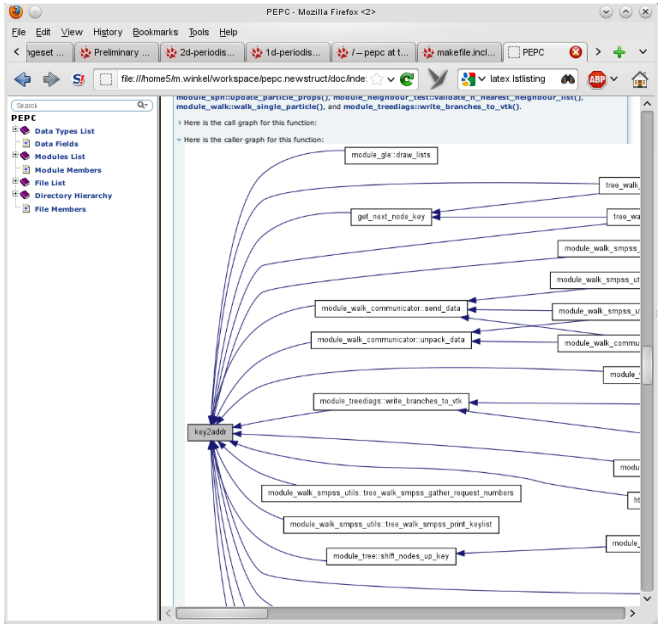
Doxygen documentation

type

```
make doc
firefox ./doc/index.html
```

for creating and viewing a fully-fledged browsable and cross-linked source-code documentation system ☺



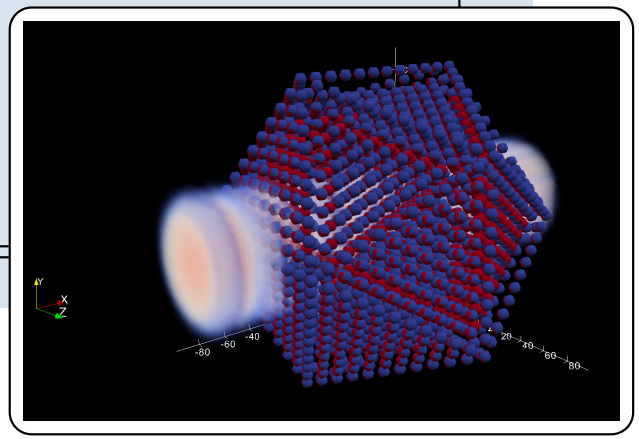


module_vtk

convenient output to parallel vtk files to be visualized using ParaView (http://www.paraview.org/) or VisIt (https://wci.llnl.gov/codes/visit/)

src/frontends/pepc-mini/module_helper.f90

```
call vtk%create_parallel("particles", step, my_rank, n_ranks, time, vtk_step)
call vtk%write_headers(np, 0)
call vtk%startpoints()
call vtk%write_data_array("xyz", np, p(:)%x(1), p(:)%x(2), p(:)%x(3))
call vtk%finishpoints()
call vtk%startpointdata()
call vtk%write_data_array("el_field", np, p(:)%results%e(1), \
p(:)%results%e(2), p(:)%results%e(3))
call vtk%write_data_array("el_pot", np, p(:)%results%pot)
call vtk%write_data_array("charge", np, p(:)%data%q)
...
call vtk%finishpointdata()
call vtk%dont_write_cells()
call vtk%write_final()
call vtk%close()
```

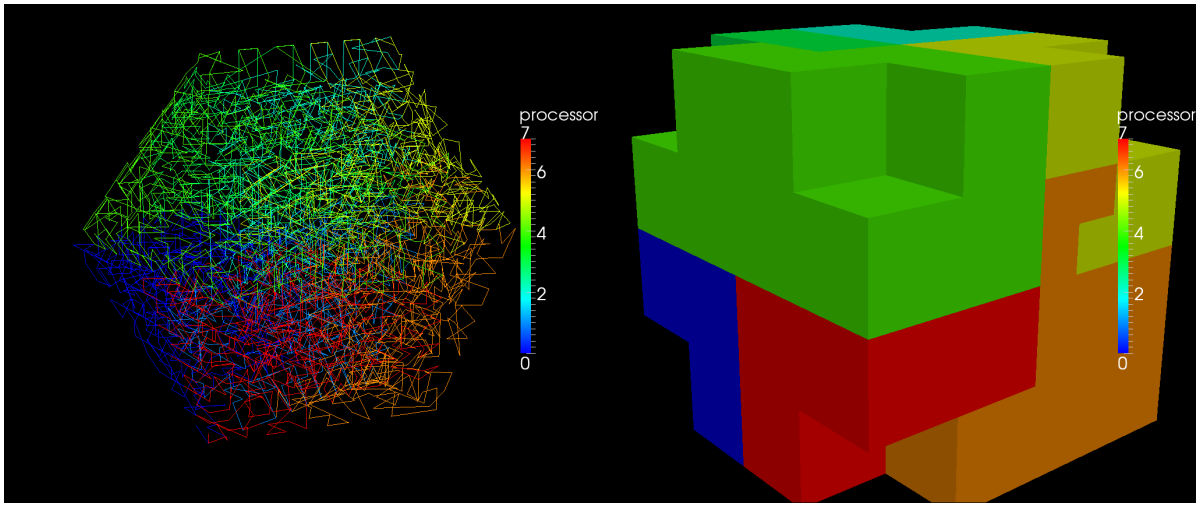


module_treediags

convenient output of tree structures to vtk files

src/frontends/pepc-mini/module_helper.f90

```
use module_vtk
use module_treediags
...
call write_branches_to_vtk(step, dt * step, vtk_step)
call write_spacecurve_to_vtk(step, dt * step, vtk_step, p)
```



module_directsum

direct $\mathcal{O}(N^2)$ force summation for precision tests etc.

src/utils/module_directsum.f90

```
!> direct computation of force onto a selection of local particles
!> due to contributions of all (also remote) other particles
!> MPI- and OpenMP-parallel

subroutine directforce(particles, np_local, testidx, ntest, directresults,
      my_rank, n_cpu, comm)
```

module_checkpoint

functions for checkpointing and restarting using MPI-I/O

src/utils/module_directsum.f90

```
subroutine read_particles_mpiio(itime_in, comm, my_rank, n_cpu, itime,
      np_local, n_total, dp, filename)
subroutine write_particles_mpiio(comm, my_rank, itime, np_local, n_total, dp,
      filename)
```

- see functions `read/write_particles_type()` in `src/frontends/pepc-mw/module_diagnostics.f90` for usage example

module_mirror_boxes, module_fmm_framework

periodic boundary conditions

src/utils/module_mirror_boxes

- nearest image method
- automatically applies to all backends
- arbitrary number of image boxes

src/utils/module_mirror_boxes

- full periodicity
- only for `coulomb`-backend

- see frontend `pepc-s` for usage example

Input parameter files

Initialization of internal parameters

- conveniently done through FORTRAN-namelists:
  - frontend: `pepcb`, ...
  - backend: `calc_force_coulomb`, ...
  - treecode: `libpepc`
  - tree traversal: `walk_para_pthreads`, ...
- just search for `namelist` in the source code
- automatic parsing of all relevant sections of input file (except frontend) via `module_pepc` function

```
subroutine pepc_read_parameters_from_file_name(filename)
```

- excerpt from exemplary input file

```
&pepcmw
    ...
/
&calc_force_coulomb
  ! 3D coulomb
  force_law   = 3
  ! BH-mac
  mac_select = 0
  ! theta = 0.3
  theta2      = 0.09
/
```

```
&libpepc
 debug_level = 0
 np_mult = -300
 interaction_list_length_factor = 8
/

&walk_para_pthreads
 num_walk_threads          = 8
 max_particles_per_thread = 2000
/
```

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

PEPC – Code Structure

An Overview

April 27, 2012 | Mathias Winkel – Jülich Supercomputing Centre