LA-UR-06-7048

Approved for public release;
distribution is unlimited.

Title: *Quick Reference Guide: PFLOTRAN 1.0 (LA-CC 06-093)*
Multiphase-Multicomponent-Multiscale Massively Parallel
Reactive Transport Code

Author(s): SciDAC-2 Project (PI: Peter C. Lichtner, lichtner@lanl.gov)

Contacts: Glenn Hammond (glenn.hammond@pnnl.gov)
Richard Mills (rmills@ornl.gov)

Date: April 25, 2008

DRAFT

TABLE OF CONTENTS

1 Introduction

PFLOTRAN solves a system of generally nonlinear partial differential equations describing multiphase, multicomponent and multiscale reactive flow and transport in porous materials. The code is designed to run on massively parallel computing architectures as well as workstations and labtops. Parallelization is achieved through domain decomposition using the PETSc (Portable Extensible Toolkit for Scientific Computation) libraries for the parallelization framework (Balay et al., 1997).

2 Installation

2.1 Openmpi

Set environment variables `PKGS` and `MPI_HOME` and the appropriate `PATH`:

```
setenv PKGS /Users/lichtner/petsc/packages

setenv MPI_HOME $PKGS/openmpi/openmpi-1.2.5-gcc-4.3.0-absoft-10.1

setenv PATH \$PKGS/openmpi/openmpi-1.2.5-gcc-4.3.0-absoft-10.1:\$PATH

setenv F90 f90

setenv F77 'f90 -YEXT_NAMES=LCS -YEXT_SFX= -f'

setenv FC 'f90 -YEXT_NAMES=LCS -YEXT_SFX= -f'

setenv CC gcc
```

Configure using:

```
./configure --prefix=$PKGS/openmpi/openmpi-1.2.5-gcc-4.3.0-absoft-10.1
```

Finally, compile, check installation and install:

```
make

make check

make install
```

2.2 PETSc

PFLOTRAN uses the Developer version of PETSc. To install PETSc first set the enviroment variables PETSC_DIR and PETSC_ARCH:

```
setenv PETSC_DIR /Users/lichtner/petsc/petsc-dev

setenv PETSC_ARCH Intel_MacOSX10.4.11
```

Configure PETSc on a Mac using openmpi and Fortran 90 Absoft 10.1:

```
./config/configure.py
--with-blas-lapack-lib="-framework vecLib"
--with-mpi-dir=$PKGS/openmpi/openmpi-1.2.5-gcc-4.3.0-absoft-10.1
--with-debugging=0
--with-shared=0
```

Compile and test the PESTc installation with:

```
make all test
```

Optionally install PETSc:

```
make install
```

2.3 HDF5

To install HDF5 set the following environment variables:

```
setenv HDF5_INCLUDE $PKGS/hdf/hdf5-1.6.7-gcc-4.3.0-absoft-10.1/include

setenv HDF5_LIB $PKGS/hdf/hdf5-1.6.7-gcc-4.3.0-absoft-10.1/lib

setenv CC $MPI_HOME/bin/mpicc

setenv F9X $MPI_HOME/bin/mpif90

setenv CFLAGS -fno-strict-aliasing

setenv FFLAGS ""
```

```
./configure --enable-fortran
--prefix=$PKGS/hdf/hdf5-1.6.7-gcc-4.3.0-absoft-10.1
--disable-debug --enable-production --enable-parallel
--enable-static --disable-shared
```

```
make

make check

make install
```

2.4 PFLOTRAN

Compile PFLOTRAN using the command

```
make [hdf5=1] pflotran
```

Create input file `pflotran.in` and run PFLOTRAN with the command:

```
mpirun -n #proc pflotran
```

where `#proc` is the desired number of processor cores.

3 PFLOTRAN Objects

This section gives an overview in alphabetical order of the objects and their data structures used in PFLOTRAN. The upper most object is **Simulation** followed by **Realization**, followed by **Level**, **Patch** and **Grid**.

3.1 Condition

```
type, public :: condition_dataset_type
  PetscInt :: rank
  logical :: is_transient
  logical :: is_cyclic
  PetscInt :: interpolation_method
  PetscReal, pointer :: times(:)
  PetscReal, pointer :: values(:,:)
  PetscReal, pointer :: cur_value(:)
  PetscInt :: cur_time_index
  PetscInt :: max_time_index
end type condition_dataset_type

type, public :: condition_type
  PetscInt :: id                    ! id from which condition can be referenced
```

```
    character(len=MAXWORDLENGTH) :: class ! character string describing class of
                                            condition
    PetscInt :: iclass                            ! integer id for class
    logical :: sync_time_with_update
    character(len=MAXWORDLENGTH) :: name ! name of condition (e.g. initial, recharge)
    PetscInt :: num_sub_conditions
    PetscInt :: iphase
    PetscInt, pointer :: itype(:)
    character(len=MAXWORDLENGTH) :: time_units
    character(len=MAXWORDLENGTH) :: length_units
    type(sub_condition_type), pointer :: pressure
    type(sub_condition_type), pointer :: temperature
    type(sub_condition_type), pointer :: concentration
    type(sub_condition_type), pointer :: enthalpy
    type(sub_condition_ptr_type), pointer :: sub_condition_ptr(:)
    type(condition_type), pointer :: next ! pointer to next condition_type for
                                            linked-lists
  end type condition_type

  type, public :: sub_condition_type
    PetscInt :: itype                 ! integer describing type of condition
    character(len=MAXWORDLENGTH) :: ctype ! character string describing type of
                                            condition
    character(len=MAXWORDLENGTH) :: units     ! units

    type(condition_dataset_type) :: datum
    type(condition_dataset_type) :: gradient
    type(condition_dataset_type) :: dataset

  end type sub_condition_type

  type, public :: sub_condition_ptr_type
    type(sub_condition_type), pointer :: ptr
  end type sub_condition_ptr_type

  type, public :: condition_ptr_type
    type(condition_type), pointer :: ptr
  end type condition_ptr_type

  type, public :: condition_list_type
    PetscInt :: num_conditions
    type(condition_type), pointer :: first
    type(condition_type), pointer :: last
```

```
    type(condition_ptr_type), pointer :: array(:)
  end type condition_list_type
```

3.2 Connection

```
  type, public :: connection_set_type
    PetscInt :: id
    PetscInt :: itype                  ! connection type (boundary, internal,
                                         source sink
    PetscInt :: num_connections
    PetscInt, pointer :: id_up(:)      ! list of ids of upwind cells
    PetscInt, pointer :: id_dn(:)      ! list of ids of downwind cells
    PetscReal, pointer :: dist(:,:)    ! list of distance vectors,
                                         size(-1:3,num_connections) where
                                      !   -1 = fraction upwind
                                      !   0 = magnitude of distance
                                      !   1-3 = components of unit vector
    PetscReal, pointer :: area(:)        ! list of areas of faces normal to
                                           distance vectors
!    PetscReal, pointer :: velocity(:,:)  ! velocity scalars for each phase
    type(connection_set_type), pointer :: next
  end type connection_set_type


  ! pointer data structure required for making an array of region pointers in F90
  type, public :: connection_set_ptr_type
    type(connection_set_type), pointer :: ptr           ! pointer to the
                                                          connection_set_type
  end type connection_set_ptr_type

  type, public :: connection_set_list_type
    PetscInt :: num_connection_objects
    type(connection_set_type), pointer :: first
    type(connection_set_type), pointer :: last
    type(connection_set_ptr_type), pointer :: array(:)
  end type connection_set_list_type
```

3.3 Coupler

```
  type, public :: coupler_type
```

```
    PetscInt :: id                                      ! id of coupler
    PetscInt :: itype                                   ! integer defining type
    character(len=MAXWORDLENGTH) :: ctype        ! character string defining type
    character(len=MAXWORDLENGTH) :: condition_name      ! character string
                                         defining name of condition to be applied
    character(len=MAXWORDLENGTH) :: region_name         ! character string
                                         defining name of region to be applied
    PetscInt :: icondition          ! id of condition in condition array/list
    PetscInt :: iregion             ! id of region in region array/list
    PetscInt :: iface                                ! for structured grids only
    PetscInt, pointer :: aux_int_var(:,:) ! auxilliary array for integer value
    PetscReal, pointer :: aux_real_var(:,:) ! auxilliary array for real values
    type(condition_type), pointer :: condition          ! pointer to condition in
                                                          condition array/list
    type(region_type), pointer :: region                ! pointer to region in
                                                          region array/list
    type(connection_type), pointer :: connection        ! pointer to an array/list
                                                          of connections
    type(coupler_type), pointer :: next                 ! pointer to next coupler
  end type coupler_type

  type, public :: coupler_ptr_type
    type(coupler_type), pointer :: ptr
  end type coupler_ptr_type

  type, public :: coupler_list_type
    PetscInt :: num_couplers
    type(coupler_type), pointer :: first
    type(coupler_type), pointer :: last
    type(coupler_ptr_type), pointer :: array(:)
  end type coupler_list_type
```

3.4 Grid

```
  type, public :: grid_type

    character(len=MAXWORDLENGTH) :: ctype
    PetscInt :: itype  ! type of grid (e.g. structured, unstructured, etc.)

    PetscInt :: nmax   ! Total number of nodes in global domain
    PetscInt :: nlmax  ! Total number of non-ghosted nodes in local domain.
    PetscInt :: ngmax  ! Number of ghosted & non-ghosted nodes in local domain.
```

```
    !nL2G :  not collective, local processor: local  =>  ghosted local
    !nG2L :  not collective, local processor:  ghosted local => local
    !nG2N :  collective,  ghosted local => global index , used for
    !                     matsetvaluesblocked ( not matsetvaluesblockedlocal)
    !nL2A :   collective, local => natural index, used for initialization
    !                              and source/sink setup
    PetscInt, pointer :: nL2G(:), nG2L(:), nL2A(:)
    PetscInt, pointer :: nG2A(:)

    PetscReal, pointer :: x(:), y(:), z(:)

    PetscReal :: x_min, x_max, y_min, y_max, z_min, z_max

    PetscInt, pointer :: hash(:,:,:)
    PetscInt :: num_hash_bins

    type(structured_grid_type), pointer :: structured_grid
    type(unstructured_grid_type), pointer :: unstructured_grid

    type(connection_list_type), pointer :: internal_connection_list

  end type grid_type
```

3.5 Level

```
  type, public :: level_type

    PetscInt :: id
    type(patch_list_type), pointer :: patch_list
    type(level_type), pointer :: next

  end type level_type

  ! pointer data structure required for making an array of level pointers in F90
  type, public :: level_ptr_type
    type(level_type), pointer :: ptr           ! pointer to the level_type
  end type level_ptr_type

  type, public :: level_list_type
    PetscInt :: num_level_objects
    type(level_type), pointer :: first
```

```
    type(level_type), pointer :: last
    type(level_ptr_type), pointer :: array(:)
  end type level_list_type
```

3.6 Material

```
  type, public :: material_type
    PetscInt :: id
    character(len=MAXWORDLENGTH) :: name
    PetscReal :: permeability(3,3)
    PetscReal :: permeability_pwr
    PetscReal :: porosity
    PetscReal :: tortuosity
    PetscInt :: ithrm
    PetscInt :: icap
    type(material_type), pointer :: next
  end type material_type

  type, public :: material_ptr_type
    type(material_type), pointer :: ptr
  end type material_ptr_type

  type, public :: thermal_property_type
    PetscInt :: id
    PetscReal :: rock_density
    PetscReal :: spec_heat
    PetscReal :: therm_cond_dry
    PetscReal :: therm_cond_wet
    PetscReal :: pore_compress
    PetscReal :: pore_expansivity
    PetscReal :: tort_bin_diff
    PetscReal :: vap_air_diff_coef
    PetscReal :: exp_binary_diff
    PetscReal :: enh_binary_diff_coef
    type(thermal_property_type), pointer :: next
  end type thermal_property_type

  type, public :: saturation_function_type
    PetscInt :: id
    character(len=MAXWORDLENGTH) :: saturation_function_ctype
    PetscInt :: saturation_function_itype
    character(len=MAXWORDLENGTH) :: permeability_function_ctype
```

```
    PetscInt :: permeability_function_itype
    PetscReal, pointer :: Sr(:)
    PetscReal :: m
    PetscReal :: lambda
    PetscReal :: alpha
    PetscReal :: pcwmax
    PetscReal :: betac
    PetscReal :: power
    PetscInt :: ihist
    PetscReal :: BC_pressure_low
    PetscReal :: BC_pressure_high
    PetscReal :: BC_spline_coefficients(4)
    type(saturation_function_type), pointer :: next
  end type saturation_function_type

  type, public :: saturation_function_ptr_type
    type(saturation_function_type), pointer :: ptr
  end type saturation_function_ptr_type
```

3.7 Option

```
  type, public :: option_type

    PetscMPIInt :: myrank                    ! rank in PETSC_COMM_WORLD
    PetscMPIInt :: commsize                  ! size of PETSC_COMM_WORLD

    ! defines the mode (e.g. mph, richards, vadose, etc.
    character(len=MAXWORDLENGTH) :: flowmode
    PetscInt :: iflowmode
    character(len=MAXWORDLENGTH) :: tranmode
    PetscInt :: itranmode

    PetscInt :: nphase
    PetscInt :: nflowdof
    PetscInt :: nspec

    PetscInt :: ntrandof
    PetscInt :: ncomp

    PetscReal :: uniform_velocity(3)

    ! Program options
```

```
    PetscTruth :: use_matrix_free  ! If true, do not form the Jacobian.

    PetscInt :: imod

    PetscTruth :: use_isoth

    character(len=MAXWORDLENGTH) :: generalized_grid
    logical :: use_generalized_grid

    PetscReal :: flow_time, tran_time, time  ! The time elapsed in the simulation.
    PetscReal :: flow_dt, tran_dt, dt ! The size of the time step.

!    PetscReal, pointer :: tplot(:)
    PetscReal, pointer :: tfac(:)
      ! An array of multiplicative factors that specify how to increase time step.

    PetscInt :: iblkfmt ! blocked format

      ! Basically our target number of newton iterations per time step.
    PetscReal :: dpmxe,dtmpmxe,dsmxe,dcmxe !maximum allowed changes in field vars.
    PetscReal :: dpmax,dtmpmax,dsmax,dcmax

    PetscReal :: scale
    PetscReal, pointer :: rock_density(:),cpr(:),dencpr(:),ckdry(:),ckwet(:), &
                          tau(:),cdiff(:),cexp(:)
    PetscReal, pointer :: swir(:),lambda(:),alpha(:),pckrm(:),pcwmax(:),pcbetac(:), &
                          pwrprm(:),sir(:,:)
    PetscInt, pointer:: icaptype(:)

    PetscReal :: m_nacl
    PetscReal :: difaq, delhaq, gravity(3), fmwh2o= 18.0153D0, fmwa=28.96D0, &
              fmwco2=44.0098D0, eqkair, ret=1.d0, fc=1.d0

    PetscInt :: ideriv
    PetscReal :: tref,pref

    PetscReal :: disp

!   table lookup
    PetscInt :: itable=0

    PetscTruth :: restart_flag
    PetscReal :: restart_time
```

```
    character(len=MAXWORDLENGTH) :: restart_file
    PetscTruth :: checkpoint_flag
    PetscInt :: checkpoint_frequency

    PetscLogDouble :: start_time
    PetscTruth :: wallclock_stop_flag
    PetscLogDouble :: wallclock_stop_time

    PetscInt :: log_stage(10)

    logical :: numerical_derivatives
    logical :: compute_statistics
    logical :: use_touch_options
    logical :: overwrite_restart_transport
    PetscInt :: io_handshake_buffer_size

    character(len=MAXWORDLENGTH) :: permx_filename
    character(len=MAXWORDLENGTH) :: permy_filename
    character(len=MAXWORDLENGTH) :: permz_filename

  end type option_type

  type, public :: output_option_type

    character(len=2) :: tunit
    PetscReal :: tconv

    logical :: print_hdf5
    logical :: print_hdf5_velocities
    logical :: print_hdf5_flux_velocities

    logical :: print_tecplot
    logical :: print_tecplot_velocities
    logical :: print_tecplot_flux_velocities

    PetscInt :: plot_number
    character(len=MAXWORDLENGTH) :: plot_name

  end type output_option_type
```

3.8 Patch

```
type, public :: patch_type

  PetscInt :: id

  ! thiese arrays will be used by all modes, mode-specific arrays should
  ! go in the auxilliary data stucture for that mode
  PetscInt, pointer :: imat(:)
  PetscReal, pointer :: internal_velocities(:,:)
  PetscReal, pointer :: boundary_velocities(:,:)

  type(grid_type), pointer :: grid

  type(region_list_type), pointer :: regions

  type(coupler_list_type), pointer :: transport_boundary_conditions
  type(coupler_list_type), pointer :: transport_initial_conditions
  type(coupler_list_type), pointer :: transport_source_sinks

  type(coupler_list_type), pointer :: flow_boundary_conditions
  type(coupler_list_type), pointer :: flow_initial_conditions
  type(coupler_list_type), pointer :: flow_source_sinks

  type(strata_list_type), pointer :: strata
  type(breakthrough_list_type), pointer :: breakthrough

  type(auxilliary_type) :: aux

  type(patch_type), pointer :: next

end type patch_type

! pointer data structure required for making an array of patch pointers in F90
type, public :: patch_ptr_type
  type(patch_type), pointer :: ptr           ! pointer to the patch_type
end type patch_ptr_type

type, public :: patch_list_type
  PetscInt :: num_patch_objects
  type(patch_type), pointer :: first
  type(patch_type), pointer :: last
  type(patch_ptr_type), pointer :: array(:)
```

```
end type patch_list_type
```

3.9 Realization

```
type, public :: realization_type

  type(discretization_type), pointer :: discretization
  type(level_list_type), pointer :: level_list
  type(patch_type), pointer :: patch

  type(option_type), pointer :: option
  type(field_type), pointer :: field
  type(pflow_debug_type), pointer :: debug
  type(output_option_type), pointer :: output_option

  type(region_list_type), pointer :: regions
  type(condition_list_type), pointer :: flow_conditions
  type(condition_list_type), pointer :: transport_conditions

  type(material_type), pointer :: materials
  type(material_ptr_type), pointer :: material_array(:)
  type(thermal_property_type), pointer :: thermal_properties
  type(saturation_function_type), pointer :: saturation_functions
  type(saturation_function_ptr_type), pointer :: saturation_function_array(:)

  type(waypoint_list_type), pointer :: waypoints

end type realization_type
```

3.10 Region

```
type, public :: block_type
  PetscInt :: i1,i2,j1,j2,k1,k2
  type(block_type), pointer :: next
end type block_type

type, public :: region_type
  PetscInt :: id
  character(len=MAXWORDLENGTH) :: name
  character(len=MAXWORDLENGTH) :: filename
```

```
    PetscInt :: i1,i2,j1,j2,k1,k2
    PetscReal :: coordinate(3)
    PetscInt :: iface
    PetscInt :: num_cells
    PetscInt, pointer :: cell_ids(:)
    PetscInt, pointer :: faces(:)
    type(region_type), pointer :: next
  end type region_type

  type, public :: region_ptr_type
    type(region_type), pointer :: ptr
  end type region_ptr_type

  type, public :: region_list_type
    PetscInt :: num_regions
    type(region_type), pointer :: first
    type(region_type), pointer :: last
    type(region_type), pointer :: array(:)
  end type region_list_type
```

3.11 Simulation

```
  type, public :: simulation_type

    type(realization_type), pointer :: realization
    type(stepper_type), pointer :: flow_stepper
    type(stepper_type), pointer :: tran_stepper

  end type simulation_type
```

3.12 Solver

```
  type, public :: solver_type
    PetscReal :: linear_atol       ! absolute tolerance
    PetscReal :: linear_rtol       ! relative tolerance
    PetscReal :: linear_dtol       ! divergence tolerance
    PetscInt :: linear_maxit     ! maximum number of iterations

    PetscReal :: newton_atol       ! absolute tolerance
    PetscReal :: newton_rtol       ! relative tolerance
```

```
    PetscReal :: newton_stol       ! relative tolerance (relative to previous
                                     iteration)
    PetscReal :: newton_dtol       ! divergence tolerance
    PetscReal :: newton_inf_res_tol    ! infinity tolerance for residual
    PetscReal :: newton_inf_upd_tol    ! infinity tolerance for update
    PetscInt :: newton_maxit     ! maximum number of iterations
    PetscInt :: newton_maxf      ! maximum number of function evaluations

        ! Jacobian matrix
    Mat :: J
    MatFDColoring :: matfdcoloring
      ! Coloring used for computing the Jacobian via finite differences.

    ! PETSc nonlinear solver context
    SNES :: snes
    KSPType :: ksp_type
    PCType  :: pc_type
    KSP   ::  ksp
    PC    ::  pc

    PetscTruth :: inexact_newton

    PetscTruth :: print_convergence
    PetscTruth :: print_detailed_convergence
    PetscTruth :: check_infinity_norm
    PetscTruth :: force_at_least_1_iteration

  end type solver_type
```

3.13 Stepper

```
  type, public :: stepper_type

    PetscInt :: steps          ! The number of time-steps taken by the code.
    PetscInt :: nstepmax       ! Maximum number of timesteps taken by the code.
    PetscInt :: icut_max       ! Maximum number of timestep cuts within one time step.
    PetscInt :: ndtcmx         ! Steps needed after cutting to increase time step
    PetscInt :: newtcum        ! Total number of Newton steps taken.
    PetscInt :: icutcum        ! Total number of cuts in the timestep taken.
    PetscInt :: iaccel         ! Accelerator index

    PetscReal :: dt_min
```

```
  PetscReal :: dt_max

  type(solver_type), pointer :: solver

  type(waypoint_type), pointer :: cur_waypoint

  type(convergence_context_type), pointer :: convergence_context

end type stepper_type
```

3.14 Strata

```
type, public :: strata_type
  PetscInt :: id                                  ! id of strata
  logical :: active
  character(len=MAXWORDLENGTH) :: material_name  ! character string defining
                                                 name of material to be applied
  character(len=MAXWORDLENGTH) :: region_name    ! character string defining
                                                 name of region to be applied
  PetscInt :: imaterial                ! id of material in material array/list
  PetscInt :: iregion                      ! id of region in region array/list
  type(material_type), pointer :: material       ! pointer to material in
                                                   material array/list
  type(region_type), pointer :: region           ! pointer to region in region
                                                   array/list
  type(strata_type), pointer :: next             ! pointer to next strata
end type strata_type

type, public :: strata_ptr_type
  type(strata_type), pointer :: ptr
end type strata_ptr_type

type, public :: strata_list_type
  PetscInt :: num_strata
  type(strata_type), pointer :: first
  type(strata_type), pointer :: last
  type(strata_ptr_type), pointer :: array(:)
end type strata_list_type
```

4 Creating the Input File: PFLOTRAN Keywords

The PFLOTRAN input file construction is based on keywords. Lines beginning with a colon (:) are treated as comments. Each entry to the input file must begin in the first column. Keywords `SKIP` and `NOSKIP` are used to skip over sections of the input file. Blank lines may occur in input file. Alternate keyword spelling is indicated in round brackets `()`. Input options are indicated in square brackets `[]`, as well as default values. Curly brackets `{}` indicate the result of invoking the corresponding keyword. Always refer to source code when in doubt!

Initial and boundary conditions and material properties are assigned to spatial regions using a novel *coupler* approach. In this approach, initial and boundary conditions (keyword CONDITION) are assigned to regions (keyword REGION) using keywords INITIAL_CONDITION and BOUNDARY_CONDITION. Material properties (keyword MATERIAL) are assigned to regions using the keyword STRATIGRAPHY.

Keyword	Description
BOUNDARY_CONDITION	
BREAKTHROUGH	
BRINE (BRIN)	
CHECKPOINT	
COMPUTE_STATISTICS (STATISTICS)	
CONDITION	
DATASET	
DEBUG	
DIFF	
DTST	
DXYZ	
GRAVITY	
GRID	
HDF5	
IMOD	
INVERT_Z (INVERTZ)	
INITIAL_CONDITION	
LINEAR_SOLVER	
MATERIAL (MATERIALS, PHIK)	
MODE	

Keyword: BOUNDARY_CONDITION

BOUNDARY_CONDITION

REGION region_name

CONDITION condition_name

TYPE [initial, boundary, source_sink]

FACE [WEST, EAST, NORTH, SOUTH, BOTTOM, TOP]

END

Keyword: BREAKTHROUGH (BRK)

BREAKTHROUGH

REGION region_name

VELOCITY {print_velocities == PETSC_TRUE}

(., /, END)

Keyword: BRINE (BRIN)

BRIN, BRINE m_nacl [MOLAL, MASS, MOLE]

Keyword: CHECKPOINT

CHECKPOINT checkpoint_frequency

Keyword: COMPUTE_STATISTICS (STATISTICS)

COMPUTE_STATISTICS, STATISTICS {compute_statistics = .true.}

Keyword: CONDITION (COND)

CONDITION (COND) condition_name

UNITS

s, sec, min, hr, d, day, y, yr

mm, cm, m, met, meter, dm, km

Pa, KPa

m/s, m/yr

C, K

M, mol/L

KJ/mol

(., /, END)

CLASS [flow, transport (tran)]

CYCLIC {is_cyclic = .true.}

INTERPOLATION step linear

TYPE

PRESSURE (PRES, PRESS) [dirichlet, neumann, mass, hydrostatic (hydro, hydrostat), static, zero_gradient, seepage]

FLUX [dirichlet, neumann, mass, hydrostatic (hydro, hydrostat), static, zero_gradient, seepage]

TEMPERATURE (TEMP) [dirichlet, neumann, mass, hydrostatic (hydro, hydrostat), static, zero_gradient, seepage]

CONCENTRATION (CONC) [dirichlet, neumann, mass, hydrostatic (hydro, hydrostat), static, zero_gradient, seepage]

ENTHALPY (H) [dirichlet, neumann, mass, hydrostatic (hydro, hydrostat), static, zero_gradient, seepage]

(., /, END)

TIME

IPHASE

DATUM (DATM)

[Continued]

Keyword: CONDITION (COND) [Continued]

GRADIENT (GRAD)

- PRESSURE (PRES, PRESS)
- FLUX
- TEMPERATURE (TEMP)
- CONCENTRATION (CONC)
- ENTHALPY (H)

(., /, END)

TEMPERATURE (TEMP)

ENTHALPY (H)

PRESSURE (PRES, PRESS)

FLUX (VELOCITY, VEL)

CONCENTRATION (CONC)

(., /, END)

Keyword: DATASET

DATASET	[permx, permy, permz] [permx_filename, permy_filename, permz_filename]

Keyword: DEBUG

DEBUG

PRINT_SOLUTION (VECVIEW_SOLUTION, VIEW_SOLUTION)

PRINT_RESIDUAL (VECVIEW_RESIDUAL,VIEW_RESIDUAL)

PRINT_JACOBIAN (MATVIEW_JACOBIAN, VIEW_JACOBIAN)

PRINT_JACOBIAN_NORM (NORM_JACOBIAN)

PRINT_COUPLERS (PRINT_COUPLER)

PRINT_JACOBIAN_DETAILED (MATVIEW_JACOBIAN_DETAILED, VIEW_JACOBIAN_DETAILED)

PRINT_NUMERICAL_DERIVATIVES (VIEW_NUMERICAL_DERIVATIVES)

END

Keyword: DIFF

DIFF difaq delhaq

Keyword: DTST

DTST dt_min

dt1, dt2, dt3, ..., dt_max

Keyword: DXYZ

DXYZ [STRUCTURED_GRID, AMR_GRID]

dx0

dy0

dz0

Keyword: GRAVITY (GRAV)

GRAVITY (GRAV) gravity

Keyword: GRID

GRID

TYPE [structured, unstructured, amr]

NXYZ nx ny nz

FILE

END

Keyword: HDF5

HDF5 [VELO, FLUX]

Keyword: IMOD

IMOD mod

Keyword: INVERT_Z (INVERTZ)

INVERT_Z (INVERTZ) {invert_z_axis = .true.}

Keyword: INITIAL_CONDITION

```
INITIAL_CONDITION
        REGION region_name
        CONDITION condition_name
        TYPE [initial, boundary, source_sink]
        FACE [WEST, EAST, NORTH, SOUTH, BOTTOM, TOP]
END
```

Keyword: LINEAR_SOLVER

```
LINEAR_SOLVER
        TRAN, TRANSPORT (tran_solver) / DEFAULT (flow_solver)
        SOLVER_TYPE (SOLVER, KRYLOV_TYPE, KRYLOV, KSP, KSP_TYPE)
                NONE (PREONLY)
                GMRES
                BCGS (BICGSTAB, BI-CGSTAB)
        PRECONDITIONER_TYPE (PRECONDITIONER, PC, PC_TYPE)
                ILU (PCILU)
                LU (PCLU)
                BJACOBI (BLOCK_JACOBI)
                ASM (ADDITIVE_SCHWARTZ)
                PCASM
                ATOL
                RTOL
                DTOL
                MAXIT
(., /, END)
```

Keyword: MATERIAL (MATERIALS, PHIK)

MATERIAL (MATERIALS, PHIK)

name id icap ithrm por tor permx permy permz permpwr

(., /, END)

Keyword: MODE

MODE [RICHARDS_LITE, RICHARDS, MPH]

Keyword: NEWTON_SOLVER

NEWTON_SOLVER

TRAN, TRANSPORT (tran_solver) / DEFAULT (flow_solver)

INEXACT_NEWTON

NO_PRINT_CONVERGENCE

NO_INF_NORM (NO_INFINITY_NORM)

NO_FORCE_ITERATION

PRINT_DETAILED_CONVERGENCE

ATOL

RTOL

STOL

DTOL

ITOL (INF_TOL, ITOL_RES, INF_TOL_RES)

ITOL_UPDATE (INF_TOL_UPDATE)

MAXIT

MAXF

(., /, END)

Keyword: NUMERICAL_JACOBIAN

NUMERICAL_JACOBIAN {numerical_derivatives = .true.}

Keyword: ORIGIN (ORIG)

ORIGIN (ORIG) X_DIRECTION Y_DIRECTION Z_DIRECTION

Keyword: OVERWRITE_RESTART_TRANSPORT

OVERWRITE_RESTART_TRANSPORT {overwrite_restart_transport = .true.}

Keyword: REGION

REGION region_name

BLOCK i1 i2 j1 j2 k1 k2

COORDINATE x-coordinate y-coordinate z-coordinate

FILE filename

LIST (not implemented)

FACE [WEST, EAST, NORTH, SOUTH, BOTTOM, TOP]

END

Keyword: RESTART

RESTART restart_file restart_time

Keyword: RICH

RICH pref

Keyword: SATURATION_FUNCTION (SATURATION_FUNCTIONS, PCKR)

SATURATION_FUNCTION (SATURATION_FUNCTIONS, PCKR)

id icaptype [(Sr[np],np=1,nphase), Sr] pckrm alpha pcwmax pbetac pwrprm

(., /, END)

Keyword: SOURCE_SINK

SOURCE_SINK

REGION region_name

CONDITION condition_name

TYPE [initial, boundary, source_sink]

FACE [WEST, EAST, NORTH, SOUTH, BOTTOM, TOP]

END

Keyword: STRATIGRAPHY (STRATA)

STRATIGRAPHY (STRATA)

REGION region_name

MATERIAL material_name

INACTIVE

(., /, END)

Keyword: TECP

TECP [VELO, FLUX]

Keyword: THRM (THERMAL_PROPERTY, THERMAL_PROPERTIES)

THRM (THERMAL_PROPERTY, THERMAL_PROPERTIES)

id rock_density spec_heat therm_cond_dry therm_cond_wet tort_bin_diff vap_air_diff_coef exp_binary_diff

(., /, END)

Keyword: TIME

```
TIME        [s, m, h, d, mo, y] [every #]

            t1, t2, t3, ...
```

Keyword: TIMESTEPPER

```
TIMESTEPPER

            NUM_STEPS_AFTER_TS_CUT [5]

            MAX_STEPS [999999]

            TS_ACCELERATION [5]

            MAX_TS_CUTS [16]

            MAX_PRESSURE_CHANGE [5.d4]

            MAX_TEMPERATURE_CHANGE [5.d0]

            MAX_CONCENTRATION_CHANGE [1.d0]

            MAX_SATURATION_CHANGE [0.5d0]

(., /, END)
```

Keyword: TRAN

```
TRAN        ntrandof
```

Keyword: UNIFORM_VELOCITY

```
UNIFORM_VELOCITY  vlx vly vlz
```

Keyword: USE_TOUCH_OPTIONS

USE_TOUCH_OPTIONS {use_touch_options = .true.}

Keyword: WALLCLOCK_STOP

WALLCLOCK_STOP wallclock_stop_time

Example Input File

```
:Description: 2D problem for saturated layered medium
:
:MODE RICHARDS
MODE RICHARDS_LITE
TRAN 1
:
:NUMERICAL_JACOBIAN
:INEXACT_NEWTON
:USE_TOUCH_OPTIONS
:
:CHECKPOINT 1000
:RESTART steady.chk 0.d0
:OVERWRITE_RESTART_TRANSPORT
:COMPUTE_STATISTICS
:USE_TOUCH_OPTIONS
:WALLCLOCK_STOP 0.d0
:
DEBUG
:MATVIEW_JACOBIAN
:VECVIEW_RESIDUAL
:VECVIEW_SOLUTION
:PRINT_COUPLERS
END
:
GRID
TYPE structured
NXYZ 450 1 4430
END
```

```
:
ORIGIN 0.d0 0.d0 0.d0
:
NEWTON_SOLVER
RTOL 1.d-5
ATOL 1.d-7
STOL 1.d-10
:ITOL_RES 1.d-8
:ITOL_UPDATE 0.05d0  ! Pa
NO_INFINITY_NORM
:NO_FORCE_ITERATION
:NO_PRINT_CONVERGENCE
:PRINT_DETAILED_CONVERGENCE
MAXIT 20
END
:noskip
:
NEWTON_SOLVER TRANSPORT
:RTOL 1.d-50
ATOL 1.d-50
STOL 1.d-50
ITOL_RES 1.d-8
:ITOL_UPDATE 5.d0  ! Pa
:NO_INFINITY_NORM
:NO_FORCE_ITERATION
:NO_PRINT_CONVERGENCE
:PRINT_DETAILED_CONVERGENCE
MAXIT 10
END
:
TIMESTEPPER
TS_ACCELERATION 8
END
:
:HDF5 !VELO !FLUX
TECP VELO !FLUX
:
DXYZ
0.02d0
1.d0
0.002d0
:
:  d0[m^2/s] delhaq[kJ/mol]
```

```
DIFF  1.D-9       12.6
:
: Richards Equation Pref
RICH 101325.
:
SATURATION_FUNCTIONS
: van Genuchten
:id itype swir   m      alpha     pcwmax betac pwr
 1  1     0.1600 0.3391 7.2727d-4 1.e8   0.d0  1.d0
 2  1     0.1299 0.7479 1.4319d-4 1.e8   0.d0  1.d0
: Brooks-Corey
:id itype swir   lambda alpha     pcwmax betac pwr
: 1  2     0.1600 1.97   7.2727d-4 1.e8   0.d0  1.d0
: 2  2     0.1299 0.5193 1.4319d-4 1.e8   0.d0  1.d0
END

THERMAL_PROPERTIES
:ithm rho    cpr     ckdry cksat tau cdiff   cexp
  1   2.76e3 1000.e0 0.5   0.5   0.5 2.13d-5 1.8
END
:
MATERIALS
:name id icap ithm por  tau permx  permy  permz  permpwr
tuff 1  1    1    0.2 0.5 1.d-19 1.d-19 1.d-19 1.d0
END
:
:
:TIME y every 10.
TIME y
0.1 0.25 0.5 0.75 1.
:
DTST 1.d-8
1. 0.001d0
:
:define regions-------------------------------------------------------------
:
REGION all
BLOCK 1 450 1 1 1 4430
END

REGION Left
FACE west
BLOCK 1 1 1 1 3931 4430
```

```
END

REGION Right
FACE east
BLOCK 450 450 1 1 1 500
END

:define initial and boundary conditions---------------------------------------

:flow----------------------

CONDITION initial
CLASS flow
TYPE
PRESSURE hydrostatic
END
DATUM 0.d0 0.d0 10.d0
PRESSURE 101325.d0
END

CONDITION Left
CLASS flow
TYPE
PRESSURE neumann
END
PRESSURE 1.5854896d-7 ! 5000 mm/yr
END

CONDITION Right
CLASS flow
TYPE
PRESSURE neumann
END
PRESSURE -1.5854896d-7 ! 5000 mm/yr
END

:transport-----------------

CONDITION initial_c
CLASS transport
CONCENTRATION 1.d-8
END
```

```
CONDITION outlet_c
CLASS transport
TYPE
CONCENTRATION zero_gradient
END
CONCENTRATION 1.d-8
END

CONDITION inlet_c
CLASS transport
CONCENTRATION 1.d0
END

:set initial and boundary conditions---------------------------------------

:flow-----------------------

: initial condition
INITIAL_CONDITION
CONDITION initial
REGION all
END

BOUNDARY_CONDITION
CONDITION Left
REGION Left
END

BOUNDARY_CONDITION
CONDITION initial
REGION Right
END

:transport------------------

: initial condition
INITIAL_CONDITION
CONDITION initial_c
REGION all
END

BOUNDARY_CONDITION
CONDITION inlet_c
```

```
REGION Left
END

BOUNDARY_CONDITION
CONDITION outlet_c
REGION Right
END

:set material properties---------------------------------------------------

STRATA
MATERIAL tuff
REGION all
END

:read in permeability field------------------------------------------------

DATASET permx perm_inv.dat
DATASET permy perm_inv.dat
DATASET permz perm_inv.dat
```

5 References

Balay S, Eijkhout V, Gropp WD, McInnes LC and Smith BF (1997) Modern Software Tools in Scientific Computing, Eds. Arge E, Bruaset AM and Langtangen HP (Birkhaüser Press), pp. 163–202.

6 FAQ

6.1 *iobuf load errors*

It may be the case that the 'iobuf'module is causing problems. That is a module that, if it's loaded, links with an IO buffering library. It can speed up IO considerably, but there are have been some bugs (hopefully fixed) identified with it before. It is loaded by default. You might want to try a 'module unload' of that before building PFLOTRAN, and seeing if that works. Unfortunately, it may be necessary to mess with the configuration files for the PETSc builds to make sure that you don't link with the iobuf library.