ParFlow 模型手册

ParFlow Solvers

ParFlow 使用几种不同的求解器：IMPES（默认的）和 RICHARDS。

```
pfset Solver Impes
pfset Solver Richards
```

ParFlow 还可以与 CLM 模式耦合，ParFlow 中的 CLM 代码是修改的，作为一个子程序，支持并行架构，包括 I/O 和最重要的物理过程，支持与 ParFlow 中的集成水文模块的耦合操作。

为与 ParFlow 耦合 CLM，在./configure 时增加-with-clm。然后使用：

```
pfset Solver.LSM CLM
```

定义问题

ParFlow 的主要输入文件有：一个.tcl TCL 脚本，一个.py 的 Python 脚本或一个.ipynb 的 Jupyter notebook。

输入脚本用于 PFTools 创建一个数据库，用作 ParFlow 的输入。数据库后缀名.pfidb。

ParFlow 的输入定义为 key/value 对。

理想的模拟域定义

使用 PFTools 转换工具，将几何转换为.pfsol 文件格式。

地形可使用 *xy* 方向的坡度的.pfb 文件来定义。

不管使用哪种方式，用户都要在.pfb 脚本中设置计算网格：

```
#---------------------------------------------------------------------------
# Computational Grid
#---------------------------------------------------------------------------
pfset ComputationalGrid.Lower.X -10.0
pfset ComputationalGrid.Lower.Y 10.0
pfset ComputationalGrid.Lower.Z 1.0
pfset ComputationalGrid.DX 8.89
pfset ComputationalGrid.DY 10.67
pfset ComputationalGrid.DZ 1.0
pfset ComputationalGrid.NX 18
pfset ComputationalGrid.NY 15
pfset ComputationalGrid.NZ 8
```

pfset Geom.domain.Patches "left right front back bottom top"

pfset Geom.domain.Patches {left right front back bottom top}

真实的模拟域设置

一般方法如下：

1 收集基础的输入数据，首先确定计算分辨率：

a. Elevation (DEM)

b. Soil data for the near surface layers

c. Geologic maps for the deeper subsurface

d. Land Cover

2 创建一致的网格化层，都剪裁到模拟域，有相同的网格单元数

3 将网格化文件转换为.pfb：使用转换工具 PFTCL

4 从高程数据集计算 *xy* 方向的坡度

5 为地下创建指示文件

6 为各地下单元（指示文件中定义的），确定水文特性

7 现在可以运行没有 CLM 的 ParFlow

8 转换 IGBP 的土地利用类型为 CLM 使用的分类

- 1. Evergreen Needleleaf Forest
- 2. Evergreen Broadleaf Forest
- 3. Deciduous Needleleaf Forest
- 4. Deciduous Broadleaf Forest
- 5. Mixed Forests
- 6. Closed Shrublands
- 7. Open Shrublands
- 8. Woody Savannas
- 9. Savannas
- 10. Grasslands
- 11. Permanent Wetlands
- 12. Croplands
- 13. Urban and Built-Up
- 14. Cropland/Natural Vegetation Mosaic
- 15. Snow and Ice
- 16. Barren or Sparsely Vegetated
- 17. Water
- 18. Wooded Tundra

9 创建 CLM vegm 文件，提供各单元的土地利用占比（参考 Washita 例子的

clm input 路径）

10 创建 CLM 驱动文件，设置 CLM 模型的参数

11 收集模拟域的气象驱动数据

- DSWR: Visible or short-wave radiation $[W/m^2]$.
- DLWR: Long wave radiation $[W/m^2]$
- APCP: Precipitation $[mm/s]$
- Temp: Air Temperature $[K]$
- UGRD: East-west wind speed $[m/s]$
- VGRD: South-to-North wind speed $[m/s]$
- Press: Atmospheric pressure $[pa]$
- SPFH: Specific humidity $[kg/kg]$

12 运行模型！

运行 ParFlow

使用 TCL 或 Python 运行 ParFlow。

TCL

Python

```
pip install pftools
```

脚本：

```
from parflow import Run
from parflow.tools.fs import mkdir, get_absolute_path
dsingle = Run("dsingle", __file__)
#---------------------------------------------------------------------------
dsingle.FileVersion = 4

dsingle.run()
```

```
python default_single.py
```

重启运行

可视化输出结果

SILO 格式：使用 VisIT

输出结果的转换

测试算例的路径

介绍各算例的脚本和运行…

模型控制方程

ParFlow 的文件

主要输入文件(.tcl, .py, .ipynb)

二进制文件(.pfb)

二进制格式，用于存储 ParFlow 网格数据，大端序排列。

```
<double : X> <double : Y> <double : Z>
<integer : NX> <integer : NY> <integer : NZ>
<double : DX> <double : DY> <double : DZ>
<integer : num_subgrids>
FOR subgrid = 0 TO <num_subgrids> - 1

BEGIN
    <integer : ix> <integer : iy> <integer : iz>
    <integer : nx> <integer : ny> <integer : nz>
    <integer : rx> <integer : ry> <integer : rz>
    FOR k = iz TO iz + <nz> - 1
    BEGIN
        FOR j = iy TO iy + <ny> - 1
        BEGIN
            FOR i = ix TO ix + <nx> - 1
            BEGIN
                <double : data_ijk>
            END
```

```
        END
    END
END
```

CLM 整体输出二进制文件(.c.pfb)

用于存储 CLM 输出数据，存储于一个文件，大端序排列。

Scattered 二进制文件(.pfsb)

用于存储 ParFlow 网格数据。当网格数据是“scattered”—即大多数数据是 0，使用该格式。对于此种类型数据，.pfsb 文件格式能显著降低存储空间。

Solid 文件(.pfsol)

ASCII 格式文件，用于定义 3D solids。Solid 由封闭的三角化表面表征，表面“补丁”可能与各 solid 相关联。

.pfsol 不能包含注释行。

井输出文件(.wells)

当定义井后，ParFlow 会输出井文件，包含关于内部计算中使用的井数据，以及关于井的函数的统计。

简单的 ASCII 和二进制文件(.sa 和.sb)

Pftools 使用简单的 ASCII 格式文件.sa，输出 ParFlow 网格数据。.sb 文件也是一样，是大端序的二进制格式。

输入键值

TCL 中的命令 pfset，或者 Python 中的<runname>.Key=，用于创建数据库入口。ParFlow 输入脚本就是这些键值的一系列命令。

输入文件格式编号

```
<runname>.FileVersion = 4
```

计算拓扑

分配计算进程数来设置并行计算。P, Q, R

<runname>.Process.Topology.P = 2

计算网格

定义问题。

```
#---------------------------------------------------------
# Computational Grid
#---------------------------------------------------------

<runname>.ComputationalGrid.Lower.X  = -10.0
<runname>.ComputationalGrid.Lower.Y = 10.0
<runname>.ComputationalGrid.Lower.Z  = 1.0

<runname>.ComputationalGrid.NX       = 18
<runname>.ComputationalGrid.NY       = 18
<runname>.ComputationalGrid.NZ       = 8

<runname>.ComputationalGrid.DX   = 8.0
<runname>.ComputationalGrid.DY       = 10.0
<runname>.ComputationalGrid.DZ       = 1.0
```

几何

计算域（以及域上的补丁—施加边界条件）、岩石特性或水文地层单元、断层等，都要考虑几何。

需要定义 2 项：几何输入和几何

几何输入时一类几何输入（例如一个盒子或一个输入文件）。一个几何输入可包含多个几何。

计时信息

```
## Python Example

#-----------------------------------------------------------------------------
# Setup timing info [hr]
# 8760 hours in a year. Dumping files every 24 hours. Hourly timestep
#-----------------------------------------------------------------------------
<runname>.TimingInfo.BaseUnit = 1.0
<runname>.TimingInfo.StartCount = 0
<runname>.TimingInfo.StartTime = 0.0
<runname>.TimingInfo.StopTime = 8760.0
<runname>.TimingInfo.DumpInterval = -24

## Timing constant example
<runname>.TimeStep.Type    = "Constant"
<runname>.TimeStep.Value = 1.0

## Timing growth example
<runname>.TimeStep.Type    = "Growth"
<runname>.TimeStep.InitialStep = 0.0001
<runname>.TimeStep.GrowthFactor = 1.4
<runname>.TimeStep.MaxStep        = 1.0
<runname>.TimeStep.MinStep        = 0.0001
```

时间循环

```
## Python example

#-----------------------------------------------------------
# Time Cycles
#-----------------------------------------------------------
<runname>.Cycle.Names = "constant rainrec"
<runname>.Cycle.constant.Names = "alltime"
<runname>.Cycle.constant.alltime.Length = 8760
<runname>.Cycle.constant.Repeat = -1

# Creating a rain and recession period for the rest of year
<runname>.Cycle.rainrec.Names     = "rain rec"
<runname>.Cycle.rainrec.rain.Length       = 10
<runname>.Cycle.rainrec.rec.Length = 8750
<runname>.Cycle.rainrec.Repeat = -1
```

模拟域

```
<runname>.Domain.GeomName = "domain" ## Python syntax
```

相与污染物

重力、相密度和相粘度

化学反应

渗透性

孔隙度

比储蓄量(specific storage, S_s-Eq.4.3)

dZMultipliers

```
#-----------------------------------------
# Variable dz Assignments
#-----------------------------------------
# Set VariableDz to be true
# Indicate number of layers (nzlistnumber), which is the same as nz
# (1) There is nz*dz = total depth to allocate,
# (2) Each layer's thickness is dz*dzScale, and
# (3) Assign the layer thickness from the bottom up.
# In this example nz = 5; dz = 10; total depth 40;
# Layers  Thickness [m]
# 0               15                    Bottom layer
# 1               15
# 2               5
# 3               4.5
# 4               0.5                   Top layer
<runname>.Solver.Nonlinear.VariableDz = True
<runname>.dzScale.GeomNames = "domain"
<runname>.dzScale.Type = "nzList"
<runname>.dzScale.nzListNumber = 5
<runname>.Cell._0.dzScale.Value = 1.5
<runname>.Cell._1.dzScale.Value = 1.5
<runname>.Cell._2.dzScale.Value = 0.5
<runname>.Cell._3.dzScale.Value = 0.45
<runname>.Cell._4.dzScale.Value = 0.05
```

Flow Barriers

曼宁糙率系数

<runname>.Mannings.GeomNames = "domain" ## Python syntax

<runname>.Mannings.Type = "Constant" ## Python syntax

<runname>.Mannings.Geom.domain.Value = 5.52e-6 ## Python syntax

或者

<runname>.Mannings.FileName = "roughness.pfb" ## Python syntax

```
## Python example
<runname>.Mannings.Type = "Constant"
<runname>.Mannings.GeomNames = "domain"
<runname>.Mannings.Geom.domain.Value = 5.52e-6
```

地形坡度

滞留时间

完整的多相流运动

Richards 方程的相对渗透率

相的源项

毛细孔压力

饱和度

与 Richards 方程有关。

内部边界条件

边界条件：压强

边界条件：饱和度

初始条件：相饱和度

初始条件：压强

初始条件：相浓度

已知的精确解

井

代码参数

除了与物理属性和模拟定义有关的键值，还需要定义算法和一般控制文件相关的键值。例如：

```
<runname>.Solver.Linear = "MGSemi"
<runname>.Solver.AdvectOrder = 2
<runname>.Solver.CFL = 0.7
```

SILO 选项

控制 SILO 如何输出数据的键值，SILO 允许输出到 PDB 和 HDF5 文件格式。

SILO 还允许使用数据压缩，节省存储空间。

```
<runname>.SILO.Filetype = "PDB"        ## Python syntax
<runname>.SILO.CompressionOptions = "METHOD=GZIP" ## Python syntax
```

Richards 方程求解器参数

```
<runname>.Solver.Nonlinear.ResidualTol = 1e-4
<runname>.Solver.Nonlinear.StepTol = 1e-4
<runname>.Solver.Nonlinear.MaxIter = 50
```

Spin-up 选项

仅用于初始化阶段，不用于正规运行时。

CLM 求解器参数

ParFlow NetCDF4 Parallel I/O

NetCDF4 Chunking

```
<runname>.NetCDF.Chunking = True
```

NetCDF4 Compression

```
<runname>.NetCDF.Compression = True
<runname>.NetCDF.CompressionLevel = 1
```

ROMIO Hints

ROMIO is a poratable MPI-IO implementation developed at ArgonneNational Laboratory, USA. Currently it is released as a part of MPICH. ROMIO sets hints to optimize I/O operations for MPI-IO layer through MPI_Info object. This object is passed on to NetCDF4 while creating a file.

Node Level Collective I/O

A node level collective strategy has been implemented for I/O. One process on each compute node gathers the data, indices and counts from the participating processes on same compute node. All the root processes from each compute node open a parallel NetCDF4 file and write the data. e.g. If ParFlow is running on 3 compute nodes where each node consists of 24 processors(cores); only 3 I/O streams to filesystem would be opened by each root processor each compute node. This

strategy could be particularly useful when ParFlow is running on large number of processors and every processor participating in I/O may create a bottleneck. **Node level collective I/O is currently implemented for 2-D domain decomposition and variables Pressure and Saturation only. All the other ParFlow NetCDF output Tcl flags should be set to false (default value). CLM output is independently handled and not affected by this key. Moreover, on special architectures, this may not be a portable feature. Users are advised to test this feature on their machine before putting into production.**

NetCDF4 Initial Conditions: Pressure

NetCDF4 Slopes

NetCDF4 Transient EvapTrans Forcing

NetCDF4 CLM Output

NetCDF4 CLM Input/Forcing

NetCDF Testing Little Washita Test Case

基本的 NetCDF 的输出（压力和饱和度）以及初始条件（压力）的功能，可使用下面的 tcl 脚本来测试。CLM 输入和输出功能也可以用以下脚本测试。

parflow/test/washita/tcl_scripts/LW_NetCDF_Test.tcl

测试算例将使用下面的初始条件文件、坡度和气象驱动文件完成初始化：

parflow/test/washita/parflow_input/press.init.nc
parflow/test/washita/parflow_input/slopes.nc
parflow/test/washita/clm_input/metForcing.nc

操作数据：PFTools

介绍 PFTCL 命令

首先，加载 ParFlow 软件包到 TCL 脚本，如下：

```
#
# To Import the ParFlow TCL package
#
lappend auto_path $env(PARFLOW_DIR)/bin
package require parflow
namespace import Parflow::*
```

PFTCL Commands 提供一些使用工具做前后处理的例子。

pfhelp 查询命令。

PFTCL 命令

Table 7.1: List of PFTools commands by function

使用 PFTCL 的例子

Python

PFTools

https://pypi.org/project/pftools/

安装

pip install pftools[all]

执行

python3 /path/to/script/run_script.py [args]

Usage:

```
run_script.py [-h] [--parflow-directory PARFLOW_DIRECTORY] [--parflow-version PARFLOW_
↪VERSION]
[--working-directory WORKING_DIRECTORY] [--skip-validation] [--dry-run] [--show-line-
↪error]
[--exit-on-error] [--write-yaml] [--validation-verbose] [-p P] [-q Q] [-r R]
```

运行脚本

语法

parflow/test/python/

```
from parflow import Run
test_run = Run("test_run", __file__)
```

设置键值

```
from parflow import Run
test_run = Run("test_run", __file__)

# Now that the Run object is initialized, you can set keys:
test_run.Process.Topology.P = 1
test_run.Process.Topology.Q = 1
test_run.Process.Topology.R = 1
```

有效的键值名

使用 pfset()设置键值

使用 pfset()设置库中没有的键值

键的验证

方法

Other methods that can be called on a Run object are shown below:

```python
from parflow import Run

# Instantiate a Run object
test_run = Run("test_run", __file__)

# Distribute a ParFlow binary file associated with a run
# P, Q, and R optional arguments override Process.Topology values
test_run.dist('test_slopes.pfb')

# Validate the values set to the keys of the Run object
test_run.validate()

# Write out key/value pairs to a file
test_run.write(file_format='pfidb')
test_run.write(file_format='yaml')
test_run.write(file_format='json')

# Write pfidb file and run ParFlow in the same directory as the script, skipping␣
↪validation
test_run.run(skip_validation=True)

# Clone the run into a new Run object
cloned_run = test_run.clone('cloned_run')
```

Full API

例子

Tutorials

把 TCL 脚本转换为 Python 脚本

$PARFLOW_DIR

测试：

python3 $PARFLOW_SOURCE/test/python/base_3d/default_richards/default_richards.py

转换：

mkdir -p pftools_tutorial/tcl_to_py
cd pftools_tutorial/tcl_to_py
cp $PARFLOW_SOURCE/test/default_richards.tcl .

python3 -m parflow.cli.tcl2py -i default_richards.tcl

还需要一些手动修改。

然后：python3 default_richards.py

Troubleshooting when converting TCL script to Python

文件系统

```
from parflow.tools.fs import mkdir, cp

mkdir('input-data')
cp('$PF_SRC/test/input/*.pfb', './input-data/')
```

PFB

Solid Files

Tables for Subsurface Parameters

Domain definition helpers

Data Accessor

Hydrology Module

Contributing keys

YAML definitions

Steps to add a new key