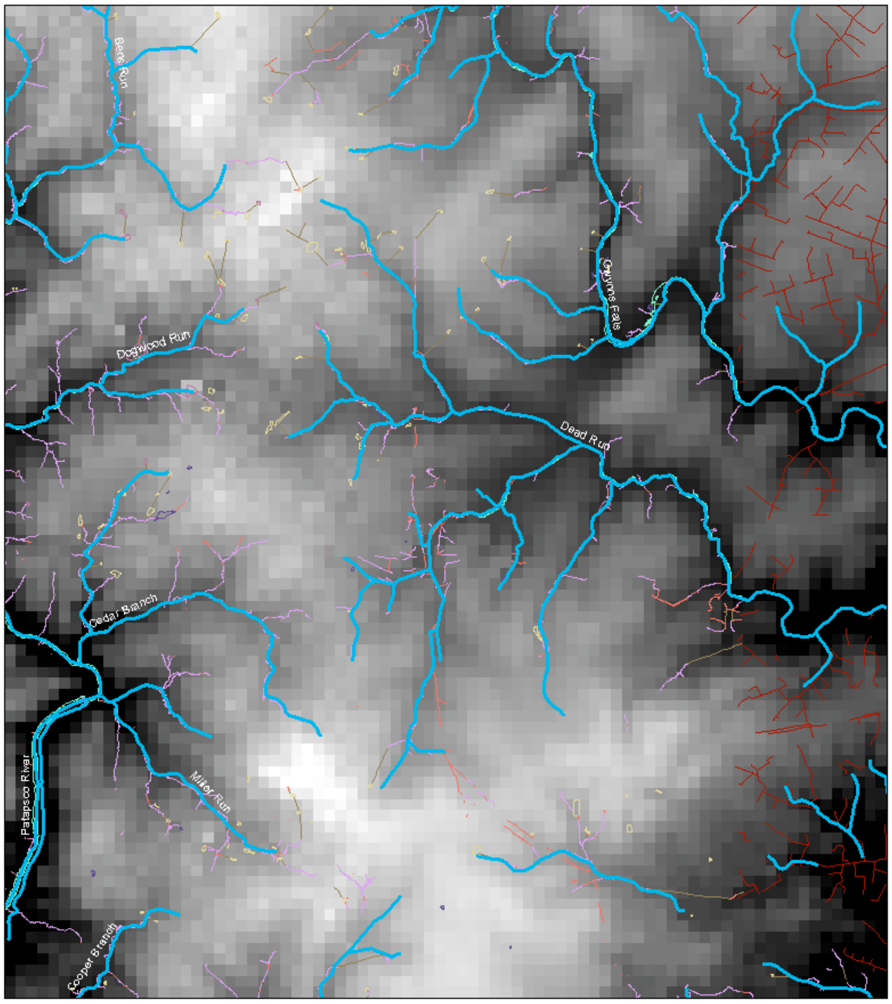
Center for Urban Environmental Research and Education
University of Maryland, Baltimore County

Getting Started with ParFlow: Dead Run, Baltimore, Maryland Example

CUERE Technical Report 2010/002
October 2010

Aditi Bhaskar



Getting Started with ParFlow: Dead Run, Baltimore, Maryland Example

CUERE Technical Report 2010/002
October 2010

Aditi Bhaskar

University of Maryland, Baltimore County
Center for Urban Environmental Research and Education
1000 Hilltop Circle, Technology Research Center 102
Baltimore, Maryland 21250

This work was carried out under support of NSF Grants # 0549469 and 0709659, C. Welty, PI.
This report is available as a downloadable pdf file from the internet at http://www.umbc.edu/cuere/BaltimoreWTB.

Please cite this publication as:
Bhaskar, Aditi. 2010. Getting Started with ParFlow: Dead Run, Baltimore, Maryland Example. UMBC/CUERE Technical Report 2010/002. University of Maryland Baltimore County, Center for Urban Environmental Research and Education, Baltimore, MD.

ON THE COVER
Streams (blue), drainage connectors (brown) and storm sewers (red) overlain on digital elevation model of Dead Run watershed.

Table of Contents

List of Figures

Abstract

This document shows how to get started with ParFlow by using the example of Dead Run, an urbanized watershed in Baltimore, Maryland, and especially focuses on pre- and post-processing steps. This document takes the reader through the background reading on ParFlow, necessary software downloads and installations, pre-processing steps for the DEM, generation of the necessary input files, setting up the problem input script and visualizing the resulting output using VisIt or MATLAB. A water balance check is also described. Three examples are included: a saturated, steady-state model run, an overland flow test or parking lot test, which is a fast way to tell whether the topographic slopes are well connected, and a coupled overland flow – variably saturated model run. Because the overland flow component of the model is used, a number of pre-processing steps are necessary to modify the DEM and topographic slopes in order to get the streams connected and flowing at the 100 m resolution used. This document includes the description and instructions for a code that is used to enforce slopes along multiple branching stream channels in the domain.

1. Introduction

ParFlow is an integrated, parallel watershed model that makes use of high-performance computing to simulate surface and subsurface fluid flow. The goal of the ParFlow project is to enable detailed simulations for use in the assessment and management of groundwater and surface water, to investigate system physics and feedbacks and to understand interactions at a range of scales.

This document details how to get started with ParFlow using the example of a small urban watershed in Baltimore, Maryland: Dead Run. The impetus for writing this user guide was to provide detailed instructions for using ParFlow in the overland flow mode, with steps provided for a specific example. The instructions include critical information on how to import, process, and resample a DEM. If DEM processing is not done correctly, running the model with the overland flow option will be very difficult. The focus of this report is on file pre- and post-processing, and will take the reader from the beginning of reading about and downloading ParFlow through getting the Dead Run example model running.

2. Journal articles for background reading

The ParFlow project was started at Lawrence Livermore National Laboratory (LLNL, https://computation.llnl.gov/casc/parflow/parflow_home.html), and continues to be developed by LLNL and Reed Maxwell of Colorado School of Mines (http://inside.mines.edu/~rmaxwell/maxwell_software.shtml). The following reading list provides the theoretical background and details of code implementation. Reed Maxwell's website provides links to the key ParFlow papers, as well as information on how to cite ParFlow development in publications.

Ashby, S., C. Baldwin, B. Bosl, R. Falgout, R. Hornung, S. Smith and C. Woodward. 2001. ParFlow User's Manual, LLNL Report, UCRL-MA-123204, 87p.

Chow F.T., S. J. Kollet, R. M. Maxwell, Q. and Duan. 2006. Effects of soil moisture heterogeneity on boundary layer flow with coupled groundwater, land-surface, and mesoscale atmospheric modeling, *AMS 17th Symposium on Boundary Layers and Turbulence*, San Diego.

Jones J.E. and C.S. Woodward. 2001. Newton-Krylov-multigrid solvers for large-scale, highly heterogeneous, variably saturated flow problems, *Advances in Water Resources*; 24: 763-774.

Kollet, S.J. and R.M. Maxwell. 2006. Integrated surface-groundwater flow modeling: A free-surface overland flow boundary condition in a parallel groundwater flow model, *Advances in Water Resources*, 29(7), 945-958.

Kollet, S.J. and R.M. Maxwell. 2008a. Capturing the influence of groundwater dynamics on land surface processes using an integrated, distributed watershed model, *Water Resources Research,* 44:W02402, doi:10.1029/2007WR006004.

Kollet, S.J. and R.M. Maxwell. 2008b. Demonstrating fractal scaling of baseflow residence time distributions using a fully-coupled groundwater and land surface model. *Geophysical Research Letters,* 35, L07402, doi:10.1029/2008GL033215.

Maxwell, R.M., F. K. Chow, and S. J. Kollet. 2007. The groundwater-land-surface-atmosphere connection: soil moisture effects on the atmospheric boundary layer in fully-coupled simulations, *Advances in Water Resources* 30(12), 2447-2466.

Maxwell, R.M. and S.J. Kollet. 2008. Quantifying the effects of three-dimensional subsurface heterogeneity on Hortonian runoff processes using a coupled numerical, stochastic approach, *Advances in Water Resources,* doi:10.1016/j.advwatres.2008.01.020.

Maxwell, R.M. and N.L. Miller. 2005. Development of a coupled land surface and groundwater model, *Journal of Hydrometeorology,* 6(3), 233-247.

3. Software downloads and installations

3.1 ParFlow

The latest release of ParFlow can be downloaded from http://inside.mines.edu/~rmaxwell/maxwell_software.shtml. The user manual is also available there, as well as other good sources of information such as how to join the ParFlow users' list and the ParFlow blog. The archive of messages to the ParFlow users' list can be found at https://mailman.mines.edu/pipermail/parflow-users, where answers to a number of commonly asked questions can be found.

ParFlow can be installed on a Linux/Unix system or OSX; it does not run in Windows. The example used in this report was run on Ubuntu 8.04 (http://releases.ubuntu.com/8.04). The ParFlow manual describes how to do a standard installation. There are some peculiarities of installing the current ParFlow version on Ubuntu or other Debian-based Linux systems. John Williams has written a detailed post on the ParFlow blog about installing ParFlow on Ubuntu 9.04 at http://parflow.blogspot.com/2009/06/installing-parflow-on-ubuntu-904-jaunty.html.

Included in John Williams' installation instructions are other packages needed for running ParFlow: Fortran and C compilers, Tcl/Tk, Silo and Hypre. For the example described here, almost all of the detailed instructions in this blog post were followed. One difference is that the Dead Run example is a relatively small problem, and so was run on a single processor on a local computer. Therefore, installing OpenMPI was not required, and the configure line (used to configure the installation to a specific computer and include the packages desired) was changed from "--with-amps=mpi1" to "--with-amps=seq", as follows:

```
./configure --prefix=$PARFLOW_DIR --with-amps=seq --with-silo=$SILO_DIR --
with-hypre=$HYPRE_DIR --enable-timing --with-tcl=/usr/local
```

The version of ParFlow that was used for the example was revision 488 of ParFlow version 3. The most recent ParFlow updates can be downloaded using subversion version control (http://subversion.apache.org/) by individuals who have obtained permission from Reed Maxwell to have the user name and password. Once subversion is installed, the most recent version of ParFlow can be installed (or 'checked out') by entering the following line in the terminal within the directory where the download is desired:

```
svn co https://parflow.svn.cvsdude.com/parflow/parflow/trunk .
```

ParFlow users recommend installing each new version of ParFlow in a new directory to preserve old versions. Note that the user's .bash_profile (as described in John Williams' blog post) must be updated whenever this is done and the path to the desired ParFlow version changes.

3.2 MATLAB and ArcGIS

The discussion in this report assumes that MATLAB (http://www.mathworks.com/products/matlab) and ArcGIS (http://www.esri.com/software/arcgis/index.html) software packages are available to the user for data processing and visualization. These are proprietary products that must be purchased and installed. Many universities provide site licenses for faculty and students. Since ArcGIS only runs in Windows, a dual boot system or VirtualBox (http://virtualbox.org) can be used in order to switch between Windows for ArcGIS and Linux for ParFlow.

3.3 VisIt

The newest releases of ParFlow generate output in silo format, which is easily viewed in VisIt, a powerful visualization software. The ParFlow blog includes an entry by John Williams on how to install VisIt on Ubuntu (http://parflow.blogspot.com/2009/06/installing-visit-on-ubuntu-904-jaunty.html).

4. Dead Run example

The broad steps for running this example are (1) processing the DEM; (2) generating a solid file; (3) generating slopes files; and (4) visualizing the output. Additional essential considerations for running the model are discussed. All files and codes created or modified for this example are listed in Appendix A.

4.1 Processing the DEM

For this example, the desired Δx-Δy gridding is 100 m x 100 m. The choice of gridding is dependent upon project objectives.

USGS DEM is available as approximately 30 m x 30 m pixels (the exact size varies with latitude and longitude). It will always need to be resampled onto an exact Δx-Δy grid for purposes of constructing model input.

The steps for processing the DEM using an Arc GIS tool are summarized in Table 1.

Table 1. Steps for processing the DEM (courtesy of Michael P. McGuire).

---

1. Download DEM file from USGS seamless server
   - a. Navigate web browser to http://seamless.usgs.gov.
   - b. Click on "View & Download United States Data."
   - c. Use "zoom in" tool in left panel to zoom to area of interest.
   - d. Click on download tab in right panel.
   - e. Expand the elevation layer set and select 1" NED (approximately equal to 30 m).
   - f. Select the "Define rectangular download area" tool in the left panel and draw a rectangle around the area of interest. This rectangle should be slightly larger than the exact model domain because later the DEM will be re-projected and clipped.
   - g. A popup window should appear. In this window click the download button. It may take a few minutes for the layer to be clipped and available for download. In the case of very large domains, it may be necessary to download the DEM in multiple files.
   - h. Save the zip file to the folder where data are to be stored, and extract.

2. Re-project DEM to a projected coordinate system (in this example, State Plane NAD83 Meters is used in the Dead Run example)
   - a. Open a blank map document in ArcMap.
   - b. Click on "add data" button ✚ and add the DEM grid that was downloaded from the seamless server.
   - c. Click on the ArcToolbox icon in ArcMap.
   - d. In the ArcToolbox window, navigate to the project raster tool under Data Management Tools > Projections and Transformations > Raster > Project Raster.
   - d. In the input raster field select the DEM file.
   - e. Supply a name for output projected DEM file in the Output raster text input box.
   - f. Click on the select output coordinate system button
   - g. Click select and navigate to Projected Coordinate Systems > State Plane > NAD 1983 > NAD 1983 StatePlane Maryland FIPS 1900.prj and click OK. The DEM should now be available as a raster in ArcMap.

3. Resample the DEM

a. Add the spatial analyst toolbar by going to the Tools menu and selecting customize. Click on the Spatial Analyst check box.

b. In the Spatial Analyst toolbar make sure that the projected DEM file is selected in the dropdown box.

c. Click on Spatial Analyst > Options

d. Under the General Tab set the working directory to the folder where to the output files are to be saved.

e. Click on the extent tab and manually enter extent parameters. In the case for the subject Dead Run example, the extent is (in meters):

Top: 187030
Bottom: 178030
Left: 418260
Right: 426260

f. Click on the cell size tab and enter the desired cell size (choose as specified below) for the resampled grid (in the case of Dead Run the cell size is 100 m).

i. In the Spatial Analyst toolbar, select the raster calculator tool.

The syntax for resampling is:

```
<output grid> =  resample([<input grid>], <cell size>, {NEAREST | BILINEAR |
CUBIC | SEARCH})
```

Other resampling algorithms can be used, but in the case of Dead Run example syntax we used:

```
ned_deadrun = resample([proj_ned], 100, BILINEAR)
```

j. Click “Evaluate”.

4. Convert DEM to ASCII

a. In the ArcToolBox window navigate to Conversion Tools > From Raster > Raster to ASCII.

b. In the input raster select the resampled raster.

c. Supply an output name for the ASCII file.

d. Click OK.

4.2 General notes on axes naming and executing Fortran codes

Different conventions can be used for naming the axes for ParFlow input and output files. In the case of this example, the x axis was chosen as the N-S direction (increasing from north to south) or vertical in map view, and y was chosen as the E-W direction (increasing from E to W) or horizontal in map view. This choice was made because "i" is used to loop over the x direction in the solid file and slopes codes and is also used to represent rows in the DEM. Note that this is likely unintuitive to most users because these labels are switched from the more common usage. Either choice is fine as long as consistency is retained when plotting, etc.

Both the codes to create the solid file and the slopes files (discussed in sections 4.3 and 4.4) are written in Fortran, so the user needs to know how to execute such codes. To run a Fortran code, the user should navigate to the location where the code is stored (in the terminal, the user should type "cd" and the path for the directory). To use gfortran, the user should enter 'gfortran nameofcode.f90' in the command line interface terminal. To use Intel Fortran (if it was installed during the ParFlow installation as suggested in the ParFlow blog post in the installation section), the user should type 'ifort nameofcode.f90'. After either of these, if the code is compiled correctly, an executable will be created in the directory labeled a.out. To execute this, the user must type ./a.out in the terminal. The f90 codes included with ParFlow may give compiler errors when compiled with gfortran as they appear to be made for compiling with ifort (Intel Fortran). We found additional modifications were necessary for use with gfortran. For example, for gfortran the open statement with recordtype = 'stream' was changed to access = 'stream'; however, for other compilers other syntax may be needed. The other lines that may cause problems are the write statements that specify the format, for example, as ('100e'). This can be changed to any other desired format or to free format by changing the ('100e') to *.

4.3 Generating a solid file

A solid file is a necessary input to ParFlow if any variation from simple box geometry domain is to be used. The computational grid, specified in the Tcl script, defines the maximum extent of the solid file and gridding for the problem, but the solid file allows the user to make some of the cells in the depth or Z dimension inactive, so that the land surface has varied topography. The solid file code creates a "solid file" of the model domain, a type of TIN (triangulated information network) file, using the DEM information acquired for the problem as described above. Executing the solid file Fortran 90 code produces output having the extension .pfsol in addition to two text files. One straightforward way to create a pfsol file is to modify the code provided by Reed Maxwell and modified by Aditi Bhaskar for the Dead Run case, "pf_solid_file_create.f90". In the version of ParFlow used at the time of this writing (parflow.r488), the pfsol code can be found in the $PARFLOW_DIR/pftools/prepostproc directory after ParFlow is installed. The code uses as input DEM in the format of a text file with the rows representing the x dimension in the domain and the columns representing the y dimension.

In the Dead Run example, elevation is measured from the bottom of the domain, which in this case is the bottom of the aquifer. Therefore, in the solid file code, a thickness of the domain of 200 m below the lowest surface elevation is added to the DEM values to account for the aquifer thickness.

The patch order is specified in the solid file code. Patches are boundaries of the 3D domain over which boundary conditions are specified. The solid file code assumes that there are 6 patches, one on each side of the ‘box’, with the top of the box being an irregular surface defined by elevations. The order of the patches specified in the Tcl file (e.g. pfset Geom.domain.Patches "z-upper z-lower x-lower x-upper y-lower y-upper") needs to be consistent with that in the solid file. If the patch order differs between the Tcl file and the solid file, ParFlow will not run properly.

The solid file can be checked using the mask file that is produced with a ParFlow run. The silo mask file (NameOfRun.out.mask.silo) can be read into VisIt to verify the 3D domain is set up as intended; details are provided in section 4.6.1.1.

For other applications, the solid file “pf_solid_file_create_inactivity.f90” is included, which only creates 3 patches. This code was given to the author by R. M. Maxwell. This code can be used to exclude part of the X-Y domain, in addition to part of the Z domain as done in the code pf_solid_file_create.f90. It may be desirable to also exclude features such as lakes or bays from the model domain, and this can be done in using this code. This is not necessary for the Dead Run case, but this code is included for situations where it may be needed.

4.4 Generating slopes files

The x and y slope in each cell is needed for the overland flow component of the model. As with other parameters, constant values for the slopes can be set directly in the Tcl file, but if spatially variable slopes are desired, then files specifying the values for the slopes need to be created and input in ParFlow binary format (extension .pfb). The slopes code creates two .pfb files that specify the topographic slopes to be used as input for ParFlow. Although this code also uses DEM file as input, generation of the slopes file is an independent step from generating the solid file.

4.4.1 Processing raw DEM data

There are errors in DEMs, especially as represented at a coarse resampled resolution, resulting in cells that are at a lower elevation than all of their neighbors. These are referred to as “sinks”, and before calculating slopes, these sinks must be filled. In the case of the ParFlow overland flow component, flow is not computed diagonally across cells, so each cell has four neighbors (that is not at the edge of a domain) to/from which flow can occur. In the following example, where numbers indicate elevations,

```
9 9 9
9 8 9
9 9 7
```

the four cells directly adjacent to the middle “8” cell point inward in terms of slope, and therefore water would accumulate there. These cells are called sinks because water can flow into but not out of that cell based on the elevation. However, problems may be even more complex because there may be multi-cell sinks in the domain.

Because of the way ParFlow is set up to use 4 surrounding cells only in the flow computations, commonly used D8 sink-filling routines (e.g. tools in ArcGIS) cannot be used. D8 refers to 8-directional, meaning that calculations use 4 diagonal neighbors in additional to 4 directly adjacent neighbors to each cell. These D8 sink filling routines would not consider a cell with a neighboring diagonal lower elevation cell (7 in the above example) as a sink, and therefore cannot be used for DEM processing for ParFlow.

To address this problem, there are two D4 sink filling codes available that use different algorithms: pit-filling and moving average. These codes were provided by R. M. Maxwell. The pit-filling algorithm (slopes/fillsinks1_pitfilling.f90) raises the elevation of each sink cell in the domain by a small amount (e.g. 1.5 cm in the example) until either there are no more sinks or the code reaches the maximum number of iterations through the domain specified. The moving average algorithm (slopes/fillsinks1_mavg.f90) uses a different approach. If a sink cell is found, then the elevation at that cell is replaced with the elevation of a small aerial average of the 9 cells (3 x 3 square) surrounding and including the sink cell. As can be seen from the code, edge cells are dealt with differently.

The choice of algorithm depends on terrain and personal judgement. For example, R. M. Maxwell (personal communication, July 2009) suggested that the moving average algorithm is better for flat or rolling terrain. For the Dead Run example, we used the pit-filling algorithm because more detail was retained in the result, whereas the moving average algorithm resulted in much more smoothing of the elevations.

The Fortran code is labeled make_slopes_pitfill.f90. This differs from what we originally obtained from R. M. Maxwell in that this code includes the pit-filling algorithm from the fillsinks1_pitfilling.f90 code, and then calculates slopes and writes the pfb files. It should be noted that we take nz*dz = total z distance, including the aquifer thickness.

To check that the slopes code is working, the output text files can be opened and visualized and the pfb files can be read using pfb_read.m, which is included in the pftools directory.

Slopes generated using the slopes code may cause computational problems for some landscapes and for some resolutions, and stream channels may need to be enforced, as discussed in section 4.7.2. This can be checked with an overland flow test (also called a "parking lot test"), which is a quick test run of ParFlow. In order to run ParFlow, the ParFlow Tcl file first needs to be set up.

4.5 Generating the ParFlow Tcl file

In making a Tcl file for the desired application, it is recommended to start from a Tcl example file and make one change at a time, keeping a copy of the latest Tcl file that works. There are a number of example Tcl files for different conditions in the $PARFLOW_DIR/test/ directory included with the ParFlow installation. For the Dead Run application, we started with the

Harvey Flow Tcl file ($PARFLOW_DIR/test/harvey.flow.tcl) as a template and added features that were necessary for our case. The Dead Run example Tcl files, deadrun_sat.tcl, deadrun_oftest.tcl and deadrun_unsat.tcl for saturated, overland flow test, and variably saturated flow can be downloaded from http://www.umbc.edu/cuere/BaltimoreWTB/modeling.html, within the zipped file and their respective directories. To check that the code has run satisfactorily, both the out.log, out.txt and pfb files should be checked. Instructions on how to view binary files are provided in section 4.6.2.

4.5.1 Computational grid

In the section of the Tcl file where computational grid is specified, the lower z should be 0.0 (in general, the same that is specified for bottom elevation in the solid file generation). All other items in this section should agree with entries in the slopes and solid file codes, e.g., dx*nx=total x distance. The computational grid used in deadrun_unsat.tcl is:

```
pfset ComputationalGrid.Lower.X                0.0
pfset ComputationalGrid.Lower.Y                0.0
pfset ComputationalGrid.Lower.Z                0.0
pfset ComputationalGrid.DX                     100.
pfset ComputationalGrid.DY                     100.
pfset ComputationalGrid.DZ                      1.
pfset ComputationalGrid.NX                      90
pfset ComputationalGrid.NY                      80
pfset ComputationalGrid.NZ                      339
```

4.5.2 Choosing a time step

In general, it is desirable to optimize the choice of timestep to be as large as possible to minimize computation time without occurrence of backtracks and convergence failures. Backtracking occurs when the solver cannot meet the tolerance specified for convergence and then divides trials to take a half a time step. The number of backtracks and linear and nonlinear convergence failures can be seen in the <runname>.out.kinsol.log file output with any variably saturated ParFlow runs. As a guideline from R. M. Maxwell, if 50 nonlinear iterations are approached or backtracks and convergence failures occur, then the timestep should be reduced. For the Dead Run example, time steps between 0.1 hours and 1 hour were used depending on the other conditions. For overland flow tests, short time steps (0.1 hour) were chosen because those runs are very fast, whereas for long coupled surface-subsurface runs longer time steps were necessary. The timing info used in deadrun_unsat.tcl is given as:

```
pfset TimingInfo.BaseUnit             0.01
pfset TimingInfo.StartCount           0
pfset TimingInfo.StartTime            0.0
pfset TimingInfo.StopTime             100.0
pfset TimingInfo.DumpInterval         -1
pfset TimeStep.Type                   Constant
pfset TimeStep.Value                  1.0
```

4.5.3 Time cycles

Time cycles can be used to change boundary conditions over time. This capability is used in the overland flow test to produce heavy rainfall in the first hour of the model run and no rainfall in the second hour. The lengths of the time cycles are set as integer multiples of the BaseUnit, which is set in the TimingInfo section. The BaseUnit may be set as a value smaller than the timestep value to avoid roundoff errors. In the case of deadrun_oftest.tcl, the BaseUnit is set as 0.001 (hours), and the time cycles are set as 1 hour of rain and 1 hour of recession:

```
pfset Cycle.Names                        "constant rainrec"
pfset Cycle.constant.Names               "alltime"
pfset Cycle.constant.alltime.Length      1
pfset Cycle.constant.Repeat              -1
pfset Cycle.rainrec.Names                "rain rec"
pfset Cycle.rainrec.rain.Length          1000
pfset Cycle.rainrec.rec.Length           1000
pfset Cycle.rainrec.Repeat               -1
```

And the z-upper boundary conditions are set using the time cycles:

```
pfset Patch.z-upper.BCPressure.Type       OverlandFlow
pfset Patch.z-upper.BCPressure.Cycle      "rainrec"
pfset Patch.z-upper.BCPressure.rain.Value -0.005
pfset Patch.z-upper.BCPressure.rec.Value   0.00
```

4.5.4 Permeability value

Intrinsic permeability is equal to hydraulic conductivity (K), if it is normalized by gravity, density and viscosity with values set equal to 1 as described in the ParFlow manual. This was also done in the Dead Run Tcl example scripts provided. The units that are entered for K determine the units of the all other values in the Tcl file. For example, K = 4 m/day implies that all other units specified in the Tcl script, as well as the outputs given, are given in units of length as meters and time as days. This also means that the pressure head and pressure are equal, and therefore, the descriptions are used interchangeably. The permeability value used in deadrun_unsat is:

```
pfset Geom.Perm.Names "domain"
pfset Geom.domain.Perm.Type Constant
pfset Geom.domain.Perm.Value 0.166667
```

4.5.5 Boundary conditions

The boundary conditions used in the Dead Run case are flux = 0 at all boundaries except for the top (upper) boundary, which is of the type “overland flow”. With the overland flow boundary condition, groundwater cannot flow in/out of the domain, but water can exit as the domain as overland flow or streams. The value for pfset Patch.z-upper.BCPressure.alltime.Value can be set to the net recharge value. The net recharge value should be a negative value if water is entering the domain (z increases with increase in elevation from the bottom of the aquifer, whereas net recharge is in the opposite direction). Units for recharge must match the units for K. For the example shown in deadrun_unsat.tcl, the net recharge value is set to 0 so that the example can run faster and use a larger time step. The boundary conditions used in deadrun_unsat.tcl are specified as:

```
pfset BCPressure.PatchNames           "z-upper z-lower x-lower x-upper \
                                       y-lower y-upper"
pfset Patch.y-lower.BCPressure.Type    FluxConst
pfset Patch.y-lower.BCPressure.Cycle   "constant"
pfset Patch.y-lower.BCPressure.alltime.Value    0.0
pfset Patch.y-upper.BCPressure.Type    FluxConst
pfset Patch.y-upper.BCPressure.Cycle   "constant"
pfset Patch.y-upper.BCPressure.alltime.Value    0.0
pfset Patch.x-lower.BCPressure.Type    FluxConst
pfset Patch.x-lower.BCPressure.Cycle   "constant"
pfset Patch.x-lower.BCPressure.alltime.Value    0.0
pfset Patch.x-upper.BCPressure.Type    FluxConst
pfset Patch.x-upper.BCPressure.Cycle   "constant"
pfset Patch.x-upper.BCPressure.alltime.Value    0.0
pfset Patch.z-lower.BCPressure.Type    FluxConst
pfset Patch.z-lower.BCPressure.Cycle   "constant"
pfset Patch.z-lower.BCPressure.alltime.Value    0.0
pfset Patch.z-upper.BCPressure.Type    OverlandFlow
pfset Patch.z-upper.BCPressure.Cycle   "constant"
pfset Patch.z-upper.BCPressure.alltime.Value    0.0
```

4.5.6 Initial condition

The initial condition used in the Dead Run example is that the water table is set to 5 m below the land surface. The initial condition used in deadrun_unsat.tcl is specified as:

```
pfset ICPressure.Type                      HydroStaticPatch
pfset ICPressure.GeomNames                 domain
pfset Geom.domain.ICPressure.Value          -5.0
pfset Geom.domain.ICPressure.RefGeom        domain
pfset Geom.domain.ICPressure.RefPatch       z-upper
```

4.5.7 Water balance check

Water balance computations can be done within the Tcl script using pftool commands. These water balance commands are shown in the deadrun_oftest.tcl and deadrun_unsat.tcl Dead Run example Tcl scripts following the pfundist commands, and were written by R.M. Maxwell. This part of the script calculates the change in storage, flux of water into the domain from the boundary conditions, and flux out from surface runoff at each timestep. The balance is checked at each timestep, and an error is given if the percent difference between the expected and calculated total water sum is greater than a threshold. Errors can indicate, among other things, errors with the solid file such that it is outside computational domain. The Tcl script can be modified to write out calculations in text files and plotted in MATLAB. Alternately, a MATLAB code “massbalance.m” written by the author and that can be found in the overland_flow_test directory, reads the pfb pressure files, calculates the water balance at each timestep and plots the result.

4.6 Visualizing output

4.6.1 VisIt

VisIt (https://wci.llnl.gov/codes/visit/) is free visualization software developed at LLNL. It is particularly good at handling 3D visualization and time series visualization. Time series animation movies can be made in VisIt by adding a set of pressure or saturation ParFlow outputs at each timestep. VisIt reads silo files as input, and ParFlow can write out parameters and outputs in silo format using keys in the Tcl script. These keys can be included in the Solver Settings section of the Tcl and are:

```
pfset Solver.WriteSiloSubsurfData                      True
pfset Solver.WriteSiloSlopes                           True
pfset Solver.WriteSiloMask                             True
pfset Solver.WriteSiloMannings                         True
pfset Solver.WriteSiloSpecificStorage                  True
pfset Solver.WriteSiloPressure                         True
pfset Solver.WriteSiloSaturation                       True
```

4.6.1.1 Solid file visualization

The solid file can be checked using the mask silo file that is output from ParFlow by constructing a simple and short run Parflow run (e.g. constant slope files, 1 timestep); ParFlow creates the mask silo file as one of the first outputs of the run. The silo mask in VisIt can be visualized by first launching VisIt by entering 'visit' in the command line terminal. Then by clicking File > Open File, entering the path of the directory in which the mask file is located, selecting the mask file and clicking Ok, the mask file can be opened.

The type of plot can be selected from the main VisIt menu on the left side of the screen, under the 'Plots' menu. The pseudocolor plot will show the 3D domain with colors corresponding to the cell values, and will work for visualizing the mask as well. Choosing Plots > Pseudocolor > Pressure and then selecting Draw will display the pseudocolor plot of the file that is chosen from the Selected files section.

The XY plot can be rotated to show the domain from any direction by simply clicking and dragging the plot, or can be reset with the button that looks like a camera with a green X on top of it.

The threshold operator (Operators > Threshold) can be used to mask inactive cells in the domain. Clicking the right arrow to the left of the plot name under 'Active plots' will show any operators used, such as the threshold operator. Clicking on threshold or pseudocolor will allow the user to modify the limits of the threshold or color bar.

The vertical exaggeration of the plot can be changed by clicking Operators > Transform. Then clicking on the black arrow to the left of the name of the plot, and then on the transform line, will allow the user to modify Transform Operator Attributes. Increasing the value for Z under Scale will increase the vertical exaggeration. An image can be saved of the current drawing by selecting File > Export Database, and changing the 'Export to' option to image.

4.6.1.2 Visualization of other output files

The silo files output at every time-step (pressure and saturation) are automatically recognized by VisIt as a set, or a silo database. These are shown grouped under Files. The complete set of timestep outputs can be opened by clicking on the databse in the open file menu, for example, deadrun.out.press.*.silo database. ParFlow binary (pfb) files cannot be opened using VisIt.

Timesteps can be stepped through in an animation by pressing the play button below the selected files. This can be saved as a movie under File > Save Movie. Similar to visualizing the solid file using the silo mask, the threshold operator (Operators > Threshold) can be used to mask inactive cells in the domain. The range of values shown in the color bar can be adjusted by clicking PlotAtts > Pseudocolor, and then modifying the max and min under limits. This can also be done by clicking the right arrow to the left of the plot name, where any operators used can also be modified.

Similar to the lines that are used at the end of the example Tcl files discussed in the MATLAB section below, pfcomputetop (part of pftools) can be used to extract the land surface of the pressure head and saturation output from ParFlow. This can be written as a 2D silo file and visualized in VisIt.

Because of the axis labeling that is used in these Dead Run examples, the X and Y axes are by switched in VisIt, so that North does not properly correspond with the top of the plot.

The author finds that MATLAB is easier to use than VisIt, and therefore the use of MATLAB to visualize ParFlow output is described in the next section.

4.6.2 MATLAB

Often it is desirable to visualize the pressure and saturation at the land surface, which is not at a constant elevation. Pftools commands can be used to extract values at the land surface from a 3D variable. Extraction of the land surface pressure and saturation is done for the Dead Run case by inserting the following lines at the bottom of the Tcl script, or in a post-run Tcl script (the latter case with the pfdist, pfrun and pfundist lines commented out):

```
set mask [pfload deadrun.out.mask.pfb]
set top [Parflow::pfcomputetop $mask]

set data [pfload deadrun.out.press.00005.pfb]
set top_data [Parflow::pfextracttop $top $data]

pfsave $top_data -sa "top.pressure.txt"

set data [pfload deadrun.out.satur.00005.pfb]
set top_data [Parflow::pfextracttop $top $data]

pfsave $top_data -sa "top.satur.txt"
```

The 3rd and 6th lines of the above script should be modified for the timestep desired to be read. The output text files can be read into MATLAB. A short MATLAB code is included for this purpose, named makeplots.m, which also does some of the formatting for the Dead Run example.

Another way to visualize any pfb file is by using the included MATLAB code “landsurface.m” modified by the author from pfb_read.m, written by Jehan Rihani. When visualizing pfb files (pressure, saturation, permeability, porosity, etc.), it is important to remember that cells above the land surface are masked in ParFlow and so the cell values do not have physical meaning. The file that is used to mask the cells above the land surface is output from any variably saturated run (based on the solid file) is <runname>.out.mask.pfb

4.7 Overland flow (parking lot) test

4.7.1 Setting up the overland flow test

The slopes are used to route water in overland flow. Especially in flat or urban domains with large cell sizes, the slopes as calculated from the processed DEM may not be adequate to define the streams. A sign that this is a problem is that the streams appear to be disconnected. At the cell upstream of the disconnection, large pressures may be generated because water is piling up there. This was a significant problem with the Dead Run case, and therefore before running ParFlow on the full domain, we performed an overland flow test.

In this test, pressure at the land surface and streams generated solely from runoff are evaluated (i.e. there is no source of water is from the water table intersecting the land surface). ParFlow users also call this a “parking lot” test because the surface hydraulic conductivity is reduced to a very small value, so that effectively there is no infiltration and rather only overland flow. With a brief, intense storm, the surface pressures are defined by the overland flow and routing is governed by the slopes.

For the Dead Run case, the test was done using the following parameters: dx = dy = 100 m, and dz = 0.1 m with a flat box domain (no solid file). The slopes to be tested were the actual slopes based on elevations. A very small K was used (e.g. K = 0.0001 m/hr) so that all precipitation resulted in runoff. In the Dead Run testing, a storm consisting of time steps of 0.1 hours for 2 hours with precipitation rate for 1 hr of -0.005 m/hr followed by 1 hour of no rain, was used. If disconnected streams are seen in the pressure head output (meaning upstream cells which have a much higher pressure head than downstream cells), this is an indication that there is a problem with the slopes specified in the slopes files.

4.7.2 Channel slope enforcement

The channel slopes enforcing method manipulates the slopes in the stream channels so that each stream points directly to its next downstream stream cell neighbor. We wrote a MATLAB code “enforceslopes.m” to take a raster file of the streams from the NHD (USGS National Hydrography Dataset) and connect slopes of the stream cells. The code can also accept branching stream networks and multiple streams within the domain. The slope direction is modified, and the slope magnitude is increased. The process done by the enforceslopes.m code is shown in Figure 1.

To create the necessary input files for enforceslopes.m, the streams should first be downloaded for the ParFlow domain from the NHD (http://nhd.usgs.gov/). If necessary, the Strahler stream order system (http://www.horizon-systems.com/nhdplus/StrahlerList.php) can be used to select only the larger rivers. This is not done in the Dead Run example, but with larger cell sizes will often be needed so that there are not multiple streams in cells. Once the streams are loaded into ArcGIS, they can be converted to a raster by using the ArcToolbox menu: conversion tools > to raster > polyline to raster. The raster can then be converted to an ASCII text file called rivers.txt. The top 6 lines of text (ArcGIS header) of this text file should be deleted, and then the text file can be input into the MATLAB code makestreamsfromrasterok.m to make a matrix of only 0’s (non stream cells) and 1’s (stream cells). The MATLAB variable editor can then be used to copy and paste the stream matrix in rivers.txt into separate text files for each stream, with the same dimensions as the original array. Alternately, this can be done in ArcGIS instead by using ‘select by attributes’, and by saving each stream as a separate text file.

Then the enforceslopes.m code needs to be modified for the specific application. The names of the stream text files should be entered, and the variables endingarray, streamlooporder, totalnumstreams, ny and ny should be changed. The code works upstream from the given exit cells (endingarray) to connect all upstream cells to this point for each stream. The order in which the streams are loaded into streamlooporder should be the same as the order in which their ending cells are entered into endingarray. Also, the user should modify the magnitude that the slopes are given (here 0.4) if the scale or background slope magnitude is significantly different from the Dead Run case. The slope magnitude is chosen to be on order of the largest background slopes in the domain. Optional modifications to the code are changing the output filenames and having the slopes enforced on a blank array for easier viewing and testing, instead of on top of the original slopes.

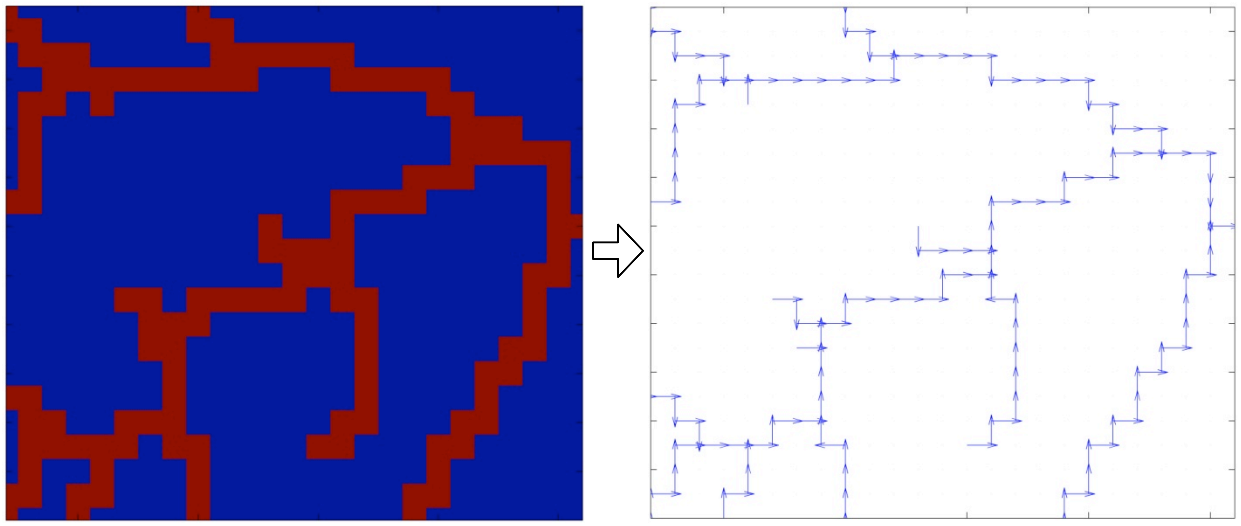


Figure 1. Diagram of the process of slope enforcing. Slope enforcing code uses raster stream information (stream cells with values of 1 are shown in red and non-stream cells with values of 0 are shown in blue) and creates a connected set of x and y slopes along the streams.

After the enforced slope text files are created,"writetextslopesaspfb.f90" is used to convert the output text files (xslopeenf.txt and yslopeenf.txt) to pfb slope files.

In the Dead Run case, even this slope enforcement was not enough to result in fully connected stream channels in the overland flow test. The reason is that because this is an urban landscape, some parts of streams are piped and underground, and therefore not designated in the NHD or DEM. Therefore, we used the Baltimore County and City GIS hydrography data that includes locations of piped streams and storm sewers to connect the daylighted streams. We manually modified the stream raster files using information like that shown in Figure 2.

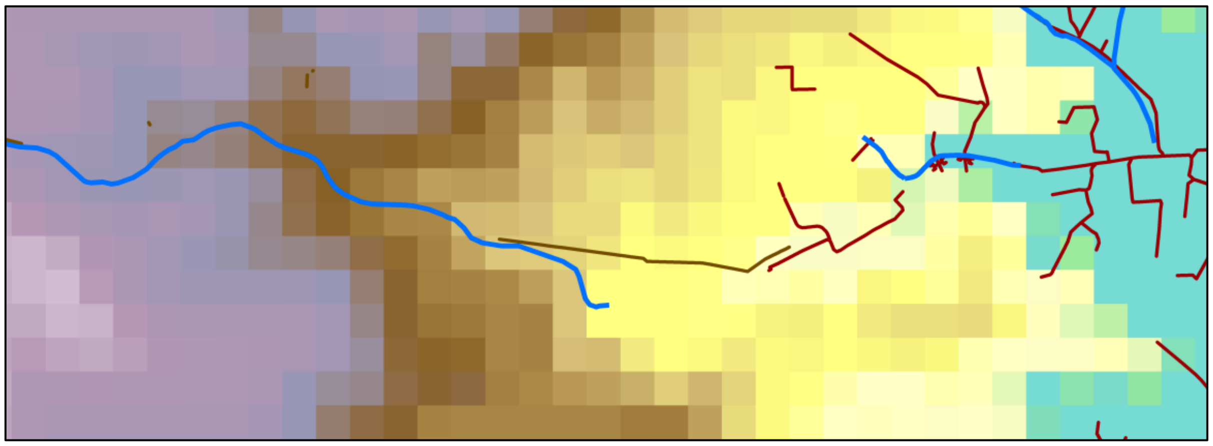


Figure 2. Streams (blue), drainage connectors (brown) and storm sewers (dark red) overlaid on elevation in the southeast part of the Dead Run domain. The center stream abruptly ends; so the flow path is inferred using local storm sewer data.

The resulting enforced slopes that are used in the later Dead Run simulations are shown in Figure 3. The pressure head and saturation resulting from the overland flow test using these slopes are shown in Figure 4. The saturation plot shows that almost the entire domain is fully saturated, as would be expected with such a small infiltration rate. The pressure head plot shows that the pressure head in the stream channels (at the land surface this represents depth of water on the surface) generally smoothly increases from upstream to downstream and are overall well connected. Furthermore, the highest pressure heads found within the domain are within the channels, which means that there are not other cells which have water building up. Once fully connected streams are evident in the pressure output from the overland flow test, the full domain (using the entire depth thickness, more realistic K values and a longer length of simulation) can be run. The example Tcl file for a full domain run is provided in deadrun_unsat.tcl.

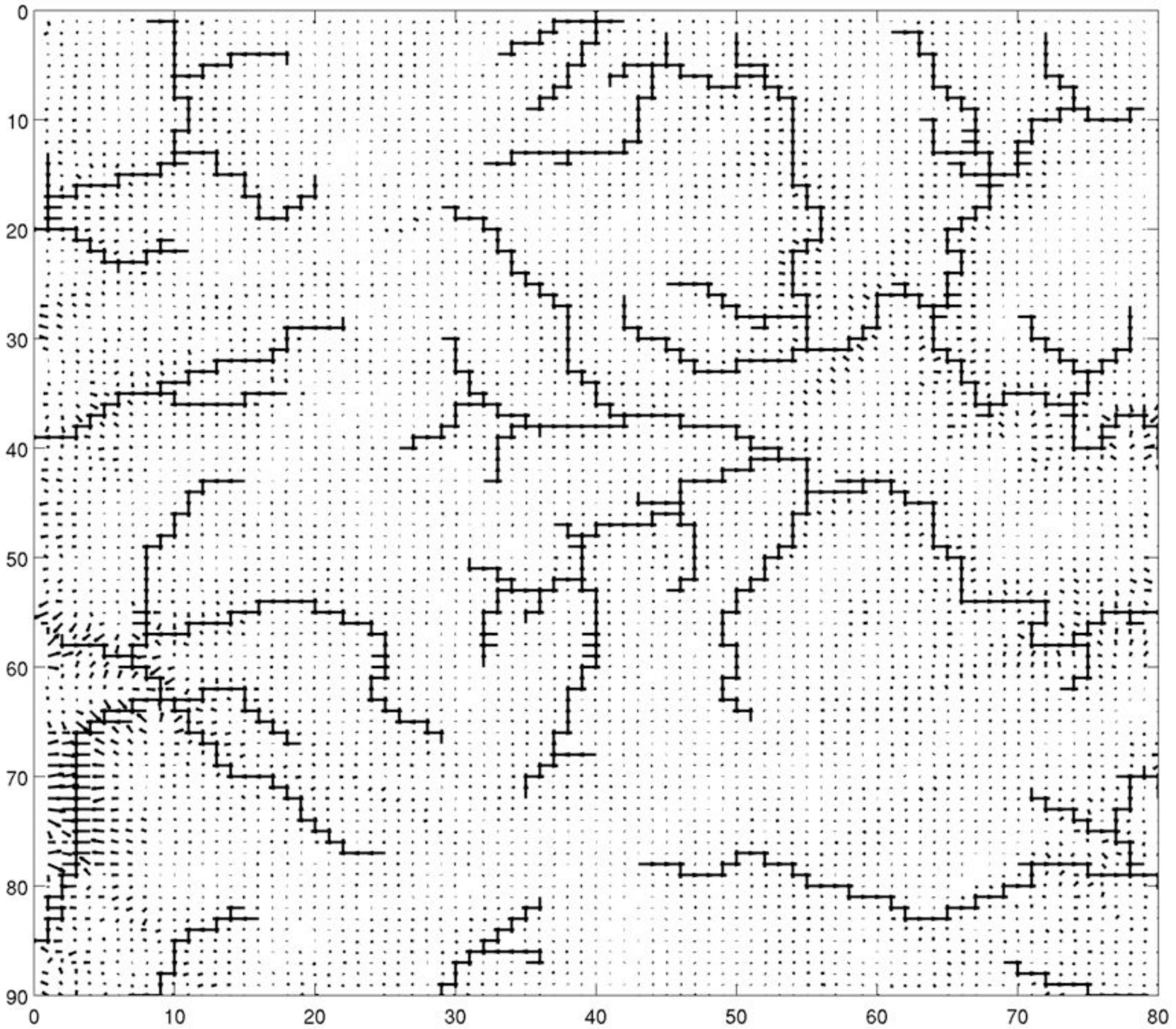


Figure 3. Enforced slopes throughout Dead Run domain.

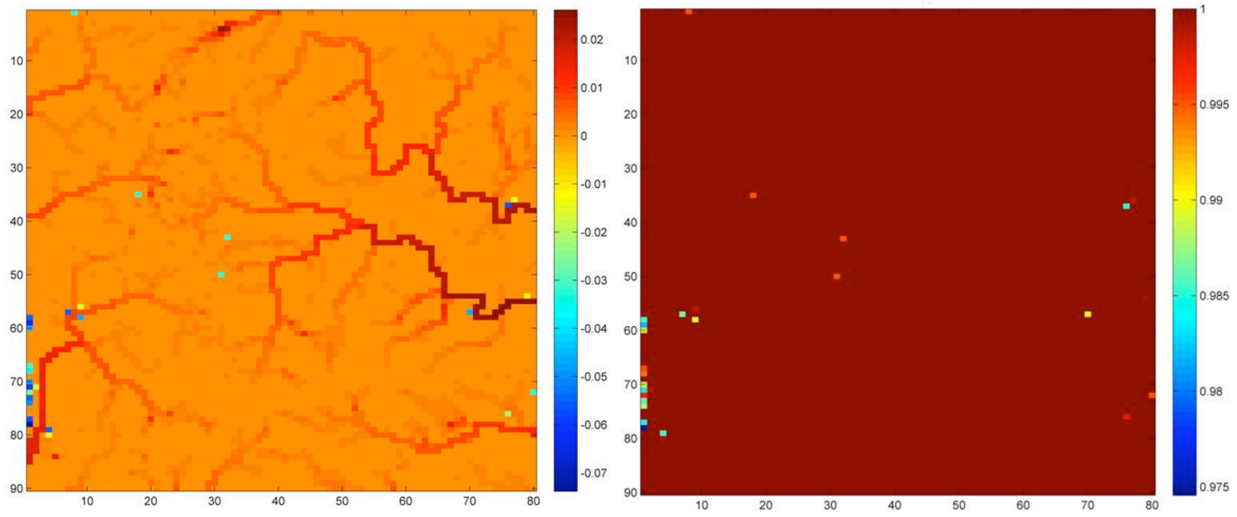


Figure 4. Pressure head (m) (left) and saturation (-) (right), both at the land surface from the Dead Run overland flow test, after 1 hour of heavy rain (0.05 m/hour) followed by 1 hour of recession with K = 0.0001 m/hr.

Table 2. Summary of Steps for reproducing results of the Dead Run overland flow test example starting from the DEM file.

---

1. The DEM dem_pcr.txt for the Dead Run domain has already been downloaded, projected, cropped and resampled to 100 m horizontal resolution. This can be found in both the slopes and solidfile directories.

2. Make the solid file by executing the file solidfile/pf_solid_file_create.f90. Copy the resulting .pfsol file to the overland_flow_test directory. Since the solid file does not make reference to the z dimension (e.g. dz and nz are not specified), the same solid file can be used for the overland flow test (which uses a nz=10) and the full domain unsaturated run later on (which uses a nz=339).

3. Make the slopes file:

    a. Fill sinks using the pit filling method and calculate slopes by executing the make_slopes_pitfill.f90 code. The required input file is the dem_pcr.txt and the output files are xslope.txt, yslope.txt and ned_centralpit.txt.

b. If starting this process from the beginning, the streams for the domain would need to be downloaded from the NHD and separated into text files (as described in section 4.7.2). This process does take some time, so instead the stream files that have been provided can be used.

c. Enforce slopes along the stream channels by running the MATLAB code enforceslopes.m. The input files needed are xslope.txt and yslope.txt. The output of this code are the text files xslopeenf.txt and yslopeenf.txt.

d. Write these text slope files in ParFlow binary format (.pfb) so that they can read by ParFlow. This is done by executing the code writetextslopesaspfb.f90, and which outputs the files xslopeenf_oftest.pfb and yslopeenf_oftest.pfb. These are the inputs to ParFlow and should be copied to the overland_flow_test directory.

4. Run ParFlow by typing 'tclsh deadrun_oftest.tcl' in a command line terminal from within the directory overland_flow_test.

5. After the run completes and the output pfb files and text files (top.press.txt and top.satur.txt) are written out, visualize the text files by using the MATLAB code makeplots.m.

6. If so desired, a plot of the water balance at each timestep can be made using the MATLAB code massbalance.m.

4.8 Description of other examples included

In addition to the overland flow test example, there are two more examples included, for saturated and unsaturated model runs.

The saturated example (deadrun_sat.tcl in the saturated directory) is a saturated, steady-state run using the IMPES solver. Because of this, there is only one output file from this run, pressure head (at all time). The example Tcl file shown is a simple file similar to the harvey.flow.tcl test in the ParFlow tests directory, but uses a solid file to define the domain instead of a box geometry.

The unsaturated example (deadrun_unsat.tcl) uses the Richards solver and is a coupled overland flow – variably saturated model run. It has an initial condition of the water table 5 m below the land surface, and uses the slopes and solid file determined from the DEM as described above. It runs for 75 hours with no rain input. A net recharge rate or rain on the overland flow boundary condition can be included by changing the line, “pfset Patch.z-upper.BCPressure.alltime.Value   0.0” to have a value other than 0.0. Since in this case there is no external input of net recharge, the change over time is the water table equilibrating from its initial condition of 5 m below the land surface throughout the domain. Because the model has only run for 75 hours, the full stream network is not visible, but the beginnings of the network are visible.

Appendix A

Names of codes and input and output files for Dead Run example

File name	Description
makeplots.m	Plot pressure and saturation text files
landsurface.m	Plot pressure and saturation pfb files
slopes/	Directory for slopes creation
fillsinks1_pitfill.f90	Sink filling (pit filling) code
fillinks1_mavg.f90	Sink filling (moving average) code
dem_pcr.txt	Input DEM text file
make_slopes_pitfill.f90	Code 1 for pit filling, write slopes as pfb's
ned_centralpit.txt	Output DEM text file from code 1
xslope.txt	Output x slope text file from code 1
yslope.txt	Output y slope text file from code 1
xslope.pfb	Output x slope pfb file from code 1
yslope.pfb	Output y slope pfb file from code 1
makestreamsfromrasterok.m	Standardize values for NHD streams raster
maidenchoicefixed.txt	Input stream text file for enforceslopes.m
stream1.txt	Input stream text file for enforceslopes.m
stream2.txt	Input stream text file for enforceslopes.m
stream3.txt	Input stream text file for enforceslopes.m
stream4.txt	Input stream text file for enforceslopes.m
stream5.txt	Input stream text file for enforceslopes.m
stream6.txt	Input stream text file for enforceslopes.m
gwynnsfallsfixed.txt	Input stream text file for enforceslopes.m
patapscofixed.txt	Input stream text file for enforceslopes.m
deadrunfixed.txt	Input stream text file for enforceslopes.m
enforceslopes.m	Code to enforce slopes along streams
xslopeenf.txt	Output slope text file from enforceslopes.m
yslopeenf.txt	Output slope text file from enforceslopes.m
writetextslopesaspfb.f90	Code to slope write slope text files as pfb's
xslopeenf_oftest.pfb	Output slope pfb file for overland flow test
yslopeenf_oftest.pfb	Output slope pfb file for overland flow test
xslopeenf_full.pfb	Output slope pfb file for full domain run
yslopeenf_fullpfb	Output slope pfb file for full domain run
solidfile/	Directory for solid file creation
dem_pcr.txt	Input DEM text file
pf_solid_file_create.f90	Solid file creation code
chunk_check.pt	Output text file for visualization in Chunk
chunk_check.tri	Output text file for visualization in Chunk
deadrun.pfsol	Output solid file for ParFlow input
pf_solid_file_create_inactivity.f90	Solid file code for inactive land surface cells

mattocol.m	For above code, writes matrix as columns
saturated/	Directory with saturated example
deadrun.pfsol	Input solid file
deadrun_sat.tcl	ParFlow Tcl script for saturated example
deadrun.out.mask.pfb	Mask pfb from unsaturated run for plotting
deadrun.pfidb	ParFlow output file
deadrun.out.log	ParFlow out.log output file
deadrun.out.perm_x.pfb	ParFlow output x permeability pfb file
deadrun.out.perm_y.pfb	ParFlow output y permeability pfb file
deadrun.out.perm_z.pfb	ParFlow output z permeability pfb file
deadrun.out.pftcl	ParFlow output file
deadrun.out.txt	ParFlow output file
deadrun.out.porosity.pfb	ParFlow output porosity pfb file
deadrun.out.press.pfb	ParFlow output pressure pfb file
top.press.txt	Land surface pressure output text file
makeplots.m	Code to plot land surface pressure
landsurface_pressure.jpg	Plot of land surface pressure output
overland_flow_test/	
xslopeenf_oftest.pfb	Slope file for overland flow test pfb
yslopeenf_oftest.pfb	Slope file for overland flow test pfb
xslopeenf.txt	Text x slope file used to make pfb file
yslopeenf.txt	Text y slope file used to make pfb file
deadrun.pfsol	Input solid file
deadrun_oftest.tcl	Tcl script for overland flow test example
deadrun.out.mannings.silo	ParFlow output Mannings silo file
deadrun.out.mask.silo	ParFlow output mask silo file
deadrun.out.perm_x.silo	ParFlow output x permeability silo file
deadrun.out.perm_y.silo	ParFlow output y permeability silo file
deadrun.out.perm_z.silo	ParFlow output z permeability silo file
deadrun.out.porosity.silo	ParFlow output porosity silo file
deadrun.out.slope_x.silo	ParFlow output x slopes silo file
deadrun.out.slope_y.silo	ParFlow output y slopes silo file
deadrun.out.specific_storage.silo	ParFlow output specific storage silo file
deadrun.pfidb	ParFlow output file
deadrun.out.kinsol.log	ParFlow solver output file
deadrun.out.log	ParFlow output log file
deadrun.out.perm_x.pfb	ParFlow output x permeability pfb file
deadrun.out.perm_y.pfb	ParFlow output y permeability pfb file
deadrun.out.perm_z.pfb	ParFlow output z permeability pfb file
deadrun.out.pftcl	ParFlow output file
deadrun.out.porosity.pfb	ParFlow output porosity pfb file
deadrun.out.press.00000.pfb	ParFlow initial pressure output pfb file
deadrun.out.press.00001.pfb	ParFlow pfb pressure output
deadrun.out.press.00002.pfb	ParFlow pfb pressure output

deadrun.out.press.00003.pfb	ParFlow pfb pressure output
deadrun.out.press.00004.pfb	ParFlow pfb pressure output
deadrun.out.press.00005.pfb	ParFlow pfb pressure output
deadrun.out.press.00006.pfb	ParFlow pfb pressure output
deadrun.out.press.00007.pfb	ParFlow pfb pressure output
deadrun.out.press.00008.pfb	ParFlow pfb pressure output
deadrun.out.press.00009.pfb	ParFlow pfb pressure output
deadrun.out.press.00010.pfb	ParFlow pfb pressure output
deadrun.out.press.00011.pfb	ParFlow pfb pressure output
deadrun.out.press.00012.pfb	ParFlow pfb pressure output
deadrun.out.press.00013.pfb	ParFlow pfb pressure output
deadrun.out.press.00014.pfb	ParFlow pfb pressure output
deadrun.out.press.00015.pfb	ParFlow pfb pressure output
deadrun.out.press.00016.pfb	ParFlow pfb pressure output
deadrun.out.press.00017.pfb	ParFlow pfb pressure output
deadrun.out.press.00018.pfb	ParFlow pfb pressure output
deadrun.out.press.00019.pfb	ParFlow pfb pressure output
deadrun.out.press.00020.pfb	ParFlow pfb pressure output
deadrun.out.press.00021.pfb	ParFlow pfb pressure output
deadrun.out.satur.00000.pfb	ParFlow initial saturation output pfb file
deadrun.out.satur.00001.pfb	ParFlow pfb saturation output
deadrun.out.satur.00002.pfb	ParFlow pfb saturation output
deadrun.out.satur.00003.pfb	ParFlow pfb saturation output
deadrun.out.satur.00004.pfb	ParFlow pfb saturation output
deadrun.out.satur.00005.pfb	ParFlow pfb saturation output
deadrun.out.satur.00006.pfb	ParFlow pfb saturation output
deadrun.out.satur.00007.pfb	ParFlow pfb saturation output
deadrun.out.satur.00008.pfb	ParFlow pfb saturation output
deadrun.out.satur.00009.pfb	ParFlow pfb saturation output
deadrun.out.satur.00010.pfb	ParFlow pfb saturation output
deadrun.out.satur.00011.pfb	ParFlow pfb saturation output
deadrun.out.satur.00012.pfb	ParFlow pfb saturation output
deadrun.out.satur.00013.pfb	ParFlow pfb saturation output
deadrun.out.satur.00014.pfb	ParFlow pfb saturation output
deadrun.out.satur.00015.pfb	ParFlow pfb saturation output
deadrun.out.satur.00016.pfb	ParFlow pfb saturation output
deadrun.out.satur.00017.pfb	ParFlow pfb saturation output
deadrun.out.satur.00018.pfb	ParFlow pfb saturation output
deadrun.out.satur.00019.pfb	ParFlow pfb saturation output
deadrun.out.satur.00020.pfb	ParFlow pfb saturation output
deadrun.out.satur.00021.pfb	ParFlow pfb saturation output
deadrun.out.txt	ParFlow output text file
deadrun.out.mask.pfb	ParFlow output mask pfb file from solid file
top.press.txt	Land surface pressure output text file

top.satur.txt	Land surface saturation output text file
makeplots.m	Code to plot pressure and saturation
massbalance.m	Code to plot mass balance
landsurface_saturation_and_pressure.jpg	Plot of land surface saturation and pressure

unsaturated/

deadrun.pfsol	Input solid file
xslopeenf_full.pfb	Input x slope pfb file
yslopeenf_full.pfb	Input y slope pfb file
xslopeenf.txt	Text x slope file used to make pfb file
yslopeenf.ftxt	Text y slope file used to make pfb file
deadrun_unsat.tcl	ParFlow Tcl script for unsaturated example
deadrun.out.perm_x.pfb	ParFlow output x permeability pfb file
deadrun.out.perm_y.pfb	ParFlow output y permeability pfb file
deadrun.out.perm_z.pfb	ParFlow output z permeability pfb file
deadrun.out.porosity.pfb	ParFlow output porosity pfb file
deadrun.out.mask.pfb	ParFlow mask file from solid file
deadrun.out.press.00075.pfb	ParFlow pressure file from 75th timestep
deadrun.out.satur.00075.pfb	ParFlow saturation file from 75th timestep
(other timesteps' outputs were not included to reduce the total file size)	
top.press.txt	Land surface pressure output text file
top.satur.txt	Land surface saturation output text file
makeplots.m	Code to plot pressure and saturation
massbalance.m	Code to plot mass balance
landsurface_saturation_and_pressure.jpg	Plot of land surface saturation and pressure