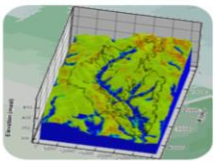
Parflow Model Installation

Professor: Xiaonan Tai, Assistant Professor, Biological Sciences

Report by: Satyam Pandey, MS, Computer Science

parflow/**parflow**

Parflow is an open-source parallel watershed flow model.



32 Contributors | 11 Used by | 134 Stars | 80 Forks

The objective of this report is to showcase the capabilities of ParFlow, an open-source, modular, and parallel watershed flow model, through its execution and analysis. Specifically, the report focuses on simulating the hydrological processes within the Washita watershed, utilizing ParFlow's advanced features, such as its ability to handle complex topography, geology, and heterogeneity, as well as its integration with land-surface processes like the land-energy budget, biogeochemistry, and snow simulation via CLM. Detailed instructions on executing the ParFlow code, including configuration steps, are provided. The report presents the results of the ParFlow execution, incorporating analysis, and executed results upon successful installation of the model. It concludes by discussing the implications of the findings, emphasizing the strengths and limitations of ParFlow, and suggesting future research directions and improvements in the field of hydrological modeling.

Introduction:

ParFlow is a hydrology model that integrates surface and subsurface flow simulations. It offers three operational modes: steady-state saturated, variably saturated, and integrated-watershed flow. Developed over a span of more than 20 years, ParFlow has evolved into a robust parallel simulation platform suitable for addressing large-scale hydrological problems across diverse computing platforms.

By utilizing advanced numerical solvers and multigrid preconditioners, ParFlow accurately models saturated and variably saturated subsurface flow in three dimensions. It employs a conjugate gradient solver with multigrid preconditioning and a Newton-Krylov nonlinear solver. A notable advantage of ParFlow is its ability to seamlessly incorporate surface-subsurface flow interactions, enabling comprehensive simulations of hillslope runoff and channel routing. Furthermore, ParFlow is fully integrated with the CLM land surface model, enabling the representation of complex land surface processes.

Primarily implemented in C, ParFlow exhibits a modular architecture that facilitates flexibility and adaptability. It incorporates a communication layer for efficient parallel process interactions and is composed of over 190 source files organized into a main executable and library structure. ParFlow can be invoked as a stand-alone application or integrated within other models like the Weather Research and Forecasting atmospheric model (WRF). Notably, ParFlow leverages an advanced octree-space partitioning algorithm to accurately depict intricate three-dimensional structures, including topography, hydrologic facies, and watershed boundaries. This comprehensive set of features equips ParFlow for conducting high-resolution, large-scale watershed simulations, making it a valuable tool for hydrological analysis and research.

Installing ParFlow on Ubuntu:

1. To prepare Linux system for development, I started by obtaining the latest package information and installing several new packages. Executed the following commands in my terminal:

```
sudo apt-get update

sudo apt-get install \
  build-essential \
  curl \
  git \
  gfortran \
  libopenblas-dev \
  liblapack-dev \
  openssh-client \
  openssh-server \
  openmpi-bin \
  libopenmpi-dev \
  tcl-dev \
  tk-dev
```

2. To set up the necessary directory structure for ParFlow, followed below steps. The example assumes that we want to install ParFlow and its dependencies in the home (~) directory, but we can choose a different location if desired.

mkdir -p ~/parflow/build \

~/parflow/dependencies/cmake \

~/parflow/dependencies/hypre-src

This will create the following directory structure:

~/parflow/build - Directory for building ParFlow

~/parflow/dependencies/cmake - Directory for the CMake dependency

~/parflow/dependencies/hypre-src - Directory for the Hypre source code

3. Fetching the dependencies for ParFlow. We'll start by downloading CMake, with version 3.17.3 as the current version. First, navigate to the CMake directory by running the command cd ~/parflow/dependencies/cmake. Then, use the following command to download and extract the CMake package: curl -L https://cmake.org/files/v3.17/cmake-3.17.3-Linux-x86_64.tar.gz | tar --strip-components=1 -xzv.

Great! Now that we have CMake, let's install Hypre, with version 2.19.0 being the current version. Change to the Hypre directory by running cd ~/parflow/dependencies/hypre-src. Next, download and extract the Hypre package using the command curl -L https://github.com/hypre-space/hypre/archive/v2.19.0.tar.gz | tar --strip-components=1 -xzv. Once that's done, navigate to the 'src' directory with cd src.

To configure Hypre for installation, run the following command: ./configure --prefix=~/parflow/dependencies/hypre --with-MPI. Finally, let's install Hypre by executing make install.

And there you have it! We've successfully fetched and installed the necessary dependencies, CMake and Hypre, for ParFlow.

4. Downloading and building ParFlow. We can get the latest release, which is v3.6.0, from the ParFlow GitHub repository. If you're using Ubuntu 20.04, remember to replace "v3.6.0" with "master" to clone the compatible master branch. To download ParFlow, we can execute the following command:

git clone --single-branch --branch v3.6.0 --recursive https://github.com/parflow/parflow.git ~/parflow/src

Now that we have ParFlow downloaded, it's time to build it. Run the following commands to initiate the build process:

~/parflow/dependencies/cmake/bin/cmake -S ~/parflow/src -B ~/parflow/build -D PARFLOW_AMPS_LAYER=mpi1 -D PARFLOW_AMPS_SEQUENTIAL_IO=TRUE -D HYPRE_ROOT=~/parflow/dependencies/hypre -D PARFLOW_ENABLE_TIMING=TRUE -D PARFLOW_HAVE_CLM=TRUE

~/parflow/dependencies/cmake/bin/cmake --build ~/parflow/build

~/parflow/dependencies/cmake/bin/cmake --install ~/parflow/build --prefix ~/parflow/install

By running these commands, ParFlow will be built successfully. The installation will be located in the ~/parflow/install directory.

5. Let's create an environment file called 'env.sh' to conveniently store the ParFlow directory. This will make it easier for us to reference ParFlow when we want to run it later. Just type the following command:

echo export PARFLOW_DIR=~/parflow/install >> ~/parflow/env.sh

Now that we have the environment file set up, we can use ParFlow from anywhere by simply sourcing it. Here's the command to do that:

source ~/parflow/env.sh

By sourcing the 'env.sh' file, ParFlow will be available for use in any location. This will make it more convenient for us to run ParFlow whenever we need it.

6. Testing ParFlow - Create an environment file: To make it easier to reference ParFlow when running it later, let's create an environment file called "env.sh" to store the ParFlow directory. Simply type the following command:

echo export PARFLOW_DIR=~/parflow/install >> ~/parflow/env.sh

By doing this, we can conveniently use ParFlow from anywhere by sourcing the environment file:

source ~/parflow/env.sh

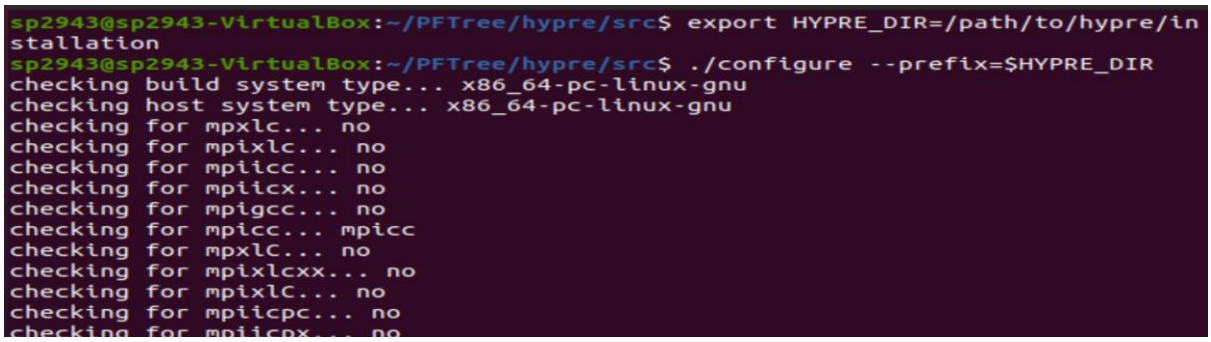
Test ParFlow: Assuming the installation went smoothly without any errors, it's time to test ParFlow. Start by sourcing the environment file we just created, and then execute the following commands:

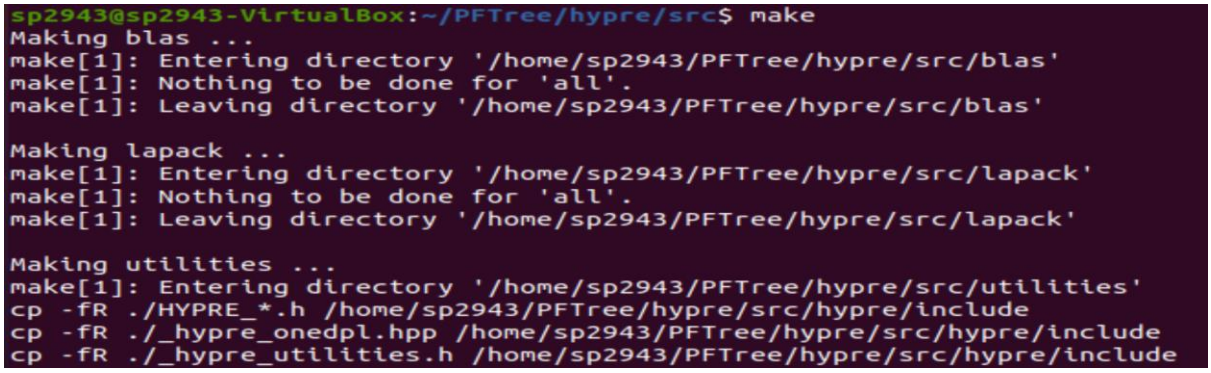
cd ~/parflow/build/

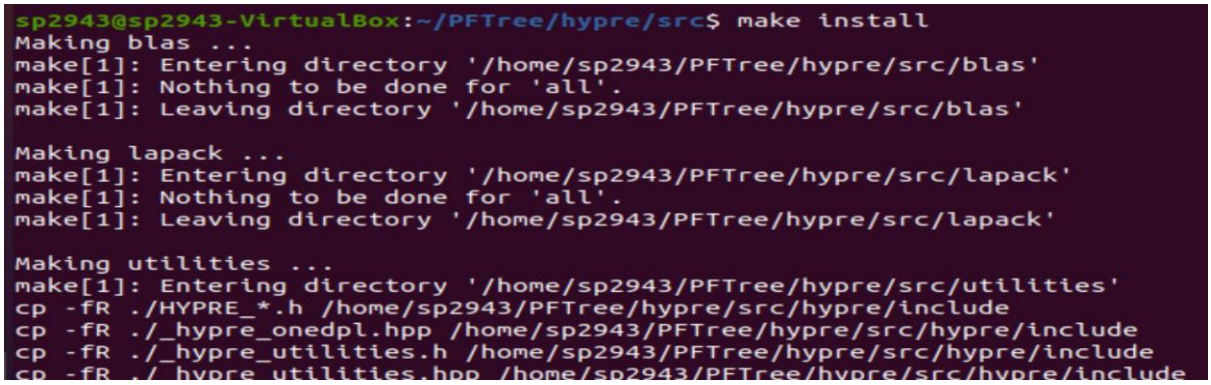
~/parflow/dependencies/cmake/bin/ctest –VV

These commands will run a series of tests to ensure that your ParFlow installation is functioning correctly. However, please note that there might be nine tests that fail due to a bug related to referencing .silo files. It's important to mention that this installation guide does not cover the installation of Silo. The tests that may fail due to this bug include: water_balance_x.hardflow.nojac.tcl (all topologies), water_balance_x.hardflow.jac.tcl (all topologies), and impes.plinear.tcl

Installation:







```
sp2943@sp2943-VirtualBox:~/PFTree/hypre/src$ cd hypre
sp2943@sp2943-VirtualBox:~/PFTree/hypre/src/hypre$ ./configure --prefix=$HYPRE_
DIR
bash: ./configure: No such file or directory
sp2943@sp2943-VirtualBox:~/PFTree/hypre/src/hypre$ make
make: *** No targets specified and no makefile found.  Stop.
sp2943@sp2943-VirtualBox:~/PFTree/hypre/src/hypre$ cd ..
sp2943@sp2943-VirtualBox:~/PFTree/hypre/src$ ./configure --prefix=$HYPRE_DIR
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking for mpxlc... no
checking for mpixlc... no
checking for mpiicc... no
checking for mpiicx... no
checking for mpigcc... no
checking for mpicc... mpicc
checking for mpxlC... no
```

```
sp2943@sp2943-VirtualBox:~/PFTree/hypre/src$ make
Making blas ...
make[1]: Entering directory '/home/sp2943/PFTree/hypre/src/blas'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/home/sp2943/PFTree/hypre/src/blas'

Making lapack ...
make[1]: Entering directory '/home/sp2943/PFTree/hypre/src/lapack'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/home/sp2943/PFTree/hypre/src/lapack'

Making utilities ...
make[1]: Entering directory '/home/sp2943/PFTree/hypre/src/utilities'
cp -fR ./HYPRE_*.h /home/sp2943/PFTree/hypre/src/hypre/include
cp -fR ./_hypre_onedpl.hpp /home/sp2943/PFTree/hypre/src/hypre/include
cp -fR ./_hypre_utilities.h /home/sp2943/PFTree/hypre/src/hypre/include
cp -fR ./_hypre_utilities.hpp /home/sp2943/PFTree/hypre/src/hypre/include
cp -fR ./fortran* h /home/sp2943/PFTree/hypre/src/hypre/include
```

```
sp2943@sp2943-VirtualBox:~/PFTree/hypre/src$ make install
Making blas ...
make[1]: Entering directory '/home/sp2943/PFTree/hypre/src/blas'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/home/sp2943/PFTree/hypre/src/blas'

Making lapack ...
make[1]: Entering directory '/home/sp2943/PFTree/hypre/src/lapack'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/home/sp2943/PFTree/hypre/src/lapack'

Making utilities ...
make[1]: Entering directory '/home/sp2943/PFTree/hypre/src/utilities'
cp -fR ./HYPRE_*.h /home/sp2943/PFTree/hypre/src/hypre/include
cp -fR ./_hypre_onedpl.hpp /home/sp2943/PFTree/hypre/src/hypre/include
```

```
sp2943@sp2943-VirtualBox:~/PFTree/hypre/src$ sudo make install
[sudo] password for sp2943:
Making blas ...
make[1]: Entering directory '/home/sp2943/PFTree/hypre/src/blas'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/home/sp2943/PFTree/hypre/src/blas'

Making lapack ...
make[1]: Entering directory '/home/sp2943/PFTree/hypre/src/lapack'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/home/sp2943/PFTree/hypre/src/lapack'

Making utilities ...
make[1]: Entering directory '/home/sp2943/PFTree/hypre/src/utilities'
cp -fR ./HYPRE_*.h /home/sp2943/PFTree/hypre/src/hypre/include
cp -fR ./_hypre_onedpl.hpp /home/sp2943/PFTree/hypre/src/hypre/include
```

```
sp2943@sp2943-VirtualBox:~/PFTree/hypre/src$ cd silo
bash: cd: silo: No such file or directory
sp2943@sp2943-VirtualBox:~/PFTree/hypre/src$ cd ..
sp2943@sp2943-VirtualBox:~/PFTree/hypre$ cd ..
sp2943@sp2943-VirtualBox:~/PFTree$ cd silo
sp2943@sp2943-VirtualBox:~/PFTree/silo$ ./configure --disable-silex
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking whether to enable maintainer-specific portions of Makefiles... no
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
```

```
SUMMARY OF THE Silo CONFIGURATION
=================================

Silo Version:              4.10.2
Configured on:             Sat Jun 10 16:22:52 EDT 2023
Configured by:             sp2943@sp2943-VirtualBox
Configure command:         ./configure '--disable-silex'
Host system:               x86_64-unknown-linux-gnu
Build system:              x86_64-unknown-linux-gnu
Installation point:        /home/sp2943/PFTree/silo
Source directory:          @UsingSrcDir@
Archiver:                  ar
Ranlib:                    ranlib

Configure Summary
Compiling Options:
                       C Compiler /usr/bin/gcc
                         CPPFLAGS
```

```
sp2943@sp2943-VirtualBox:~/PFTree/silo$ sudo make
make  all-recursive
make[1]: Entering directory '/home/sp2943/PFTree/silo'
Making all in .
make[2]: Entering directory '/home/sp2943/PFTree/silo'
make[2]: Leaving directory '/home/sp2943/PFTree/silo'
Making all in src
make[2]: Entering directory '/home/sp2943/PFTree/silo/src'
make  all-recursive
make[3]: Entering directory '/home/sp2943/PFTree/silo/src'
Making all in score
make[4]: Entering directory '/home/sp2943/PFTree/silo/src/score'
make[4]: Nothing to be done for 'all'.
make[4]: Leaving directory '/home/sp2943/PFTree/silo/src/score'
Making all in pdb
make[4]: Entering directory '/home/sp2943/PFTree/silo/src/pdb'
make  all-am
make[5]: Entering directory '/home/sp2943/PFTree/silo/src/pdb'
make[5]: Nothing to be done for 'all-am'.
make[5]: Leaving directory '/home/sp2943/PFTree/silo/src/pdb'
make[4]: Leaving directory '/home/sp2943/PFTree/silo/src/pdb'
Making all in silo
make[4]: Entering directory '/home/sp2943/PFTree/silo/src/silo'
make  all-am
```

```
sp2943@sp2943-VirtualBox:~/PFTree/silo$ sudo make install
Making install in .
make[1]: Entering directory '/home/sp2943/PFTree/silo'
make[2]: Entering directory '/home/sp2943/PFTree/silo'
make[2]: Nothing to be done for 'install-exec-am'.
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/home/sp2943/PFTree/silo'
make[1]: Leaving directory '/home/sp2943/PFTree/silo'
Making install in src
make[1]: Entering directory '/home/sp2943/PFTree/silo/src'
make  install-recursive
make[2]: Entering directory '/home/sp2943/PFTree/silo/src'
Making install in score
make[3]: Entering directory '/home/sp2943/PFTree/silo/src/score'
make[4]: Entering directory '/home/sp2943/PFTree/silo/src/score'
make[4]: Nothing to be done for 'install-exec-am'.
test -z "/home/sp2943/PFTree/silo/include" || /usr/bin/mkdir -p "
FTree/silo/include"
make[4]: Leaving directory '/home/sp2943/PFTree/silo/src/score'
make[3]: Leaving directory '/home/sp2943/PFTree/silo/src/score'
Making install in pdb
make[3]: Entering directory '/home/sp2943/PFTree/silo/src/pdb'
make  install-am
```

```
sp2943@sp2943-VirtualBox:~/PFTree/silo$ cd ~/PFTree
sp2943@sp2943-VirtualBox:~/PFTree$ cd build
sp2943@sp2943-VirtualBox:~/PFTree/build$ sudo cmake ../parflow -DCMAKE_INSTALL_
PREFIX=$PARFLOW_DIR -DHYPRE_ROOT=$HYPRE_DIR -DPARFLOW_AMPS_LAYER=mpi1 -DPARFLOW
_AMPS_SEQUENTIAL_IO=true -DPARFLOW_ENABLE_TIMING=true -DSILO_ROOT=$SILO_DIR -DP
ARFLOW_HAVE_CLM=ON -DTCL_INCLUDE_PATH=/usr/include/tcl8.6
-- Configuring version : v3.12.0-8-g9f66aab
-- Using single file AMPS I/O for PFB output
-- Could NOT find Silo (missing: SILO_LIBRARIES SILO_INCLUDE_DIRS)
-- Found Hypre: /path/to/hypre/installation/lib/libHYPRE.a
-- Checking Fortran implicit none flag : -implicitnone
-- Checking Fortran implicit none flag : -fimplicit-none
-- Configuring done
```

```
sp2943@sp2943-VirtualBox:~/PFTree/build$ sudo make
[ 19%] Built target pfclm
Scanning dependencies of target amps
[ 19%] Building C object pfsimulator/amps/CMakeFiles/amps.dir/common/amps_abort
.c.o
[ 19%] Building C object pfsimulator/amps/CMakeFiles/amps.dir/common/amps_clear
.c.o
[ 19%] Building C object pfsimulator/amps/CMakeFiles/amps.dir/common/amps_clock
.c.o
[ 20%] Building C object pfsimulator/amps/CMakeFiles/amps.dir/common/amps_excha
nge.c.o
[ 20%] Building C object pfsimulator/amps/CMakeFiles/amps.dir/common/amps_ffope
n.c.o
[ 20%] Building C object pfsimulator/amps/CMakeFiles/amps.dir/common/amps_find_
powers.c.o
```

```
[ 28%] Linking C static library libamps.a
[ 28%] Built target amps
Scanning dependencies of target test15
[ 29%] Building C object pfsimulator/amps/test/src/CMakeFiles/test1
.c.o
[ 29%] Linking C executable test15
[ 29%] Built target test15
Scanning dependencies of target test16
[ 29%] Building C object pfsimulator/amps/test/src/CMakeFiles/test1
.c.o
[ 29%] Linking C executable test16
[ 29%] Built target test16
Scanning dependencies of target test14
[ 29%] Building C object pfsimulator/amps/test/src/CMakeFiles/test1
.c.o
[ 29%] Linking C executable test14
[ 29%] Built target test14
Scanning dependencies of target test1
[ 30%] Building C object pfsimulator/amps/test/src/CMakeFiles/test1
.o
[ 30%] Linking C executable test1
```

```
[ 84%] Built target pfsimulator
Scanning dependencies of target parflow
[ 84%] Building C object pfsimulator/parflow_exe/CMakeFiles/parflow.dir/main.c.
o
[ 84%] Linking C executable parflow
[ 84%] Built target parflow
[ 85%] Linking C executable pfwell_cat
[ 85%] Built target pfwell_cat
[ 86%] Linking C executable pfwell_data
[ 86%] Built target pfwell_data
[ 86%] Linking C executable gmssol2pfsol
[ 87%] Built target gmssol2pfsol
[ 87%] Linking C executable gmstinvertices
[ 88%] Built target gmstinvertices
[ 88%] Linking C executable projecttin
[ 88%] Built target projecttin
[ 88%] Linking C executable gmstriangulate
```

```
Scanning dependencies of target pftools
[ 91%] Building C object pftools/CMakeFiles/pftools.dir/pftappinit.c.o
[ 92%] Building C object pftools/CMakeFiles/pftools.dir/printdatabox.c.o
[ 92%] Building C object pftools/CMakeFiles/pftools.dir/readdatabox.c.o
[ 92%] Building C object pftools/CMakeFiles/pftools.dir/databox.c.o
[ 92%] Building C object pftools/CMakeFiles/pftools.dir/error.c.o
[ 93%] Building C object pftools/CMakeFiles/pftools.dir/velocity.c.o
[ 93%] Building C object pftools/CMakeFiles/pftools.dir/head.c.o
[ 93%] Building C object pftools/CMakeFiles/pftools.dir/flux.c.o
[ 94%] Building C object pftools/CMakeFiles/pftools.dir/diff.c.o
[ 94%] Building C object pftools/CMakeFiles/pftools.dir/stats.c.o
[ 94%] Building C object pftools/CMakeFiles/pftools.dir/axpy.c.o
[ 95%] Building C object pftools/CMakeFiles/pftools.dir/getsubbox.c.o
[ 95%] Building C object pftools/CMakeFiles/pftools.dir/enlargebox.c.o
[ 95%] Building C object pftools/CMakeFiles/pftools.dir/load.c.o
[ 96%] Building C object pftools/CMakeFiles/pftools.dir/usergrid.c.o
[ 96%] Building C object pftools/CMakeFiles/pftools.dir/grid.c.o
[ 96%] Building C object pftools/CMakeFiles/pftools.dir/region.c.o
```

```
sp2943@sp2943-VirtualBox:~/PFTree/build$ sudo make install
[ 19%] Built target pfclm
[ 28%] Built target amps
[ 29%] Built target test15
[ 29%] Built target test16
[ 29%] Built target test14
[ 30%] Built target test1
[ 31%] Built target test4
[ 32%] Built target test17
[ 32%] Built target test3
[ 32%] Built target test12
[ 33%] Built target test2
[ 33%] Built target test7
[ 34%] Built target test8
[ 35%] Built target test13
[ 35%] Built target test5
[ 36%] Built target test9
[ 37%] Built target test10
[ 38%] Built target test6
[ 40%] Built target pfkinsol
[ 41%] Built target cjson
[ 84%] Built target pfsimulator
[ 84%] Built target parflow
[ 85%] Built target pfwell_cat
[ 86%] Built target pfwell_data
[ 87%] Built target gmssol2pfsol
[ 88%] Built target gmstinvertices
[ 88%] Built target projecttin
```

```
sp2943@sp2943-VirtualBox:~/PFTree/build$ make test
Running tests...
Test project /home/sp2943/PFTree/build
        Start   1: amps-test1--1-1
  1/213 Test   #1: amps-test1--1-1 ................................   Passed
   0.59 sec
        Start   2: amps-test4--1-1
  2/213 Test   #2: amps-test4--1-1 ................................   Passed
   0.63 sec
        Start   3: amps-test5--1-1
  3/213 Test   #3: amps-test5--1-1 ................................   Passed
   0.64 sec
        Start   4: amps-test7--1-1
  4/213 Test   #4: amps-test7--1-1 ................................   Passed
   0.58 sec
        Start   5: amps-test8--1-1
  5/213 Test   #5: amps-test8--1-1 ................................   Passed
```

```
sp2943@sp2943-VirtualBox:~/PFTree/build$ export PARFLOW_DIR=/home/snoopy/parflo
w
sp2943@sp2943-VirtualBox:~/PFTree/build$ cd ~/parflow
sp2943@sp2943-VirtualBox:~/parflow$ tar -xvf ../parflow.tar.gz
tar: ../parflow.tar.gz: Cannot open: No such file or directory
tar: Error is not recoverable: exiting now
sp2943@sp2943-VirtualBox:~/parflow$ gunzip ../parflow.tar.gz
gzip: ../parflow.tar.gz: No such file or directory
sp2943@sp2943-VirtualBox:~/parflow$ setenv PARFLOW_DIR /home/snoopy/parflow
```

```
sp2943@sp2943-VirtualBox:~/parflow$ export PARFLOW_DIR=/home/snoopy
sp2943@sp2943-VirtualBox:~/parflow$ gunzip ../parflow.tar.gz
gzip: ../parflow.tar.gz: No such file or directory
sp2943@sp2943-VirtualBox:~/parflow$ gunzip ../parflow.tar.xz
gzip: ../parflow.tar.xz.gz: No such file or directory
sp2943@sp2943-VirtualBox:~/parflow$ gunzip ../parflow.tar.xz
gzip: ../parflow.tar.xz.gz: No such file or directory
sp2943@sp2943-VirtualBox:~/parflow$ gunzip ../parflow.tar
gzip: ../parflow.tar.gz: No such file or directory
sp2943@sp2943-VirtualBox:~/parflow$ tar -xvf ../parflow.tar
tar: ../parflow.tar: Cannot open: No such file or directory
tar: Error is not recoverable: exiting now
sp2943@sp2943-VirtualBox:~/parflow$ cd ..
sp2943@sp2943-VirtualBox:~$ cd PFTree
sp2943@sp2943-VirtualBox:~/PFTree$ cd parflow
sp2943@sp2943-VirtualBox:~/PFTree/parflow$ mkdir build
mkdir: cannot create directory ‘build’: File exists
sp2943@sp2943-VirtualBox:~/PFTree/parflow$ cd build
sp2943@sp2943-VirtualBox:~/PFTree/parflow/build$ ccmake ../parflow
```

```
sp2943@sp2943-VirtualBox:~/PFTree/parflow/build$ sudo snap install cmake
[sudo] password for sp2943:
error: cannot install "cmake": persistent network error: Post
       https://api.snapcraft.io/v2/snaps/refresh: dial tcp: lookup
       api.snapcraft.io: Temporary failure in name resolution
sp2943@sp2943-VirtualBox:~/PFTree/parflow/build$ sudo snap install ccmake
error: cannot install "ccmake": persistent network error: Post
       https://api.snapcraft.io/v2/snaps/refresh: dial tcp: lookup
       api.snapcraft.io: Temporary failure in name resolution
sp2943@sp2943-VirtualBox:~/PFTree/parflow/build$ ccmake ../parflow
```

```
sp2943@sp2943-VirtualBox:~/PFTree/parflow/build$ sudo snap install ccmake
error: snap "ccmake" not found
sp2943@sp2943-VirtualBox:~/PFTree/parflow/build$ sudo apt install cmake-curses-
gui
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  gir1.2-goa-1.0 libfprint-2-tod1 libfwupdplugin1 libllvm9 libxmlb1
  linux-modules-5.4.0-1002-oem ubuntu-system-service
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  cmake-curses-gui
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 1,768 kB of archives.
After this operation, 5,595 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu focal-updates/universe amd64 cmake-cu
rses-gui amd64 3.16.3-1ubuntu1.20.04.1 [1,768 kB]
Fetched 1,768 kB in 3s (536 kB/s)
Selecting previously unselected package cmake-curses-gui.
(Reading database ... 202038 files and directories currently installed.)
Preparing to unpack .../cmake-curses-gui_3.16.3-1ubuntu1.20.04.1_amd64.deb ...
Unpacking cmake-curses-gui (3.16.3-1ubuntu1.20.04.1) ...
Setting up cmake-curses-gui (3.16.3-1ubuntu1.20.04.1) ...
```

```
sp2943@sp2943-VirtualBox:~/PFTree/parflow/build$ ccmake ../parflow


sp2943@sp2943-VirtualBox:~/PFTree/parflow/build$ cmake ../parflow \
>       -DCMAKE_INSTALL_PREFIX=${PARFLOW_DIR} \
>       -DPARFLOW_HAVE_CLM=ON
-- The C compiler identification is GNU 9.4.0
-- The Fortran compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting Fortran compiler ABI info
-- Detecting Fortran compiler ABI info - done
-- Check for working Fortran compiler: /usr/bin/gfortran - skipped
-- Checking whether /usr/bin/gfortran supports Fortran 90
-- Checking whether /usr/bin/gfortran supports Fortran 90 - yes
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
```

```
sp2943@sp2943-VirtualBox:~/PFTree/parflow/build$ DTCL_TCLSH=${PARFLOW_TCL_DIR}/
bin/tclsh8.6
sp2943@sp2943-VirtualBox:~/PFTree/parflow/build$ cmake ../parflow \
>         -DCMAKE_INSTALL_PREFIX=${PARFLOW_DIR} \
>      -DPARFLOW_HAVE_CLM=ON \
>   -DPARFLOW_AMPS_LAYER=mpi1
-- Configuring version : v3.12.0-8-g9f66aab
-- Using single file AMPS I/O for PFB output
-- Found MPI_C: /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so (found version
"3.1")
-- Found MPI_CXX: /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi_cxx.so (found ve
rsion "3.1")
-- Found MPI_Fortran: /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi_usempif08.so
 (found version "3.1")
-- Found MPI: TRUE (found version "3.1")
-- Could NOT find Silo (missing: SILO_LIBRARIES SILO_INCLUDE_DIRS)
-- Could NOT find Hypre (missing: HYPRE_LIBRARIES HYPRE_INCLUDE_DIRS)
-- Checking Fortran implicit none flag : -implicitnone
-- Checking Fortran implicit none flag : -fimplicit-none
-- Configuring done
-- Generating done
-- Build files have been written to: /home/sp2943/PFTree/parflow/build
```

```
sp2943@sp2943-VirtualBox:~/PFTree/parflow/build$ cmake ../parflow \
>         -DPARFLOW_AMPS_LAYER=mpi1 \
> -DHYPRE_ROOT=${PARFLOW_HYPRE_DIR} \
> -DHDF5_ROOT=${PARFLOW_HDF5_DIR} \
> -DSILO_ROOT=${PARFLOW_SILO_DIR} \
> -DCMAKE_BUILD_TYPE=Debug \
> -DPARFLOW_ENABLE_TIMING=TRUE \
> -DPARFLOW_HAVE_CLM=ON \
> -DCMAKE_INSTALL_PREFIX=${INSTALL_DIR}
-- Configuring version : v3.12.0-8-g9f66aab
-- Using single file AMPS I/O for PFB output
-- Could NOT find Silo (missing: SILO_LIBRARIES SILO_INCLUDE_DIRS)
-- HDF5: Using hdf5 compiler wrapper to determine C configuration
-- Found HDF5: /usr/lib/x86_64-linux-gnu/hdf5/serial/libhdf5.so;/usr/lib/x86_64
-linux-gnu/libpthread.so;/usr/lib/x86_64-linux-gnu/libsz.so;/usr/lib/x86_64-lin
ux-gnu/libz.so;/usr/lib/x86_64-linux-gnu/libdl.so;/usr/lib/x86_64-linux-gnu/lib
m.so (found version "1.10.4") found components: C
-- Could NOT find Hypre (missing: HYPRE_LIBRARIES HYPRE_INCLUDE_DIRS)
-- Checking Fortran implicit none flag : -implicitnone
-- Checking Fortran implicit none flag : -fimplicit-none
-- Configuring done
-- Generating done
```

```
sp2943@sp2943-VirtualBox:~/PFTree/parflow/build$ make
Scanning dependencies of target pfclm
[  0%] Building Fortran object pfsimulator/clm/CMakeFiles/pfclm.dir/precision.F
90.o
[  1%] Building Fortran object pfsimulator/clm/CMakeFiles/pfclm.dir/clm_varpar.
F90.o
[  1%] Building Fortran object pfsimulator/clm/CMakeFiles/pfclm.dir/clm_varcon.
F90.o
[  1%] Building Fortran object pfsimulator/clm/CMakeFiles/pfclm.dir/clmtype.F90
.o
[  1%] Building Fortran object pfsimulator/clm/CMakeFiles/pfclm.dir/drv_gridmod
ule.F90.o
[  1%] Building Fortran object pfsimulator/clm/CMakeFiles/pfclm.dir/drv_module.
F90.o
[  2%] Building Fortran object pfsimulator/clm/CMakeFiles/pfclm.dir/drv_tilemod
ule.F90.o
[  2%] Building Fortran object pfsimulator/clm/CMakeFiles/pfclm.dir/infnan.F90.
o
[  3%] Building Fortran object pfsimulator/clm/CMakeFiles/pfclm.dir/parflow_con
fig.F90.o
[  3%] Building Fortran object pfsimulator/clm/CMakeFiles/pfclm.dir/clm_typini.
F90.o
[  3%] Building Fortran object pfsimulator/clm/CMakeFiles/pfclm.dir/pf_readout.
```

```
[100%] Linking CXX executable pfmaskdownsize
[100%] Built target pfmaskdownsize
Scanning dependencies of target GeneratePythonKeys
[100%] Generate ParFlow keys for Python
-------------------------------------------------------------
-
Generate Parflow database module
-------------------------------------------------------------
-
Created 180 classes
 => No class name duplication found
Defined 545 fields were found
-------------------------------------------------------------
-
[100%] Built target GeneratePythonKeys
Scanning dependencies of target pf-python
[100%] Copying Python pftools to the build directory
[100%] Built target pf-python
sp2943@sp2943-VirtualBox:~/PFTree/parflow/build$ make install
[ 19%] Built target pfclm
[ 29%] Built target amps
[ 30%] Built target test15
[ 30%] Built target test16
[ 30%] Built target test14
[ 30%] Built target test1
[ 31%] Built target test4
```

```
sp2943@sp2943-VirtualBox:~/PFTree/parflow/build$ sudo make install
[sudo] password for sp2943:
[ 19%] Built target pfclm
[ 29%] Built target amps
[ 30%] Built target test15
[ 30%] Built target test16
[ 30%] Built target test14
[ 30%] Built target test1
[ 31%] Built target test4
[ 32%] Built target test17
[ 32%] Built target test3
[ 32%] Built target test12
[ 33%] Built target test2
[ 33%] Built target test7
[ 34%] Built target test8
[ 35%] Built target test13
[ 35%] Built target test5
[ 36%] Built target test9
[ 37%] Built target test10
[ 38%] Built target test6
[ 39%] Built target pfkinsol
[ 40%] Built target cjson
[ 84%] Built target pfsimulator
[ 84%] Built target parflow
[ 85%] Built target pfwell_cat
[ 86%] Built target pfwell_data
```

```
sp2943@sp2943-VirtualBox:~/PFTree/parflow/test/tcl$ tclsh default_single.tcl 1
1 1

default_single : PASSED
```

Washita Test Cases:

```
sp2943@sp2943-VirtualBox:~/PFTree/parflow/test/tcl/washita/tcl_scripts$ tclsh L
W_Test.tcl 1 1 1
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)


ParFlow run Complete
default_single : PASSED
```

```
sp2943@sp2943-VirtualBox:~/PFTree/parflow/test/tcl/washita/tcl_scripts$ tclsh L
W_Timing.tcl 1 1 1
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
```

```
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)
Using process grid (1,1,1)


ParFlow run Complete
default_single : PASSED
```

CMake:

CMake stands for "Cross-Platform Make" and it is an open-source build system and configuration tool that facilitates the process of building software across different platforms and operating systems. CMake provides a platform-independent way to specify the build process for a project and generate native build files (such as makefiles or project files) for various build systems (e.g., Make, Ninja, Visual Studio) and IDEs (Integrated Development Environments).

CMake uses a scripting language known as the CMake Language (CMakeLists.txt files) to define the project's build configuration. These files contain a set of commands that specify the project's source files, dependencies, compiler options, and other build-related settings.

One of the key advantages of CMake is its ability to generate build files for multiple platforms and build systems from a single set of CMakeLists.txt files. This means that developers can write the build configuration once and then use CMake to generate the appropriate build files for different platforms, making it easier to maintain and distribute software across diverse environments.

The CMake configuration process involves running CMake on the project's source code directory, which scans the CMakeLists.txt files and generates the necessary build files according to the specified configuration options. Developers can specify various build options such as compiler flags, library paths, and feature toggles using CMake variables and commands.

CMake supports a wide range of platforms, compilers, and tools, allowing developers to work with different development environments while maintaining a consistent build process. It also integrates well with other build systems, enabling seamless integration with existing projects.

Overall, CMake simplifies the build process by providing a unified and efficient way to configure and generate build files, making it easier to compile, test, and package software across multiple platforms and environments.

Tools and libraries include:

- C
- Tcl
- Python
- Fortran
- C++
- CMake
- Linux
- PFTools
- Numpy
- Matplotlib

Setting up the environment:

Before installing ParFlow on Ubuntu, we need to ensure that all the necessary programs and libraries are set up. By default, Ubuntu may not have all the required components, so we'll go through the process of installing them. Firstly, we need to install a Fortran compiler. This can be done easily by running the command "sudo apt install gfortran." To verify that the compiler is installed correctly, we can use the command "gfortran -v" and check for the displayed specifications.

Next, we need to install a text editor, git command interface, and the TCL scripting language with developer extensions. We can install these components by running the following commands: "sudo apt install vim," "sudo apt install git," "sudo apt install tcl," and "sudo apt install tcl8.6-dev." These commands will install the required tools, and if any issues arise during the installation, they will need to be resolved before proceeding further.

In addition to the program installations, we also need to define some environment variables to facilitate the ParFlow installation. These variables will serve as shortcuts for specific directories.

CMake Setup:

In order to build the ParFlow package and its supporting packages, we need to use CMake. CMake is a cross-platform build system generator that helps configure and build software projects. To set up CMake, we can follow these steps:

1. Create a directory called "PFTree" in the user's home directory. This directory will serve as the home for ParFlow and its associated components: **cd ~ mkdir PFTree cd PFTree**

2. Download the CMake installation package from the CMake downloads page. You can choose the latest build version. For example, we can use version 3.19.4:
wget https://github.com/Kitware/CMake/releases/download/v3.19.4/cmake-3.19.4.tar.gz

3. Extract the compressed tar file using the gunzip and tar commands: **gunzip cmake-3.19.4.tar.gz tar -xvf cmake-3.19.4.tar**

4. Change to the extracted CMake directory: **cd cmake-3.19.4**

5. Run the bootstrap script to configure the CMake installation. The "-DCMAKE_USE_OPENSSL=OFF" flag is used to disable the use of OpenSSL, which may not be necessary for ParFlow: **./bootstrap -- -DCMAKE_USE_OPENSSL=OFF**

6. Build CMake by running the "make" command. This will compile the source code and generate the necessary files: **make**

7. Install CMake system-wide using the "sudo make install" command. This step may require administrator privileges: **sudo make install**

By following these steps, we set up CMake on the system, which allows us to configure and build ParFlow and its dependencies effectively. CMake provides a flexible and efficient way to handle the build process, making it easier to manage the compilation and installation of the ParFlow package and its associated components.

OpenMPI Installation:

In order to install OpenMPI, which is a high-performance message passing library, we will follow these steps:

1. Download the latest release of OpenMPI from the OpenMPI downloads page. **cd ~/PFTree wget** https://download.open-mpi.org/release/open-mpi/v4.1/openmpi-4.1.0.tar.gz

2. Extract the downloaded tar file using the gunzip and tar commands: **gunzip openmpi-4.1.0.tar.gz tar -xvf openmpi-4.1.0.tar**

3. Change to the extracted OpenMPI directory: **cd openmpi-4.1.0**

4. Run the configure script to set up the installation parameters. The "--prefix=$MPI_DIR" flag specifies the installation directory for OpenMPI: **./configure --prefix=$MPI_DIR**

5. Build OpenMPI by running the "make" command. This will compile the source code and generate the necessary binaries and libraries: **make**

6. Install OpenMPI using the "make install" command. This step will copy the compiled files to the specified installation directory: **make install**

The installation process may take some time to complete. Once finished, you can verify the installation by running the "ompi_info" command, which displays detailed information about the installed OpenMPI version.

It's important to note that we are intentionally installing OpenMPI in a non-standard location, specified by the $MPI_DIR environment variable. This allows for easier updating or switching of versions in the future and avoids potential conflicts with other versions installed in the default location (/usr/local). By setting the $MPI_DIR and $MPI_RUN_DIR environment variables, we ensure that ParFlow can locate and utilize the correct OpenMPI installation and associated binaries.

Installing Hypre Library:

Hypre is a library of solvers that ParFlow utilizes for solving linear systems. It provides efficient and scalable algorithms for solving large-scale scientific and engineering problems. To install the Hypre library, we can follow these steps:

1. Navigate to the directory where we want to install the library. In this case, we'll go to the "PFTree" directory: **cd ~/PFTree**

2. Clone the Hypre repository from GitHub using the git command. This will create a local copy of the repository on our machine: **git clone https://github.com/hypre-space/hypre.git --branch master --single-branch**

3. Change to the "src" directory within the cloned Hypre repository: **cd hypre/src**

4. Configure the installation using the "./configure" command. We specify the prefix where we want to install Hypre using the "--prefix" option. In this case, we set it to the previously defined $HYPRE_DIR environment variable: **./configure --prefix=$HYPRE_DIR**

5. Build the Hypre library by running the "make" command. This will compile the source code and generate the necessary files: **make**

6. Install the compiled library and associated files to the specified directory using the "make install" command: **make install**

If the MPI version of the compiler is not automatically detected, you can modify the "configure" command to specify the path to the MPI compiler. For example, you can use the option "CC=$MPI_RUN_DIR/mpicc" to set the C compiler to the MPI version. However, in most cases, the default configuration should work fine.
By following these steps, we clone the Hypre repository, configure the installation, compile the source code, and install the Hypre library. Once installed, ParFlow can utilize the capabilities of Hypre for efficient and scalable solving of linear systems within the ParFlow simulation.

Installing Silo Library:

The Silo library is used by ParFlow as a database to manage and store simulation output data. To install Silo, we can follow these steps:

1. Navigate to the directory where we want to install Silo. In this case, we'll go to the "PFTree" directory: **cd ~/PFTree**

2. Download the Silo tarball from the official website. Visit the SILO download page and click on the Downloads tab to find the latest version. For example, we can use version 4.10.2:
wget https://wci.llnl.gov/sites/wci/files/2021-01/silo-4.10.2.tgz

3. Extract the tarball using the gunzip and tar commands: **gunzip silo-4.10.2.tgz tar -xvf silo-4.10.2.tar**

4. Rename the extracted directory to "silo" (optional). This step is not strictly necessary, but it's important that the directory name matches the $SILO_DIR environment variable. If you choose not to rename it, make sure to update the $SILO_DIR accordingly: **mv silo-4.10.2 silo**

5. Change to the "silo" directory: **cd silo**

6. Configure the installation using the "./configure" command. Here, we disable the "silex" component, which is not needed for ParFlow: **./configure --disable-silex**

7. Build the Silo library by running the "make" command. This will compile the source code and generate the necessary files: **make**

8. Install the compiled library and associated files using the "make install" command: **make install**

By following these steps, we download the Silo tarball, extract the files, configure the installation, compile the source code, and install the Silo library. With Silo installed, ParFlow can utilize its capabilities as a database to manage and store simulation output data efficiently.

Steps Involved:

Installation of Parflow and associated libraries –

Step 1: Installation Setup

In the setup phase of ParFlow, we need to choose a suitable location on our system to install ParFlow and its associated libraries. This location should align with our preferences and any specific requirements or conventions we have. This chosen directory will serve as the foundation for the ParFlow installation.

Next, we must set up the environment variable PARFLOW_DIR to point to the selected installation directory. By doing this, we allow ParFlow to easily locate its installation files and access the necessary libraries. Setting environment variables differs depending on the shell we are using. In the case of bash, we can use the export command to set the variable. For instance, if we decide to install ParFlow in the directory /sp2943/parflow, we would set the PARFLOW_DIR environment variable accordingly.

By configuring the PARFLOW_DIR environment variable, we provide ParFlow with the essential information it needs to locate its files and dependencies throughout the installation and execution processes. This ensures that ParFlow functions properly and can access all the required resources within the designated installation directory.

Step 2: Extracting the Source

To initiate the installation process, we need to obtain the ParFlow release from the GitHub website and extract the compressed tar file. We locate the specific/latest version of ParFlow, such as version 3.12.0, and download the corresponding tar file, such as "parflow-3.12.0.tar.gz," to our local machine.

Next, we create a new directory in our home directory where we will extract the ParFlow source files. By using the command mkdir ~/parflow, we create a directory named "parflow" in our home directory. We then navigate to this directory using the command cd ~/parflow.

Now, we can extract the contents of the tar file. Using the command tar -xvf followed by the path to the downloaded tar file, such as tar -xvf ../parflow-3.12.0.tar.gz, we extract the files.

The extraction process unpacks all the necessary source files and directories needed to build and install ParFlow into our current directory, which is the "parflow" directory we created earlier.

By completing this step, we have successfully obtained the ParFlow source code and extracted it into our designated directory. This lays the foundation for further configuration and building of ParFlow in the subsequent steps.

Step 3: Running CMake to configure ParFlow

In this step, we will use CMake to configure ParFlow and generate the necessary makefiles for building the software. CMake provides us with the flexibility to set compiler options and other configuration settings.

To begin, let's create a separate build directory outside the source directory to keep our build process organized. We can do this by running the command mkdir build and then navigating to the build directory using cd build.

Once inside the build directory, we have two options for configuring ParFlow using CMake: we can either use the ccmake GUI or the cmake command line.

Option 1: Building with the ccmake GUI

We can launch the ccmake GUI interface by running the command ccmake ../parflow. This will open an interactive interface where we can configure ParFlow's build options.

In the GUI interface, users have the flexibility to customize configuration options based on their specific requirements. One crucial configuration setting is the installation prefix (CMAKE_INSTALL_PREFIX), which should align with the pre-defined PARFLOW_DIR to ensure ParFlow is installed in the desired location.

After configuring the desired options, users can proceed by pressing the 'c' key to initiate the project configuration. Depending on the specific environment and requirements, additional changes or options can be set at this stage. To generate the final configuration and exit the GUI, users should press the 'g' key. This step ensures that ParFlow is appropriately configured and ready for the subsequent stages of the build process.

Option 2: Building with the cmake command line

Alternatively, we can configure ParFlow using the cmake command directly from the command line.

By running the cmake command followed by the path to the ParFlow source directory (e.g., ../parflow), we can specify various options using the -D flag. For example, -DCMAKE_INSTALL_PREFIX=${PARFLOW_DIR} sets the installation prefix to our previously defined PARFLOW_DIR, and -DPARFLOW_HAVE_CLM=ON enables the CLM (Community Land Model) integration.

We can adjust additional options and flags based on our specific requirements. We should refer to the documentation for more information on the available options.

By completing this step, we will have successfully used CMake to configure ParFlow, either through the ccmake GUI or the cmake command line. This configuration step prepares the software for the subsequent build and installation stages.

Step 4: Building and installing

After CMake has generated the necessary makefiles, we can proceed with building and installing ParFlow:

In the build directory, we run the make command to initiate the build process. This command compiles the source code and generates the ParFlow executable.

Once the build process is completed, we can proceed to install ParFlow by running the make install command. This command copies the necessary files and libraries to the specified installation directory.

By executing these two steps, we successfully build and install ParFlow, making it ready for use in our desired location.

Step 5: Running a sample problem

Assuming the successful installation of ParFlow, we can run a sample problem to ensure its proper functionality. To do this:

We navigate to the ParFlow test directory using the command cd parflow/test. This directory contains sample problem files for testing purposes.

Using the TCL shell (tclsh), we run the sample problem script. For example, we can execute the following command: tclsh default_single.tcl 1 1 1. This command initiates the execution of the sample problem with the specified parameters.

It is important to ensure that the environment variable PARFLOW_DIR is set correctly, pointing to the installation directory of ParFlow. Additionally, we need to ensure that the appropriate TCL shell is included in the system's PATH variable.

Note that on certain parallel machines, it might be necessary to execute the sample problem command within a batch file or start a parallel interactive session, depending on the system's requirements.

By following these steps, we can verify the functionality of ParFlow by running a sample problem, confirming that the installation and configuration processes were successful.

Executing ParFlow model: Washita

In this code snippet, we utilize the Python programming language along with the NumPy and Matplotlib libraries to execute the ParFlow model on the Washita dataset and generate a visual representation of the model outputs. The first step is to import the necessary libraries. We then specify the path to the downloaded data file, 'drv_vegm.alluv.dat', which contains the output data from the model. Next, we read the file line by line and preprocess the data. We iterate over each line, splitting it into individual elements based on whitespace. We only consider lines that have the expected number of elements (25) and store them in the 'lines' list. To facilitate further processing, we convert the 'lines' list into a NumPy array named 'data', with the elements having a float data type.

Moving forward, we extract the relevant columns from the 'data' array. The 'lon' variable contains the longitude values from the 4th column (index 3), the 'lat' variable contains the latitude values from the 3rd column (index 2), and the 'color' variable holds the color values from the 7th column (index 6) of the 'data' array.

With the data prepared, we proceed to plot the model outputs. Using the Matplotlib library, we create a scatter plot by calling the 'plt.scatter' function. This function takes the longitude and latitude values as the x and y coordinates, respectively, and assigns colors based on the 'color' variable using the 'c' and 'cmap' parameters. We then add axis labels to indicate longitude and latitude using 'plt.xlabel' and 'plt.ylabel' functions, set the plot's title with 'plt.title', and include a colorbar to represent the color scale using 'plt.colorbar'. Finally, we display the plot on the screen using 'plt.show'. By executing this code, we can visualize the model outputs and observe the spatial distribution of the data points on the scatter plot, providing valuable insights into the Washita dataset.

Total Evaporation for 4 time steps:

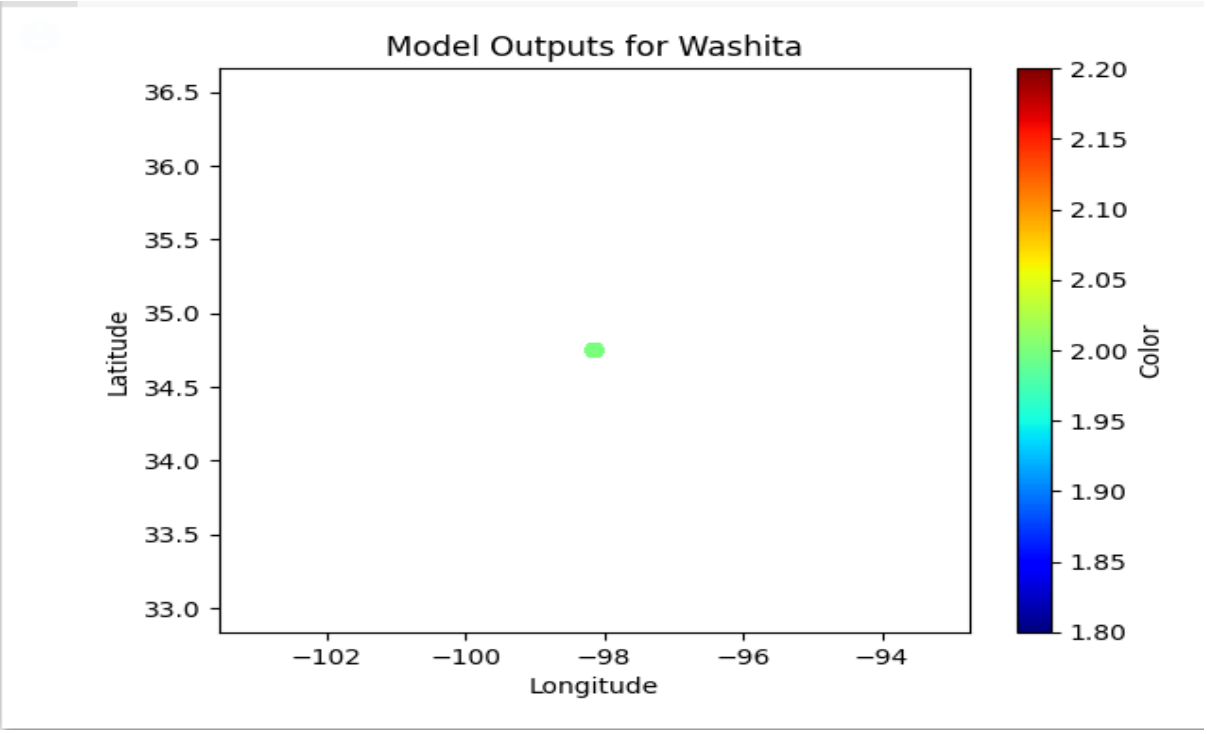
```
print(data_clm)
```

```
[[1.914223e-05 2.870696e+02 2.870696e+02]
 [1.914223e-05 2.870696e+02 2.870696e+02]
 [1.914223e-05 2.870696e+02 2.870696e+02]
 [1.914223e-05 2.870696e+02 2.870696e+02]]
```

```
Convert PFB to TXT
File: 1
sfo_example.out.clm_output.00001.C.pfb
File: 2
sfo_example.out.clm_output.00002.C.pfb
File: 3
sfo_example.out.clm_output.00003.C.pfb
File: 4
sfo_example.out.clm_output.00004.C.pfb
...Done with year sfo_example pressure file conversions
```

```
Total Evaporation for 4 time steps:
00001 0.000019
00002 0.000016
00003 0.000011
00004 0.000012
```

Model Outputs for Washita:



Splitting the data into training and testing sets:

In this code snippet, we utilize the NumPy library to handle numerical operations and the scikit-learn library to split a dataset into training and testing sets. The 'X' variable represents the features, which are extracted from the 'data' array by selecting all columns except the last one. Similarly, the 'y' variable represents the target variable, obtained by selecting only the last column of the 'data' array.

To split the data, we employ the 'train_test_split' function from scikit-learn. By passing in the 'X' and 'y' variables, along with a 'test_size' of 0.2 (indicating a 20% split), we generate four separate arrays: 'X_train', 'X_test', 'y_train', and 'y_test'. These arrays contain the features and target variable for the training and testing sets, respectively.

The resulting split allows us to train a machine learning model on the 'X_train' and 'y_train' data, and then evaluate its performance on the unseen 'X_test' data. This separation is crucial for estimating how well the model generalizes to new, unseen instances, helping us assess its effectiveness in real-world scenarios.

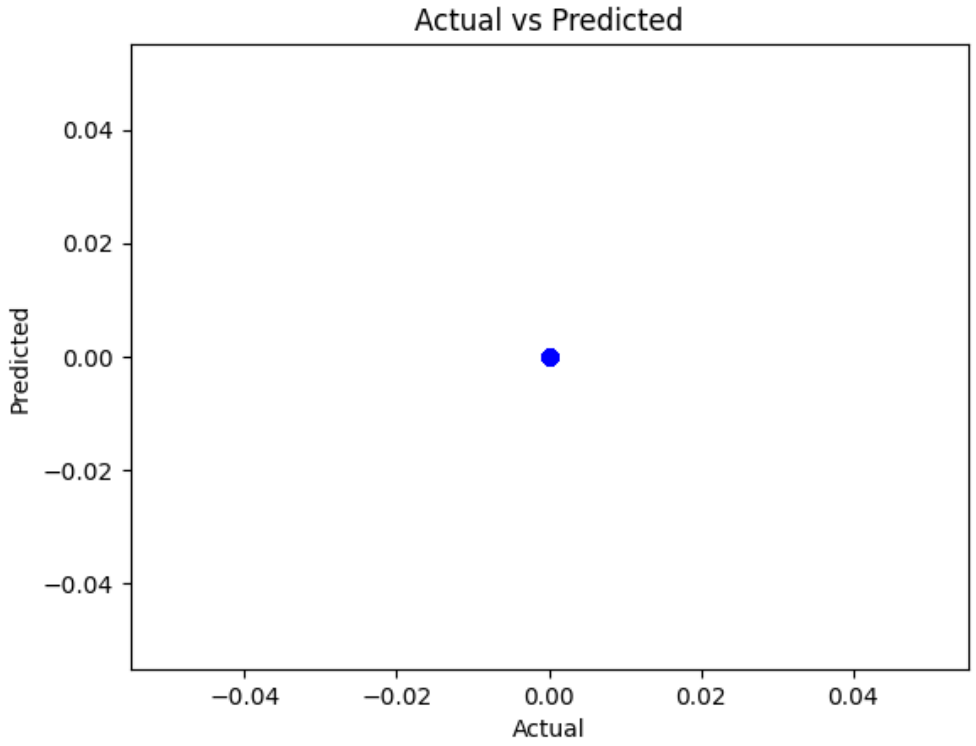
In this code, we utilize the matplotlib library for visualization and scikit-learn to perform linear regression modeling and evaluate its performance. After importing the necessary libraries, we assume that we have already defined the training and testing data as 'X_train', 'y_train', 'X_test', and 'y_test'.

To train the linear regression model, we create an instance of the 'LinearRegression' class and fit it to the training data using the 'fit' method. Once the model is trained, we make predictions on the test data by calling the 'predict' method and passing in 'X_test'. The predicted values are stored in the 'y_pred' variable.

Next, we calculate the root mean squared error (RMSE) as a measure of the model's performance. The 'mean_squared_error' function from scikit-learn is used, with the 'squared' parameter set to 'False' to obtain the RMSE directly. The calculated RMSE is stored in the 'rmse' variable.

Finally, we visualize the actual values against the predicted values using a scatter plot. The 'scatter' function is used to plot the actual and predicted values, with the 'color' parameter set to 'blue'. Additionally, we draw a red dashed line to represent the ideal scenario where the actual and predicted values perfectly align. The plot is labeled, titled, and displayed using the 'xlabel', 'ylabel', 'title', and 'show' functions, respectively.

```
Root Mean Squared Error: 0.0
```



3D Scatter Plot:

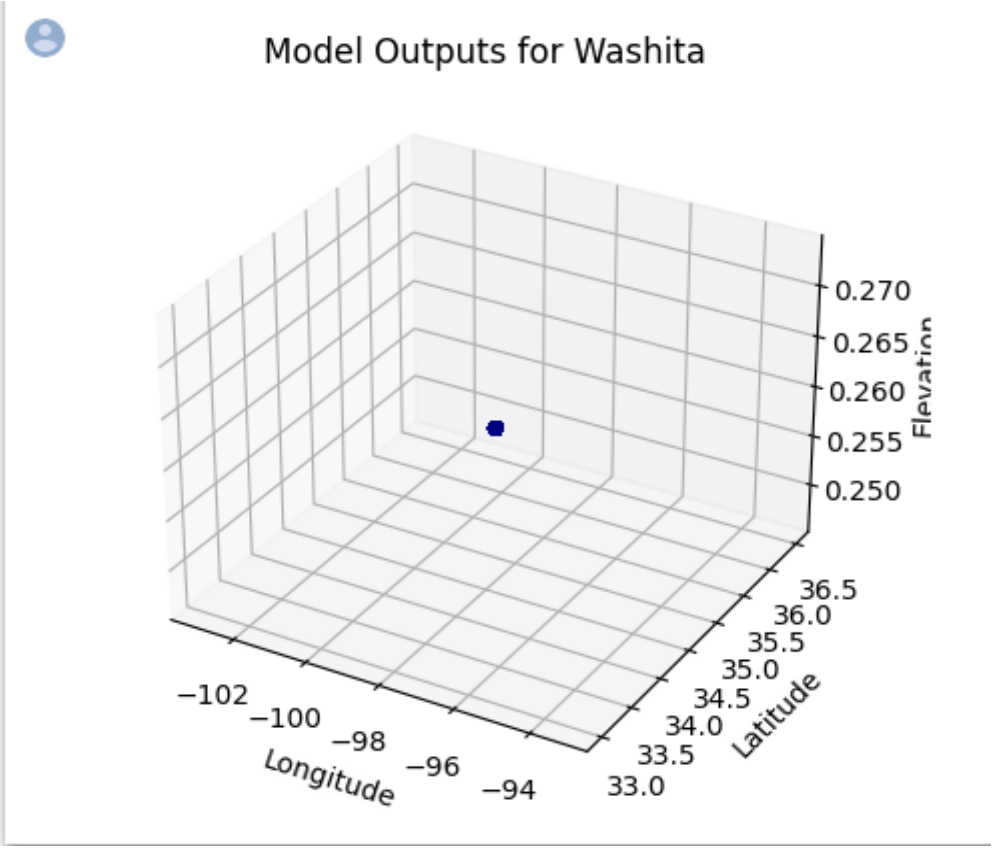
The code snippet demonstrates the creation of a 3D scatter plot using the mpl_toolkits.mplot3d library in matplotlib. Firstly, the relevant columns from the 'data' array are extracted, representing the longitude, latitude, and elevation. These columns are assigned to variables 'lon', 'lat', and 'elevation', respectively.

Next, a new figure is created using 'plt.figure()', and a subplot with 3D projection is added using 'fig.add_subplot(111, projection='3d')'. This sets up the plot as a 3D plot.

To visualize the model outputs, the scatter function is called on the subplot 'ax'. The 'lon', 'lat', and 'elevation' variables are passed as the x, y, and z coordinates, respectively. Additionally, the 'c' parameter is set to 'color' to determine the color of the points based on the 'color' variable. The 'cmap' parameter specifies the colormap to use.

Axis labels are set using the 'set_xlabel', 'set_ylabel', and 'set_zlabel' functions, specifying the labels for the longitude, latitude, and elevation axes, respectively. The plot is given a title using 'set_title'.

Finally, the plot is displayed using 'plt.show()'. The resulting plot shows the model outputs represented as points in a 3D space, with longitude on the x-axis, latitude on the y-axis, and elevation on the z-axis. The color of the points is determined by the 'color' variable, and the plot is labeled with appropriate axis labels and a title.



Advantages of the ParFlow model:

- **Robust Simulation Capabilities:** ParFlow offers advanced capabilities for simulating complex hydrological processes, including surface water, subsurface flow, and

coupled surface-subsurface interactions. It can handle a wide range of hydrological phenomena, such as infiltration, evapotranspiration, groundwater flow, and surface runoff, making it suitable for diverse applications.

- **Parallel Computing Efficiency:** ParFlow is designed for parallel computing, leveraging high-performance computing architectures to efficiently solve large-scale hydrological problems. It can effectively distribute computations across multiple processors or computing nodes, enabling faster simulations and handling larger datasets than traditional sequential models.
- **Flexible Model Configuration:** ParFlow provides flexibility in configuring and customizing hydrological simulations. It allows users to specify various boundary conditions, soil properties, and hydrologic processes, enabling detailed and tailored modeling studies. This flexibility enhances the model's versatility for addressing specific research questions and real-world hydrological scenarios.

Limitations of the ParFlow model:

- **Steep Learning Curve:** ParFlow is a complex modeling framework that requires a solid understanding of hydrological processes, numerical methods, and parallel computing concepts. Consequently, there is a steep learning curve for new users who are unfamiliar with these topics. Extensive knowledge and experience are often necessary to set up and calibrate ParFlow models effectively.
- **Computational Resource Requirements:** Due to its parallel computing nature, running ParFlow simulations on large-scale domains or with high-resolution data can demand significant computational resources. Access to high-performance computing infrastructure and computational expertise is essential for efficiently executing ParFlow simulations and obtaining timely results.
- **Calibration and Validation Challenges:** Like any hydrological model, ParFlow requires proper calibration and validation against observed data to ensure accurate and reliable simulation results. The calibration process can be challenging due to the model's complexity and the need for reliable hydrological data. Adequate data collection, preprocessing, and calibration techniques are necessary to maximize the model's performance and reliability.

Conclusion:

In conclusion, ParFlow is a powerful and versatile hydrological modeling framework that offers robust simulation capabilities and efficient parallel computing. Its ability to simulate complex hydrological processes, coupled surface-subsurface interactions, and handle large-scale problems makes it suitable for a wide range of applications. However, the use of

ParFlow requires a steep learning curve and access to computational resources, which can pose challenges for new users and limit its widespread adoption. Proper calibration and validation against observed data are crucial for ensuring accurate results. Despite its limitations, ParFlow remains a valuable tool for conducting advanced hydrological research and modeling studies. Continued advancements and enhancements in the framework, along with user support and training, can further enhance its usability and effectiveness in addressing complex hydrological challenges.