HybGe-Flow3D User Manual, Version 1.0.0

Timothy B. Costa

March 10, 2016

Contents

Chapter 1

HybGe-Flow3D Overview

HybGe-Flow3D (HGF) is a software package that solves multiscale laminar fluid flow problems in complex, uncertain geometries. This software is distributed freely AS IS and with ABSOLUTELY NO WARRANTY, and with the hope that it will be used and extended by the scientific computing community. In this document we describe how to compile and run Hybge-Flow3D.

This software was developed under the partial support of the National Science Foundation, on the project NSF-DMS 1115827 "Hybrid modeling in porous media."

1.1 License & Citation

Publications making use of HybGe-Flow3D should cite this software package. An example citation is given as:

> Costa, T., "HybGe-Flow3D", Package Version 1.0.0, http://github.com/costat/HybGe-Flow3D.

1.2 Model & Discretization

At the porescale, the basic equations solved by HGF are the Stokes equations, modified with a resistive term corresponding to an immersed boundary representation of complex geometry.

$$\begin{aligned} -\mu\nabla^2 u + \frac{1}{\eta}\chi_{\Omega_g} u = -\nabla p, \quad & x \in \Omega, \\ u = u_D, \quad & x \in \partial\Omega_D \\ \nabla u \cdot n = 0, \quad & x \in \partial\Omega_N. \end{aligned}$$

Here, $\Omega \subset \mathbb{R}^d$ with $d = 3$ or $d = 2$ is the simulation domain, and $\partial\Omega$ is its boundary. $\partial\Omega_D$ refers to the 'Dirichlet boundary' where the fluid velocity is prescribed. This is used for no-slip and inflow boundary conditions. $\partial\Omega_N$ refers to the 'Neumann boundary' where a homogeneous diffusive flux is prescribed. This is used for outflow conditions. Additionally we have $\Omega = \Omega_g \cup \Omega_f$, $\Omega_g \cap \Omega_f = \emptyset$, where Ω_g refers to the immersed boundary and Ω_f the fluid flow domain. In these equations u is the fluid velocity, p is the fluid pressure, and μ is the fluid viscosity. χ_{Ω_g} is the indicator function for the immersed boundary domain Ω_g, and η is a penalization parameter, taken to be very small. HybGe-Flow3D solves the above fluid flow model by a staggered-grid finite volume discretization.

HybGe-Flow3D has several problem scenarious built in. In the simplest case, HGF will simply solve the above model on a given geometry, and produce a constant upscaled absolute permeability based on the flow direction chosen. Additionally, HGF can be instructed to solve the flow problem in all principal axis directions and produce the full upscaled absolute permeability tensor.

More interesting are the full multiscale simulation capabilities of HGF. In later sections thsese will be examined in detail. Briefly, HGF can ben instructed to cut a geometry into subdomains and solve local porescale flow problems. On these subdomains absolute permeabilities are computed, and used in a constructed pore-network model. This pore-network model is then solved to produce a permeability for the full domain.

Additionally, HGF can be instructed to sample a stochastic resistive term on each subdomain and produce empirical probability density functions relating the permeability to the presence of growth of an obstruction in the void space. These densities are then sampled for throat permeabilities in a pore-network solve. In future work we anticipate this being useful for transient simulations at pore-network or corescale with reactive transport.

Chapter 2

Installation

HybGe-Flow3D has two variants. The first uses Paralution for linear algebra, and the second uses MAGMA. The MAGMA version is currently experimental, and thus we will not overview the build process for MAGMA in this document.

In addition to requiring either Paralution or MAGMA. HybGe-Flow3D requires CUDA and BOOST C++ Libraries. In particular, HGF requires that the system, filesystem, and serialization BOOST libraries be compiled. In this section we review the build process for Paralution, BOOST, CUDA, and HybGe-Flow3D on a linux machine. We make no promises that these steps work universally, but they have worked reliably for the author on a Debian based or Fedora machine.

2.1 Building BOOST

Many BOOST libraries do not require separate compilation, as they are header-only libraries. In our case, though, we require the following compiled libraries:

- serialization
- filesystem
- system

We will describe here how to build these 3 libraries and provide a template module file. These instructions can be found at http://www.boost.org/doc/libs/1_60_0/more/getting_started/unix-variants.html.

First, download the boost_1_60_0.tar.bz2 from above site. We then open the tar with

```
$ tar –bzip2 -xf /path/to/boost_1_60_0.tar.bz2
```

Then navigate into the unpacked directory

```
$ cd /path/to/boost_1_60_0
```

Then create a build directory where the libraries and header files will be installed. We assume this is located at /path/to/boostbuild. We then issue the following command from within the boost_1_60_0 directory, setting prefix to our build directory

```
$ ./bootstrap.sh –prefix=/path/to/boostbuild –with-libraries=serialization,system,filesystem
```

Then

```
$ ./b2 install
```

This installs the two directories /path/to/boostbuild/lib and /path/to/boostbuild/include. The –with-libraries flag is optional. Without it boost will build all compiled libraries. However, this takes some time, and HGF uses only the three compiled libraries listed above.

Next we write an environment module file for BOOST. We assume for exposition that this file is placed at /path/to/modulefiles/boost/1.60.0, and describe the process of adding the folder /path/to/modulefiles to the module path using a bash terminal.

```
#%Module 1.0
#
# Boost module
#

module-whatis "boost/1.60.0"

set boost_home /path/to/boostbuild/

prepend-path          PATH                  $boost_home

prepend-path          LIBRARY_PATH          $boost_home/lib
prepend-path          LD_LIBRARY_PATH       $boost_home/lib
prepend-path          CMAKE_LIBRARY_PATH    $boost_home/lib

prepend-path          INCLUDE_PATH          $boost_home/include
prepend-path          C_INCLUDE_PATH        $boost_home/include
prepend-path          CPLUS_INCLUDE_PATH    $boost_home/include
prepend-path          CMAKE_INCLUDE_PATH    $boost_home/include

setenv                BOOST_HOME            $boost_home
setenv                BOOST_DIR             $boost_home
```

To see the modulefiles within /path/to/modulefiles we add the following line to our .bashrc:

```
module use-append /path/to/modulefiles
```

Open a new terimal or type

```
$ source ~/.bashrc
```

Finally, type

```
$ module avail
```

and verify that boost/1.60.0 is an available module.

2.2 Building CUDA

Instructions for building CUDA can be found at http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux. We prefer not to include a link to /usr/bin/, opting instead for a cuda modulefile. Assuming the version of CUDA built is 7.5.18, the following is placed at /path/to/modulefiles/cuda/7.5.

```
#%Module

proc ModulesHelp { } {
  puts stderr "CUDA 7.5.18"
  puts stderr "TAGS: compilers mpi"
}

module-whatis "cuda/7.5.18"

conflict cuda

set cuda_home /path/to/cudabuild/

prepend-path PATH $cuda_home/bin

prepend-path LD_LIBRARY_PATH $cuda_home/lib
prepend-path LD_LIBRARY_PATH $cuda_home/lib64

prepend-path LIBRARY_PATH $cuda_home/lib
prepend-path LIBRARY_PATH $cuda_home/lib64

prepend-path INCLUDE_PATH $cuda_home/include
prepend-path C_INCLUDE_PATH $cuda_home/include

setenv CUDA_HOME  $cuda_home
setenv CUDA_DIR   $cuda_home
setenv CUDADIR    $cuda_home
```

2.3 Building Paralution

Paralution is straightforward to build on Linux machines using cmake. The instructions contained here can be found at
http://www.paralution.com/download/.

First, navigate to the directory in which you will build paralution, which we will denote by /path/to/paralution. We create a build directory for Paralution,

```
$ mkdir ./1.1.0
```

Then grab the tar file containing Paralution.

```
$ wget http://www.paralution.com/downloads/paralution-1.1.0.tar.gz
```

The following steps build paralution using cmake.

```
$ tar zxvf paralution-1.1.0.tar.gz
$ cd 1.1.0
$ cmake ../paralution-1.1.0/
```

```
$ make
```

Next we write an environment module file for Paralution. This file should be located at /path/to/modulefiles/paralution/1.1.0.

```
#%Module 1.0
#
#  Paralution module
#

module-whatis "paralution/1.1.0"

set paralution_home /path/to/paralution/1.1.0
prepend-path    PATH                $paralution_home/bin

prepend-path    LIBRARY_PATH        $paralution_home/lib
prepend-path    LD_LIBRARY_PATH     $paralution_home/lib
prepend-path    CMAKE_LIBRARY_PATH  $paralution_home/lib

prepend-path    INCLUDE_PATH        $paralution_home/inc
prepend-path    C_INCLUDE_PATH      $paralution_home/inc
prepend-path    CPLUS_INCLUDE_PATH  $paralution_home/inc
prepend-path    CMAKE_INCLUDE_PATH  $paralution_home/inc

setenv          PARALUTION_HOME     $paralution_home
setenv          PARALUTION_DIR      $paralution_home
```

2.4 Building HybGe-Flow3D

Once the dependencies are built and module files are in place, building HybGe-Flow3D is simple. First, clone (or download from www.github.com/costat/HybGe-Flow3D/) the repository.

```
$ cd /path/to/build/hgf
$ git clone git@github.com:costat/HybGe-Flow3D
```

Then navigate into the PARALUTION subfolder of the repository.

```
$ cd ./HybGe-Flow3D/PARALUTION
```

Next, load the appropriate modules,

```
$ module load paralution/1.1.0
$ module load cuda/7.5.18
$ module load boost/1.60.0
```

Next, execute the cmake script and make,

```
$ cmake .
$ make
```

And, done! The binary 'hgf' is now located in the PARALUTION folder.

Chapter 3

Running HybGe-Flow3D

To run HGF, the binary must be pointed at a directory containing, at minimum, two files: a geometry file "Geometry.dat," and a parameter file "Paramters.dat." The format of these files is most easily explained by example. HGF ships with 3 examples located at /path/to/hgf/examples/. After running a simulation with a full porescale solve a file "Mesh.dat" is created and placed in the directory. If the file "Mesh.dat" is present, then for global porescale solves the "Geometry.dat" file is ignored and the "Mesh.dat" file is loaded. All output is placed in a directory /path/to/problemdirectory/Output/. If this directory does not exist hgf creates the directory.

3.1 Geometry.dat

Opening ../examples/3dCube/Geometry.dat, we see

```
nx= 30
ny= 30
nz= 30
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
.
.
.
```

The first three lines of the Geometry.dat file tell HGF the number of voxels in the x, y, and z direction in the geometry data file. The formatting is important here. HGF reads the first integer found after a space in line 1 as the value of nx, likewise for line 2 with ny, and line 3 with nz. Thus we must have nx= with no spaces, followed by a space, followed by the integer value of nx. Similarly for ny and nz. Each voxel must be a '0', '1', or '2'. In this trivial example, all voxels are '0,' which refers to void space in the domain. A value of '1' identifies the solid matrix, which will not be meshed. Finally, a value '2' identifies part of the immersed boundary. This voxel will be in the mesh, but flow will be restricted by the immersed boundary term. The voxel array is formatted by the following rules:

- A row of voxels in Geometry.dat corresponds to a line of voxels in the x-direction in the domain, and is nx entries long.
- Each block of voxels corresponds to a z-slice, i.e. an x-y plane in the geometry, and has size nx×ny.
- Each z-slice is separated by an empty row.
- The first line of voxels must be in line 4 of the file. There is no vertical space between nz and the beginning of the voxel array.

3.2 Parameters.dat

Opening ../examples/3dCube/Geometry.dat, we see

```
length= 1.0
width= 1.0
height= 1.0
visc= 1.0
direction= 0
output= 1
nThreads= 4
prec= 0
tolAbs= 1e-8
tolRel= 1e-8
maxIt= 3000
nCuts= 4
```

This file contains the problem and solver parameters needed to run HGF. Similarly to the formatting for nx, ny, and nz in the Geometry.dat file, it is important that each entry contains "valuelabel=" followed by a space, followed by the value (double or int, case depending). Additionally, these values are read in by their order, not by their label. So, entries must be provided in the order:

1. length ← the length of the domain, i.e. size in the x-direction.
2. width ← the width of the domain, i.e. the size in the y-direction.
3. height ← the height of the domain, i.e. the size in the z-direction.
4. visc ← the fluid viscosity.
5. direction ← this is an integer which determines the type of problem HGF will solve. The possible values are identified below.

6. output ← this is an integer that determines the format of the file containing flow solutions. 0 for tecplot formatting, and 1 for paraview vtk formatting.

7. nThreads ← this is an integer that tells Paralution how many shared memory threads to use. If this is greater than the machine's maximum threads, Paralution will use all threads available. Must still be an entry in the Parameters.dat file even if MAGMA is used, since ordering determines the read in of the parameters.

8. prec ← this is an integer that tells Paralution how many p-levels to use in the ILU preconditioner. This may be relevant to MAGMA later, when preconditioning is sorted out and the MAGMA version of HGF is no longer "experimental."

9. tolAbs ← the absolute tolerance for the linear solver residual.

10. tolRel ← the relative tolderance for the linear solver residual.

11. maxIt ← the maximum number of iterations to be used by the linear solver.

12. nCuts ← the number of cuts to make in each axis direction for the definition of subdomains in the subdivide multiscale strategies.

The direction parameter requires further discussion. If direction$\in \{0, 1, 2\}$, then HGF will solve a problem in a single principal axis flow direction, and compute the absolute permeability in that direction from the solution. For these cases, direction 0 corresponds to an x-flow problem, 1 to a y-flow problem, and 2 to a z-flow problem. If direction = 3, then all flow directions are solved and a permeability tensor is computed. If direction = -1, hgf simply meshes the geometry and saves the mesh to /problemdirectory/Mesh.dat.

Directions 4 and 5 define scenarios which are experimental, and for research purposes related to upcoming papers. If direction = 4 the domain is cut nCuts times in each axis direction to define subdomains. On each subdomain, permeabilities are computed from porescale simulations in each axis direction. These values are then used to define a pore-network model, and a global permeability in the x-direction is computed from the solution to the pore-network model. If direction = 5, the domain is subdivided similarly to the direction = 4 case. This time, though, hgf randomly samples the immersed boundary vector to compute distributions of permeabilities for each subdomain based on the mass fraction of immersed boundary. This distributions are then sampled to define a pore-network model and a global permeability in the x-direction is computed.

Chapter 4

Examples

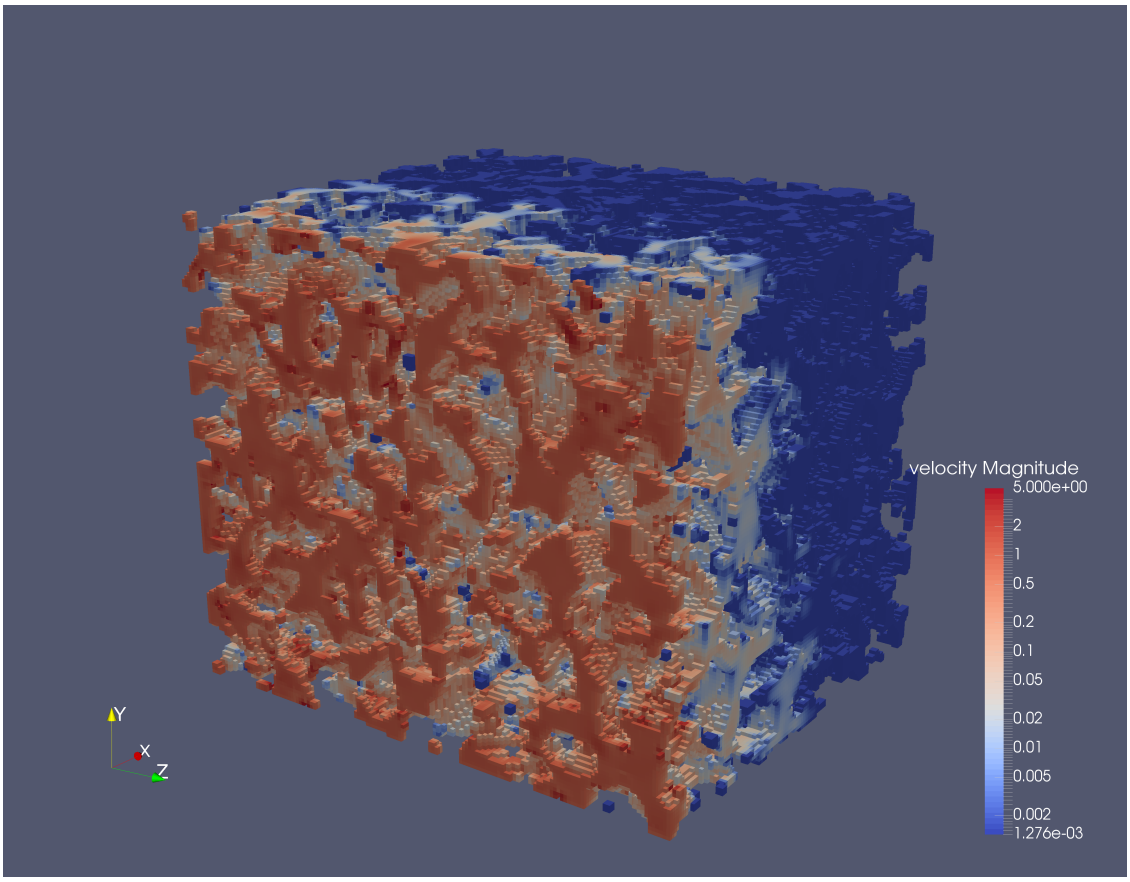

Figure 4.1: Velocity magnitude is sandstone geometry.

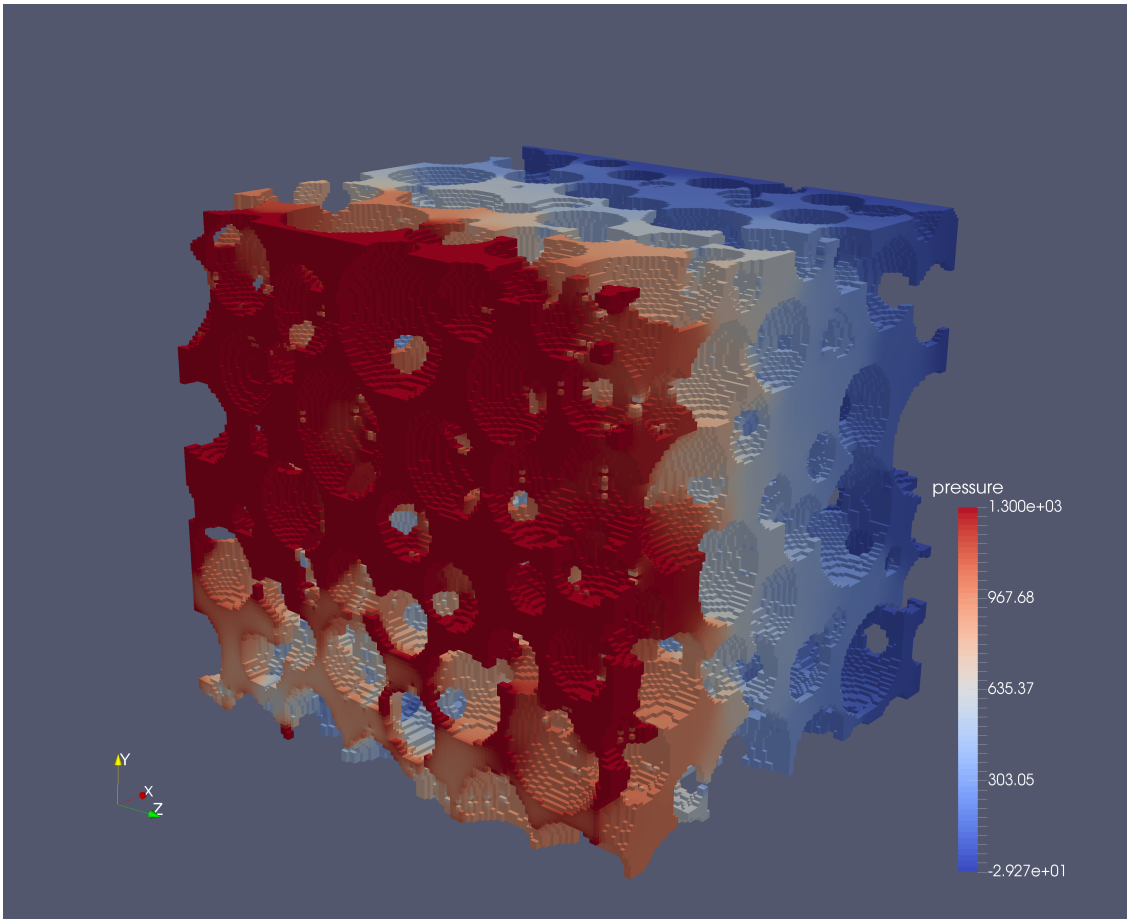

Figure 4.2: Pressure in glass bead geometry.