Usage of Dune at SINTEF Applied Mathematics

Atgeirr Flø Rasmussen

SINTEF ICT, Dept. Applied Mathematics

Dune User Meeting 2010

Introduction

The `dune-cornerpoint` module

The `dune-porsol` module

The `dune-upscaling` module

Future work: OPM

Introduction

The `dune-cornerpoint` module

The `dune-porsol` module

The `dune-upscaling` module

Future work: OPM

SINTEF

SINTEF is the largest independent research organisation in Scandinavia

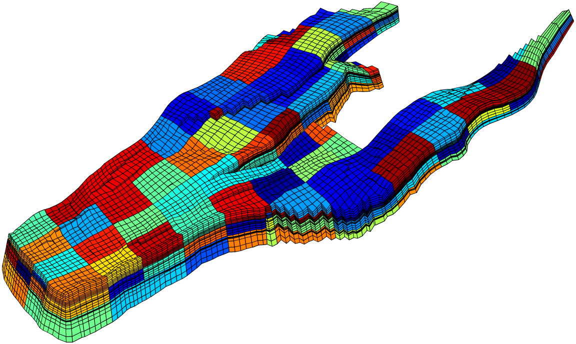
Approx. 2000 employees.

Engaged in research in many areas:

- Information/Communication Technology
- Energy
- Materials science and Chemistry
- Petroleum research
- Fisheries and Aquaculture
- Technology and society
- ... and more.

Simulation group activities

10 people, led by Knut-Andreas Lie. Located in Oslo.



Some current activities:

- Open Porous Media (developing free simulators).
- Matlab Reservoir Simulation Toolbox (also free).
- Multiscale methods.
- Streamline methods.
- Non-uniform coarsening methods for transport eqn.

In this talk...

... I will discuss

- Corner point grids, and the `dune-cornerpoint` module.
- Two-phase flow, and the `dune-porsol` module.
- Upscaling, and the `dune-upscaling` module.
- Ongoing and future work.

Introduction

The `dune-cornerpoint` module

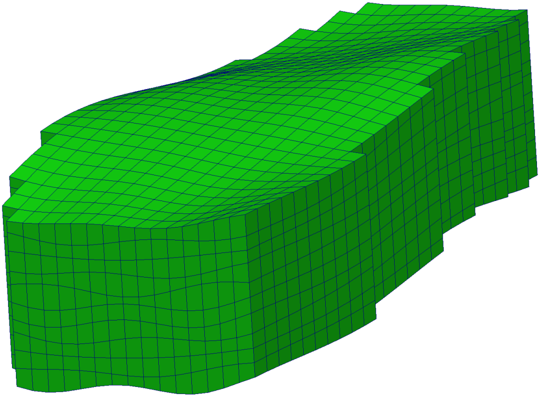
The `dune-porsol` module

The `dune-upscaling` module

Future work: OPM

Corner-point grids

Introduced by Ponting in 1989.

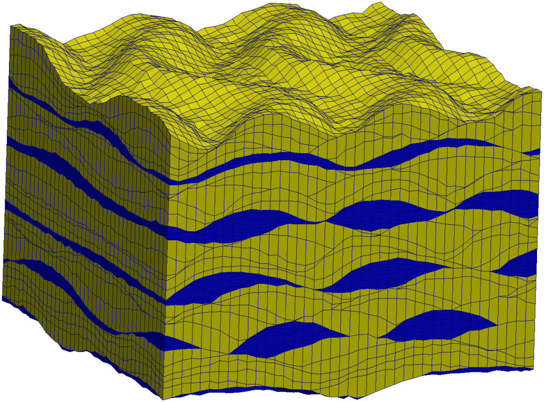


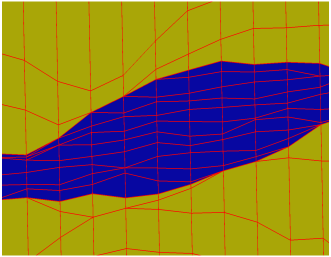
- Corners lie on pillars.
- Logical cartesian structure.
- Each cell given by eight corner locations.
- Grid layers correspond to geological layers.
- Cells can be non-convex.
- Faces are generally curved.

Grids with degeneracies

In a geological setting, erosion may be significant.

We get degenerate cells, faces.

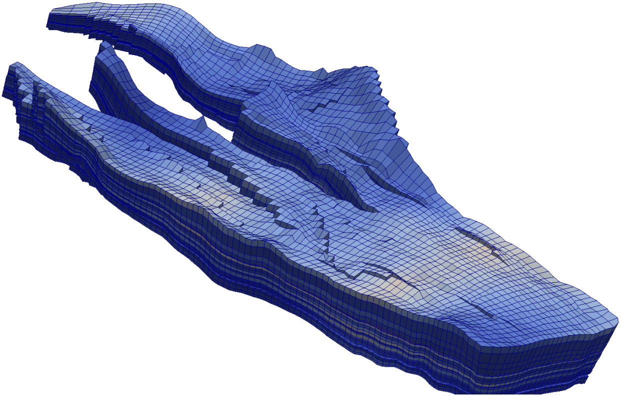




[exaggerated $5\times$ in z direction]

Grids with faults

Introduces non-neighbour connections / non-matching grid / non-conforming grid (w.r.t. logical Cartesian topology).



Our approach: ignore original Cartesian structure, all unstructured.

The `CpGrid` class — features

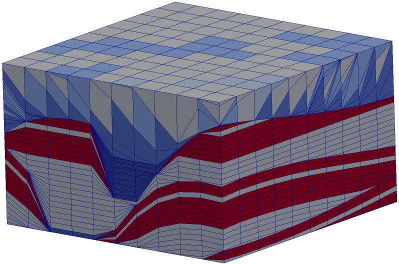
Features:

- Supports the `dune` grid interface.
  - Iterators
  - Intersections
  - Geometries
  - Cells are of type `cube` (but no mappings)
  - Intersections are of type `none`
- Reads ECLIPSE input files (commercial simulator).
- Relatively lightweight

Non-features:

- No adaptivity.
- No parallellism.
- No construction from grid factory

The `CpGrid` class — implementation



Internally, every entity and intersection has an immutable index.

$\Longrightarrow$ Easy to store geometry in arrays.

Topology is stored in sparse-matrix-like structures
(using a custom template class `SparseTable<T>`).

Could in theory support random access.

Introduction

The dune-cornerpoint module

The dune-porsol module

The dune-upscaling module

Future work: OPM

Immiscible and incompressible two-phase porous media flow

This may be written as the following nonlinear system (global pressure fractional flow formulation):

$$-\nabla \cdot \Big(\overbrace{\lambda(s_w)\mathbf{K}\nabla p - (\lambda_w(s_w)\rho_w + \lambda_o(s_w)\rho_o)\mathbf{K}\mathbf{g}}^{\mathbf{v}} \Big) = q$$

$$\phi \frac{\partial s_w}{\partial t} + \nabla \cdot \Big(f_w(s_w)\mathbf{v} + \gamma(s_w)\mathbf{K}\nabla p_{cow}(s_w) + \gamma(s_w)\mathbf{K}\Delta\rho\mathbf{g} \Big) = \frac{q_w}{\rho_w}$$

λ_w	water phase mobility	λ_o	oil phase mobility
λ	total mobility $= \lambda_w + \lambda_o$	$\mathbf{K}$	permeability
p	global pressure	p_{cow}	capillary pressure $= p_o - p_w$
f_w	fractional flow $= \lambda_w \lambda^{-1}$	γ	$= \lambda_w \lambda^{-1} \lambda_o$
s_w	water phase saturation	$\mathbf{g}$	gravity vector
ρ_w	water phase density	ρ_o	oil phase density
$\Delta\rho$	$= \rho_w - \rho_o$	q	volumetric source term

Operator splitting

This nonlinear system may be solved fully implicitly, or sequentially (operator splitting strategy):

Given the saturation s_w^n at time step n, solve

$$-\nabla \cdot \Big(\lambda^n \mathbf{K} \nabla p - (\lambda_w^n \rho_w + \lambda_o^n \rho_o) \mathbf{K}\mathbf{g}\Big) = q$$

to find $\mathbf{v}^{n+1}$. Then, find s_w^{n+1} by solving

$$\phi \frac{\partial s_w}{\partial t} + \nabla \cdot \Big(f_w(s_w) \mathbf{v}^{n+1} + \gamma(s_w) \mathbf{K} \nabla p_{cow}(s_w) + \gamma(s_w) \Delta\rho \mathbf{K}\mathbf{g} \Big) = \frac{q_w}{\rho_w}$$

with an implicit or explicit method.

Pressure/flow equation

The (global) pressure equation is elliptic.

$$-\nabla \cdot \Big(\lambda \mathbf{K} \nabla p - (\lambda_w \rho_w + \lambda_o \rho_o) \mathbf{K} \mathbf{g} \Big) = q$$

$\Longrightarrow$ may solve it with a wide variety of methods.

However, we want an accurate flux/velocity field $\mathbf{v}$ for the transport part
$\Longrightarrow$ usually finite volume, mimetic or mixed finite elements used.

Mixed hybridized formulation

Start from mixed form of (basic) pressure equation:

$$\begin{aligned} \nabla \cdot \mathbf{v} &= q \\ \mathbf{v} &= -\mathbf{K}\nabla p \end{aligned}$$

Weak hybridized form is:

$$\begin{aligned} b(u,v) - c(u,p) &= 0 & \forall u &\in H^{div}(C) \\ c(v,l) &= (q,l) & \forall l &\in L^2(\Omega) \\ d(v,\mu) &= 0 & \forall \mu &\in L^2(\partial\Omega_h \setminus \partial\Omega). \end{aligned}$$

The most important bilinear form is b, given by

$$b(\cdot,\cdot) : H^{div}(\Omega) \times H^{div}(\Omega) \to \mathbb{R}$$

$$b(u,v) = \int_\Omega u \cdot \mathbf{K}^{-1} v \, dx.$$

Mixed FEM discretisation vs. mimetic

For general domains, we want to avoid doing quadrature.
The bilinear form b is therefore (inspired by the Gauss theorem) replaced by

$$m(\cdot,\cdot) : L^2(\partial\Omega_h) \times L^2(\partial\Omega_h) \to \mathbb{R}$$
$$m(u,v) = \sum_{E_i \in \Omega_h} \left(u^{(i)}\right)^T M^{(i)} v^{(i)}.$$

This is consistent and reproduces linear solutions for certain (non-unique) choices of the inner product M.

(Bernd gave example yesterday of non-$\mathbf{K}$-orthogonal case: fails for TPF, succeeds for mimetic)

Mimetic finite difference discretisation — features

- Applies to grids with arbitrary cell shapes.
- For planar faces: 1 unknown per face, for curved faces: 3 unknowns per face.
- (cheat: use only one unknown anyway)

The mimetic method produces a linear system of the same kind as mixed finite elements (with hybridization):

$$\begin{pmatrix} B & C & D \\ C^T & & \\ D^T & & \end{pmatrix} \begin{pmatrix} v \\ p \\ \pi \end{pmatrix} = \begin{pmatrix} f \\ g \\ h \end{pmatrix}$$

Use Schur complement method to reduce to system for the face pressures π.

The IncompFlowSolverHybrid class

Assembly is done by the template class `IncompFlowSolverHybrid`

```
template <class GridInterface,
          class RockInterface,
          class BCInterface,
          template <class GridIF, class RockIF> class InnerProduct>
class IncompFlowSolverHybrid
{
    template<class Point>
    void init(const GridInterface&  g,
              const RockInterface&  r,
              const Point&          grav,
              const BCInterface&    bc);

    template<class FluidInterface>
    void solve(const FluidInterface&      r  ,
               const std::vector<double>& sat,
               const BCInterface&         bc ,
               const std::vector<double>& src,
               double residual_tolerance = 1e-8,
               int linsolver_verbosity = 1,
               int linsolver_type = 1,
               bool same_matrix = false);

    SolutionType getSolution();

...
};
```

Inner products

So far, we have two available inner product evaluators:

1. `MimeticIPEvaluator`
   - for scalar mobilities
   - caches the inner product, so that when mobilities change, minimal work needs to be performed
2. `MimeticIPAnisoRelpermEvaluator`
   - for anisotropic (tensor) mobilities
   - caches some, but not all of the inner product

Transport equation

The transport equation is parabolic, but for low capillary pressure gradients, behaviour is more like a hyperbolic equation.

$$\phi\frac{\partial s_w}{\partial t}+\nabla\cdot\Big(f_w(s_w)\mathbf{v}^{n+1}+\gamma(s_w)\mathbf{K}\nabla p_{cow}(s_w)+\gamma(s_w)\Delta\rho\mathbf{K}\mathbf{g}\Big) = \frac{q_w}{\rho_w}$$

We solve this (at the moment) with an explicit Euler discretization in time, and a finite volume discretization in space.

The EulerUpstream class

```
template <class GridInterface, class ReservoirProperties, class BCInterface>
class EulerUpstream
{
    EulerUpstream(const GridInterface&       g,
                  const ReservoirProperties& resprop,
                  const BCInterface&         bc);

    template <class PressureSolution>
    void transportSolve(std::vector<double>&                   saturation,
                        const double                           time,
                        const typename GridInterface::Vector&  gravity,
                        const PressureSolution&                pressure_sol,
                        const SparseVector<double>&            injection_rates) const;
};
```

Intel threading building blocks (TBB) has been used for shared-memory parallization.

Rock and fluid properties

Current interface example:

```
class SomeArbitraryRockFluidPropertyClass
{
    // Phase related
    double viscosityFirstPhase() const;
    double viscositySecondPhase() const;
    double densityFirstPhase() const;
    double densitySecondPhase() const;

    // Rock related
    double porosity(int cell_index) const;
    PermTensor permeability(int cell_index) const;
    double capillaryPressure(int cell_index, double saturation) const;
    double saturationFromCapillaryPressure(int cell_index, double cap_press) const;

    // Fluid-rock interaction
    double mobilityFirstPhase(int cell_index, double saturation) const;
    double mobilitySecondPhase(int cell_index, double saturation) const;
    double totalMobility(int cell_index, double saturation) const;
    double fractionalFlow(int cell_index, double saturation) const;
    ...
}
```

Going forward, we will largely adopt Dumux conventions and interfaces instead.

Introduction

The `dune-cornerpoint` module

The `dune-porsol` module

The `dune-upscaling` module

Future work: OPM

Upscaling, what and why?

Upscaling means computing *effective parameters* for a coarse scale simulation from fine-scale properties.

Examples include porosity, permeability, relative permeability, capillary pressure.

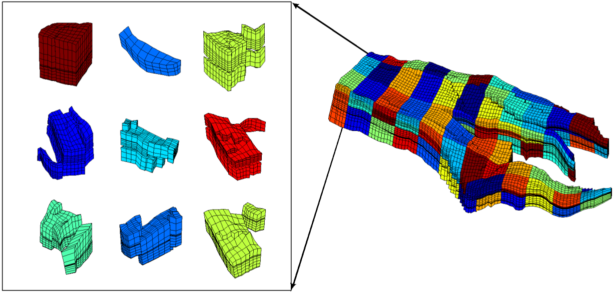
Upscaling is popular because:

1. Porous media flows are multi-scale phenomena.
2. Large number of grid cells necessary for accuracy.
3. Complex physics make problems hard to compute with high resolution in reasonable time.

Alternatives to upscaling

Recent *multi-scale* methods can be viewed as an alternative to upscaling.

Example: Multiscale mixed FEM methods.

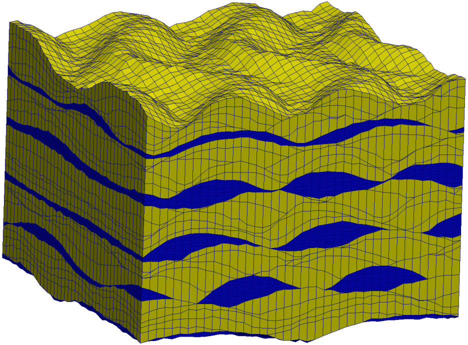


Compared to permeability upscaling:

- coarse scale basis functions resolve fine-grid effects better
- may recompute (some) basis functions when flow patterns change
- much greater coarse block flexibility

Single-phase upscaling

In single-phase upscaling we seek to compute *effective coarse-scale permeabilities* by solving fine-scale flow problems.



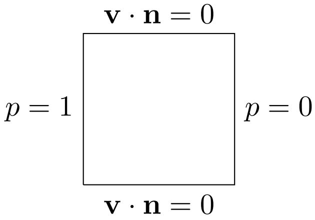
Fine-scale permeability field.

89791 cells, approximately $40 \times 40 \times 6$ cm, upscaled permeability

$$\mathbf{K}_{eff} = \begin{pmatrix} 156.11 & 0 & 0 \\ 0 & 161.81 & 0 \\ 0 & 0 & 31.25 \end{pmatrix}$$

Simple upscaling example

For a unit-sqare fine-scale domain $[0,1]^2$, solve the single-phase pressure equation with boundary conditions:



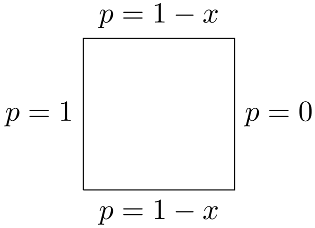
$$\nabla\cdot\mathbf{v}=0$$
$$\mathbf{v}=-\mathbf{K}\nabla p$$

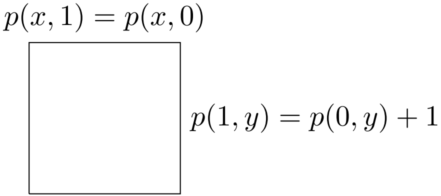
From solution, compute average velocity over boundaries and estimate $\mathbf{K}_{eff}$ for the block (x direction) from Darcy's law. Repeat for other directions.

To apply this method, the domain must be reasonably close to a shoe-box.

Other boundary conditions

Alternative boundary conditions are possible:





For linear BCs (left) we get a full tensor, which may not be symmetric.

For periodic BCs (right) we get a full, symmetric tensor.

The class SinglePhaseUpscaler implements the method.

```
...

/// Initializes the upscaler from parameters.
void init(const parameter::ParameterGroup& param);

/// Does a single-phase upscaling.
/// @return an upscaled permeability tensor.
permtensor_t upscaleSinglePhase();

/// Compute upscaled porosity.
/// @return total pore volume of all cells divided by total volume.
double upscalePorosity() const;

...
```

Steady-state upscaling

We use steady-state upscaling to upscale relative permeabilities in the presence of nonzero capillary pressure.

Idea: Simulate till steady state reached, use the steady saturation distribution for upscaling computations.

To do this we simulate two-phase flow with:

- pressure BCs as for single-phase flow
- either periodic BCs for saturation, or some given inflow saturation

The resulting properties depend on the pressure drop!
(transition from capillary limit to viscous limit)

The class SteadyStateUpscaler

The template class SteadyStateUpscaler implements the algorithm. The Traits allow us (for example) to use either scalar and tensor relative permeabilities.

```
template <class Traits>
class SteadyStateUpscaler : public UpscalerBase<Traits>
{
    std::pair<permtensor_t, permtensor_t>
    upscaleSteadyState(const int flow_direction,
                       const std::vector<double>& initial_saturation,
                       const double boundary_saturation,
                       const double pressure_drop,
                       const permtensor_t& upscaled_perm);
    const std::vector<double>& lastSaturationState() const;
    double lastSaturationUpscaled() const;
    ...
};
```

Introduction

The dune-cornerpoint module

The dune-porsol module

The dune-upscaling module

Future work: OPM

Why OPM?

OPM goal

Provide researchers and students with open simulator code for porous media problems

Subgoals:

- handle real industrial cases
- good performance
- extensibility
- ease of use

Current OPM codes

Subproject	Lines of code	Creator
Dumux	50k + 200k	University of Stuttgart
Dune-cornerpoint etc.	39k	SINTEF
MPFA based simulator	16k	IRIS

Short-term goals

This fall we are building...

1. A black-oil simulator:
    - three phases
    - compressibility
    - miscibility
    - operator splitting approach
    - implicit pressure (mimetic method)
    - explicit or implicit transport.
2. Multiscale Mixed FEM method:
    - initially only incompressible, immiscible two-phase
    - basis functions computed by mimetic method, with or without overlap
    - flexible coarse block partitioning