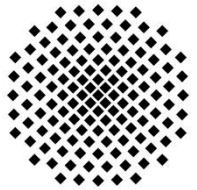
Coupling a discrete 1D network with a 3D surrounding bulk using DuMu^x

Natalie Schröder, University of Stuttgart
Timo Koch, University of Stuttgart



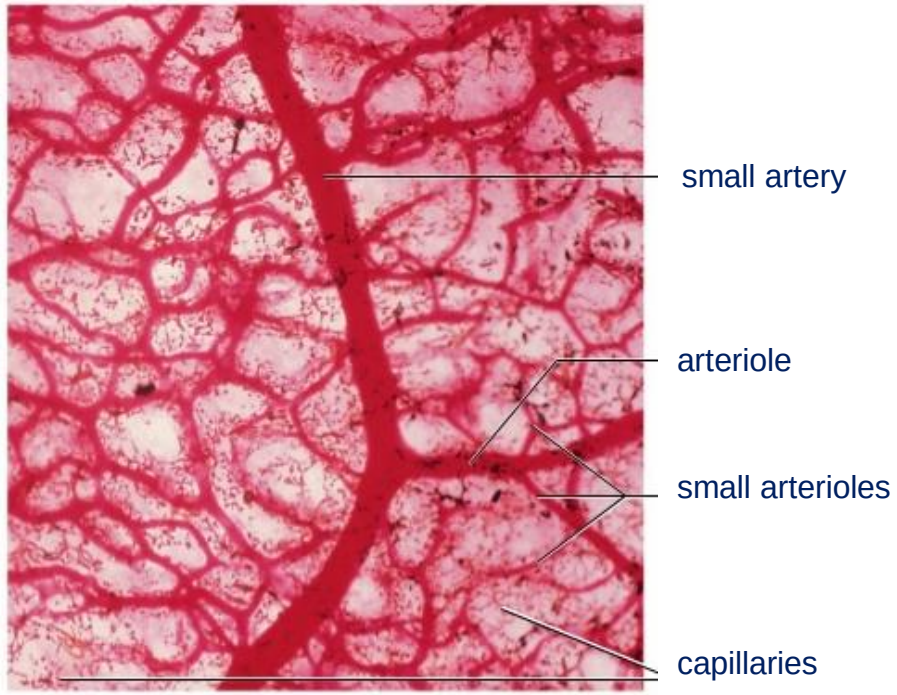
University of Stuttgart
Germany

Example applications

1. Coupling root network with embedding soil matrix

Quelle: © iStockphoto / Thomas Vogel

2. Coupling microcirculation with embedding tissue



Quelle: © 2012 Pearson Education, Inc.

Application 1: Root-Soil Interactions

- Soil flow (Richard‘s equation)

$$\frac{\partial \phi S_w \rho_w}{\partial t} - div\{\rho_w \frac{k_{rw}}{\mu_w} \mathbf{K}(\mathbf{grad} p_w - \rho_w \mathbf{g})\} = q_w$$

- Root water flow (Darcy‘s law)

$$v_x = -K_x \frac{dp_x}{dz}$$

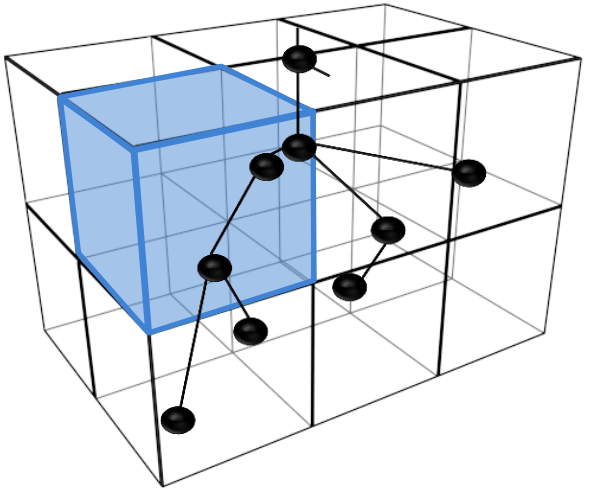
$$div(v_x(p_x)) = q(p_x)$$

$$q(p_x) = v_r = K_r A_r (p_s - p_x)$$

Coupling water flow

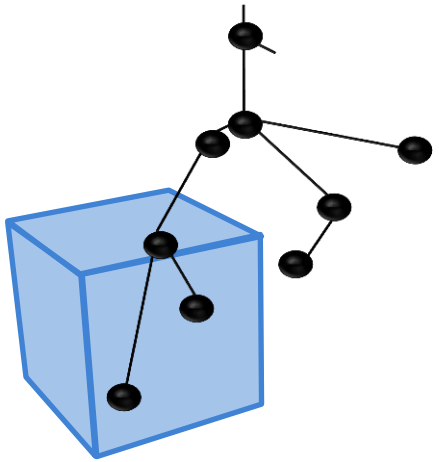
water sink term

$$q(p_x) = \frac{\sum v_r(p_x)}{volume}$$



Pressure around root segment

$$p_{cube} = p_s$$



Application 2: Vessel-Tissue Interaction

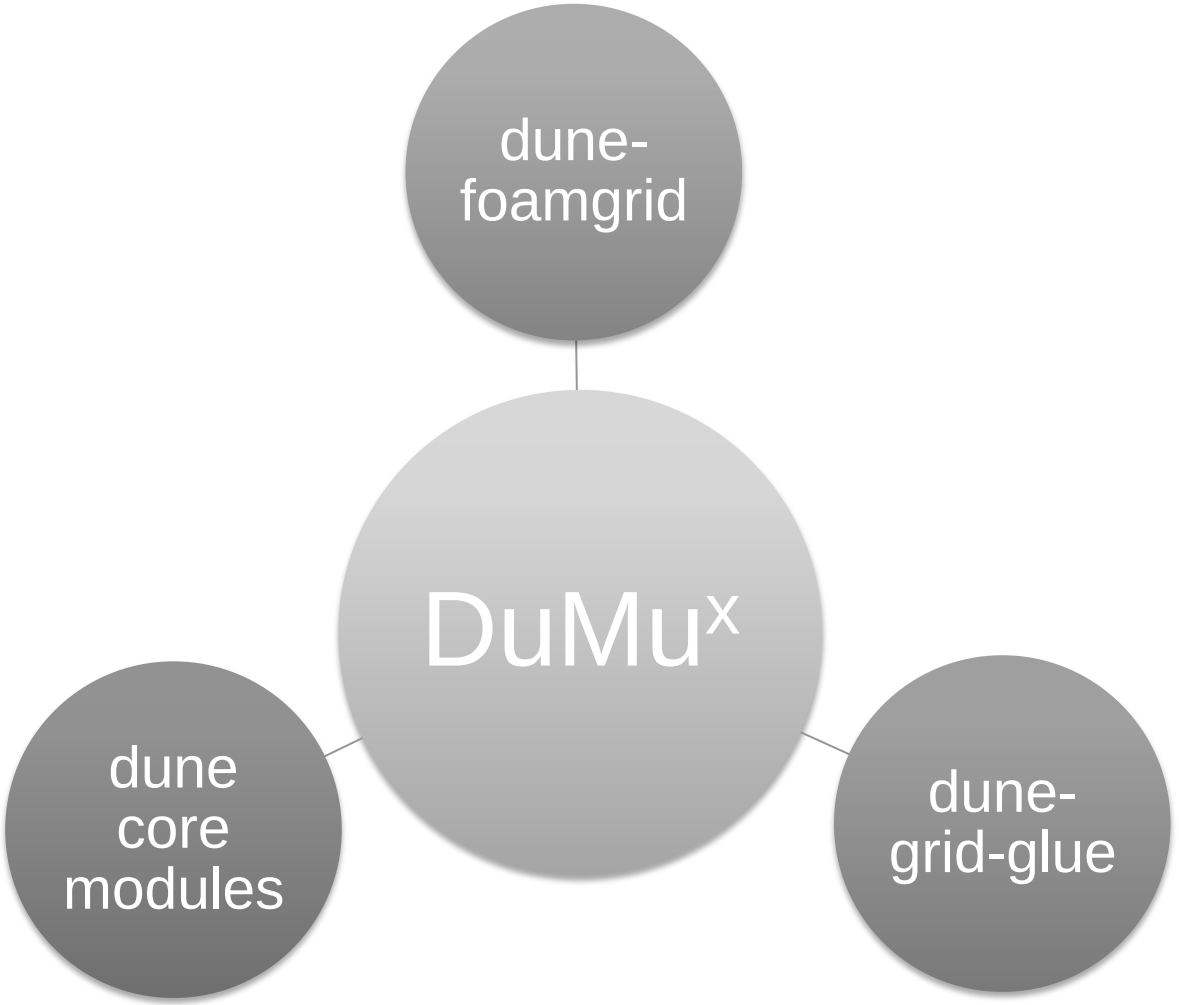
- Poiseuille flow in the blood vessels

$$\left.\begin{aligned} \bar{v}\mathbf{n_z} &= -\frac{R^2}{2\mu(2+\gamma)}\frac{\partial p_\mathrm{f}}{\partial z}\mathbf{n_z} \\ -\frac{\partial(\pi R^2 \bar{v})}{\partial z} &= 2\pi R \frac{K_\mathcal{M}}{\mu_i d_\mathcal{M}}(p_\mathrm{f} - \bar{p}_\mathrm{p}) \end{aligned}\right\} \quad \text{on } \Gamma$$

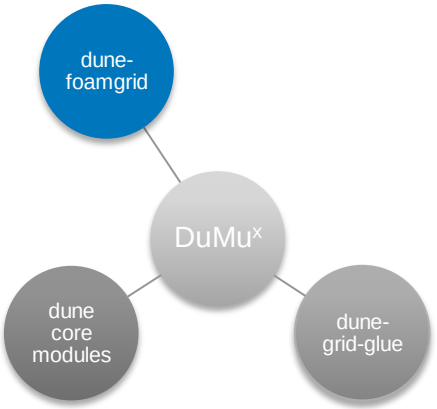
- Darcy flow in the tissue (solid phase : cells, fibres, …
  fluid phase : interstitial fluid)

$$\left.-\nabla\cdot\left(\frac{K}{\mu_i}\nabla p_\mathrm{p}\right) = 2\pi R\frac{K_\mathcal{M}}{\mu_i d_\mathcal{M}}(p_\mathrm{f} - \bar{p}_\mathrm{p})\delta_\Gamma \right\} \quad \text{in } \Omega$$

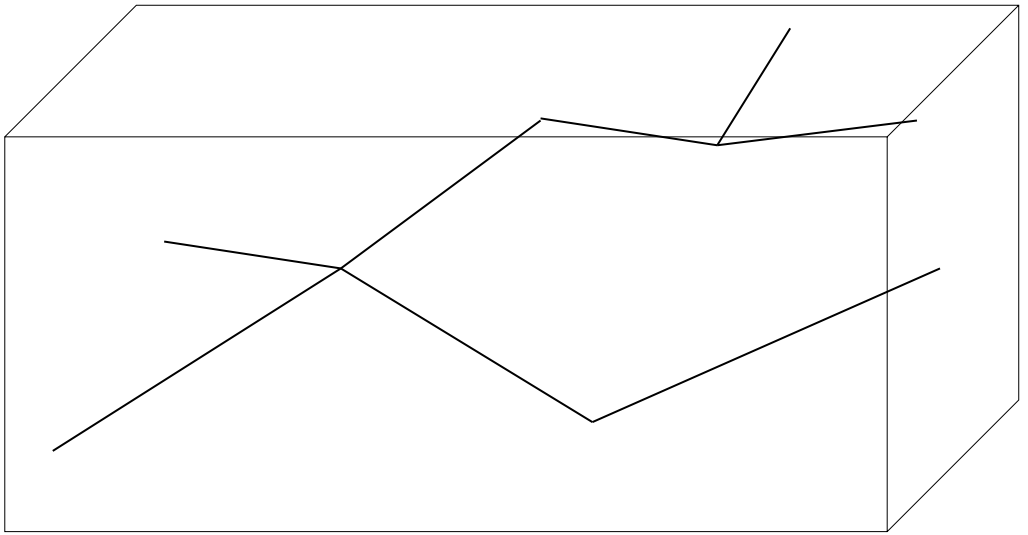
Realisation in DuMu^x: Dependencies



Dune-FoamGrid

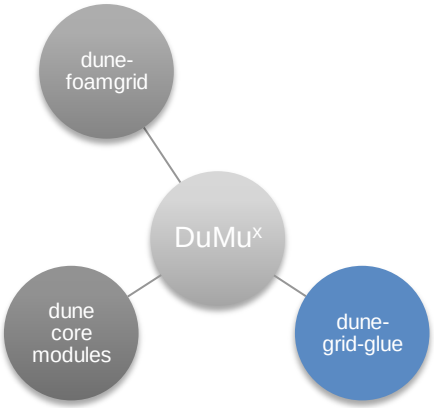


- 1D and 2D in n-dimensional grids
- Branching allowed (creates non-manifold grid)
- “Growth” possible (soon)
- Element parametrizations



e.g. a 1D in 3D FoamGrid

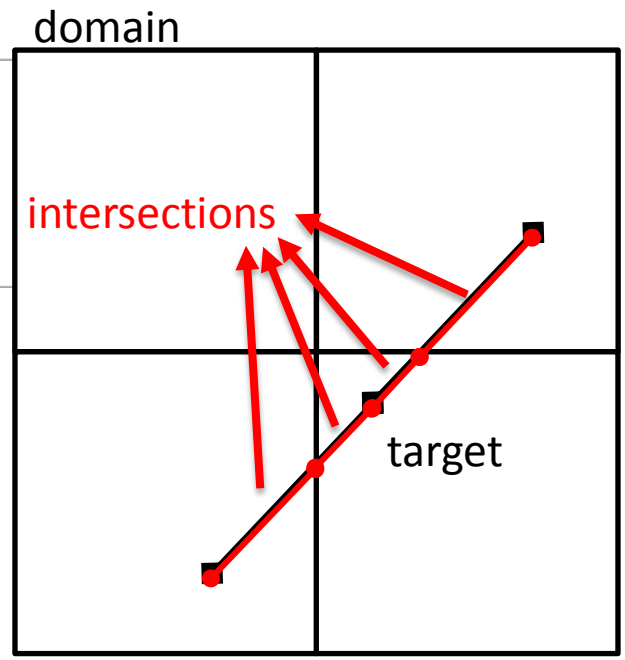
Dune-Grid-Glue



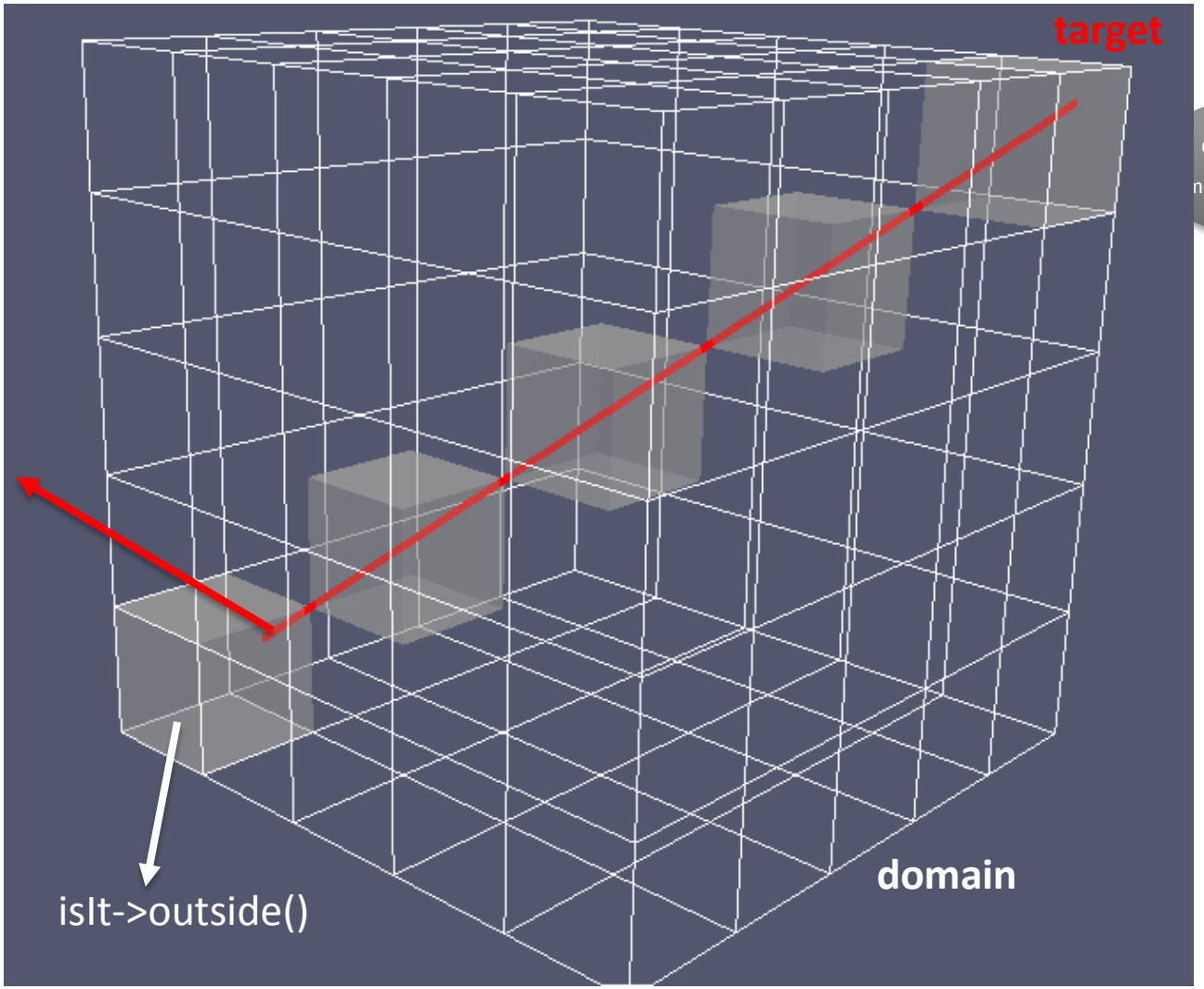
- Recently adapted to merge 1D and 3D grids
- Functional principle:

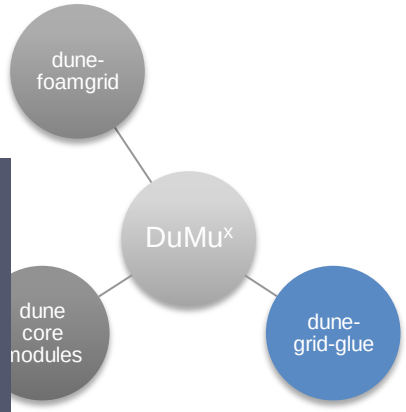
C++ code

```
RemoteIntersectionIterator isIt;
//...
auto domain_element = isIt->inside();
auto target_element = isIt->outside();
```

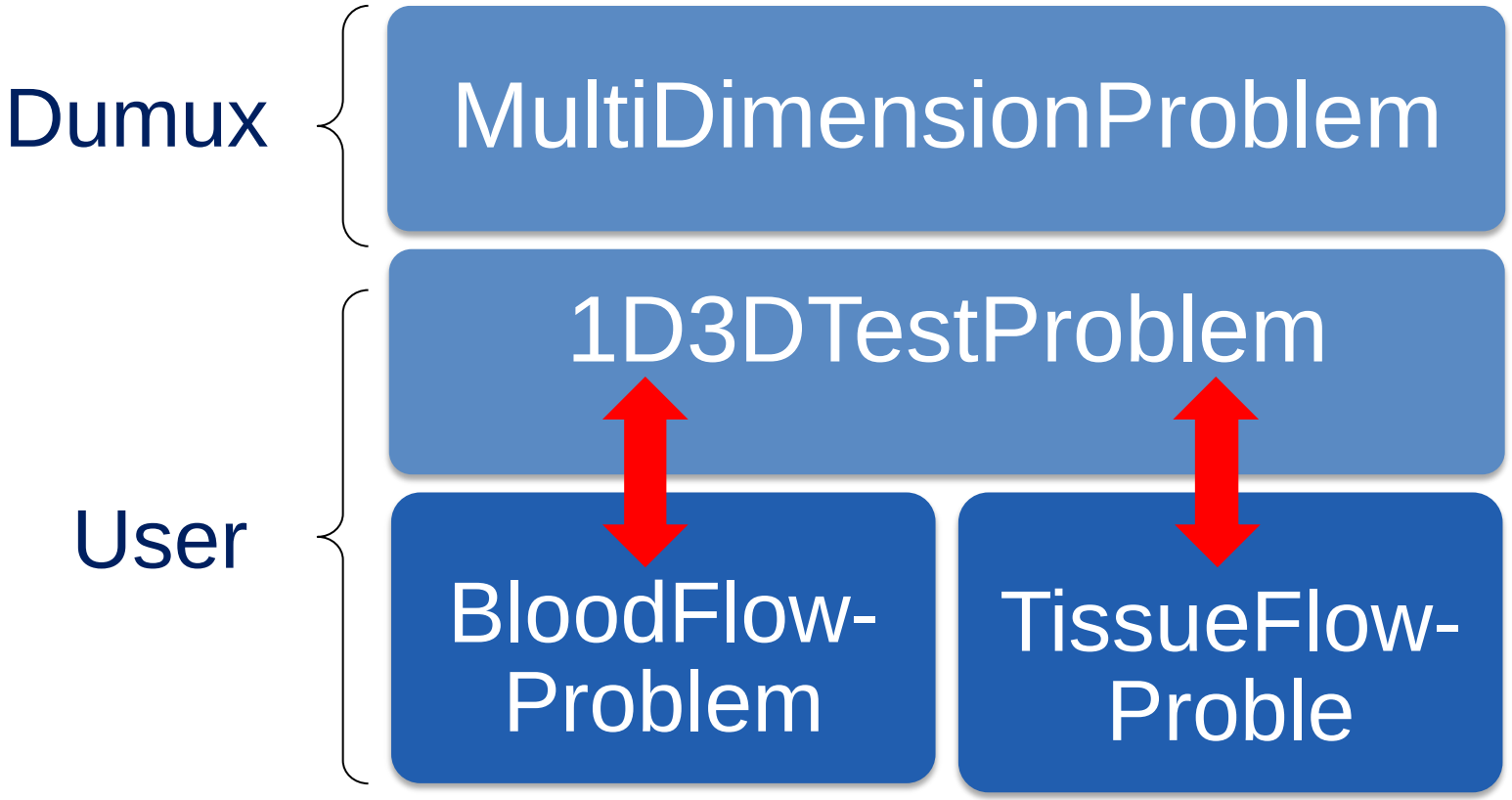


Dune-Grid-Glue





Realisation in DuMu^x: Implementation



set and get information (two-sided access)

Dumux::MultiDimensionProblem

C++ code

```
//...
namespace Dumux
{
namespace Properties
{
// NumericModel provides Scalar, GridCreator, ParameterTree
NEW_TYPE_TAG(MultiDimensionProblem,
    INHERITS_FROM(NumericModel));

// property tags that will be set in the problem at hand
NEW_PROP_TAG(SubProblem1TypeTag);
NEW_PROP_TAG(SubProblem2TypeTag);
NEW_PROP_TAG(MultiDimensionMapper);
NEW_PROP_TAG(CouplingTransferData);
NEW_PROP_TAG(Problem);
//...
} // end namespace Properties
//...
public:
    // Constructor
    MultiDimensionProblem(TimeManager &timeManager, const
        GridView1 &gridView1, const GridView2 &gridView2)
    : timeManager_(timeManager), gridView1_(gridView1),
        gridView2_(gridView2)
    {
        //...
    }
//...
```

Dumux::MultiDimensionProblem

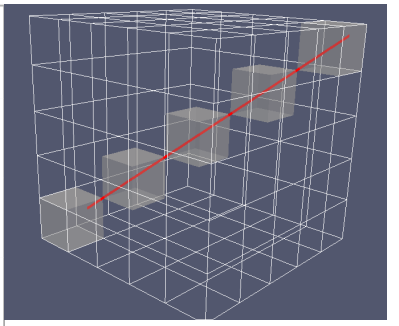
C++ code

```
//...
void init()
{
    //...
    // initialize the subproblem time managers (this also
        initializes the subproblems)
    asImp_().subTimeManager1().init(asImp_().subProblem1(),
        tStart, dtSubProblem1, tEnd, restart);
    asImp_().subTimeManager2().init(asImp_().subProblem2(),
        tStart, dtSubProblem2, tEnd, restart);

    // glue the two grids (find intersections)
    callGridGlue();
}
```

Dumux::MultiDimensionProblem

C++ code

```
//...
void callGridGlue()
{
    // target: gridview1, domain: gridview2
    // specify the target and domain
    AllElementsDescriptor<GridView1> tardesc;
    AllElementsDescriptor<GridView2> domdesc;

    TarExtractor tarEx(gridView1_, tardesc);
    DomExtractor domEx(gridView2_, domdesc);

    Dune::GridGlue::MixedDimOverlappingMerge<dim2, dim1,
        dimworld1> merger;
    GlueType glue(domEx, tarEx, &merger);

    //calculate the intersections
    glue.build();

    //write intersections to file for debug purposes
    GridGlueVtkWriter glueVtk;
    glueVtk.write(glue, name());

    //write maps of geometric information
    MultiDimensionMapper mapper(glue, map1_, map2_);
}
```



Dumux::1D3DTestProblem

C++ code

```
//...
// Set the two sub-problems of the global problem
SET_TYPE_PROP(TestOneDThreeDProblem, SubProblem1TypeTag,
    TTAG(BloodFlowProblem));
SET_TYPE_PROP(TestOneDThreeDProblem, SubProblem2TypeTag,
    TTAG(TissueFlowProblem));
//...
void timeIntegration()
{
    std::cout << std::endl;
    std::cout << "Start coupled time integration starting
        at t = " << this->timeManager().time() << std::endl;
    //...
    // iterative algorithm within each time step
    while(error > tolerance_)
    {
        // get the data from the 3d problem
        updateSource1D();

        // run the 1d problem
        //...

        // get the data from the 1d problem
        updateSource3D();

        // run the 3d problem
        //...

        // calculate the error
        error = ...;
    }
}
```

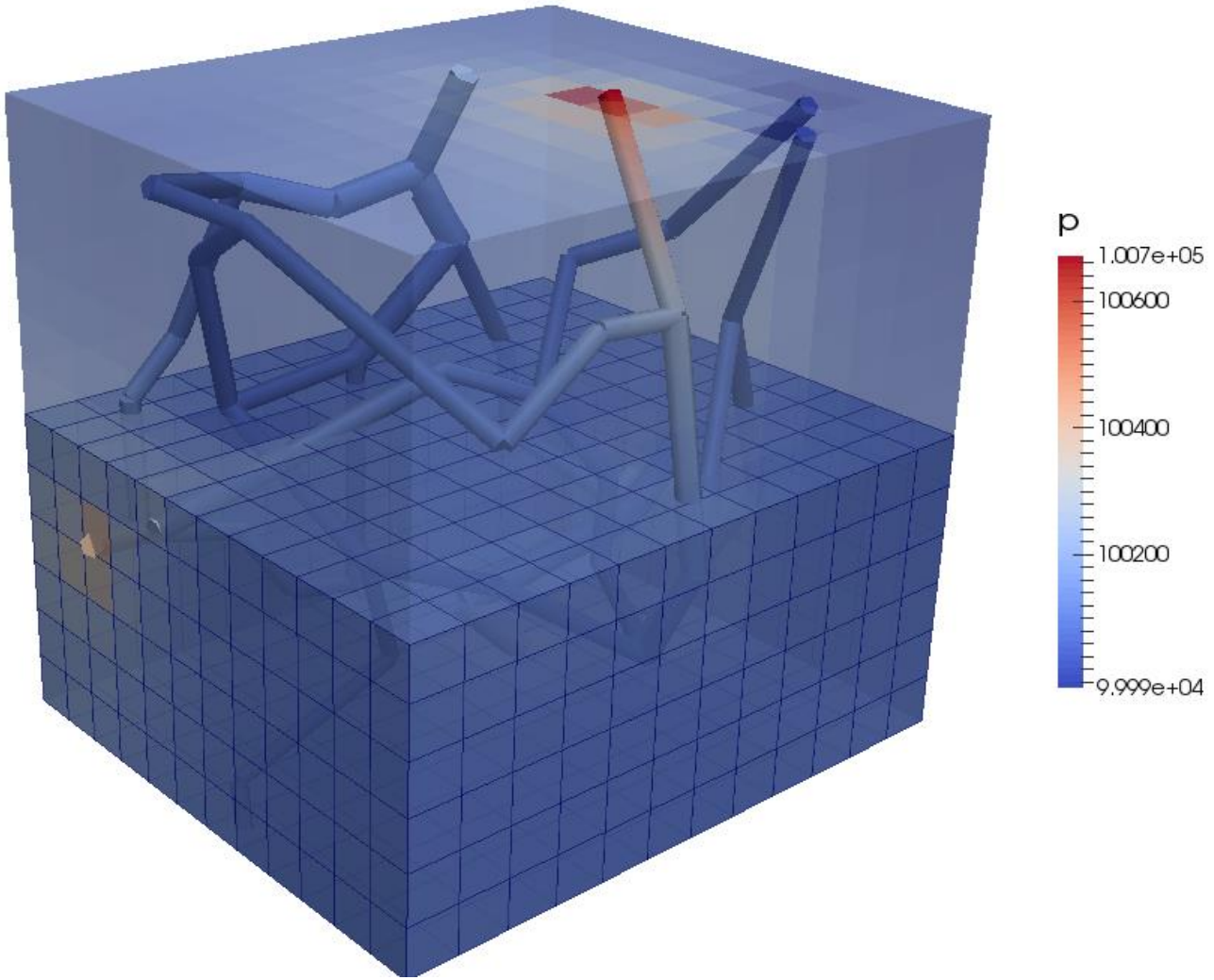
Dumux::BloodFlowProblem

C++ code

```
//...
typedef typename GET_PROP_TYPE(TypeTag,
    CouplingTransferData) MapType;
//...

    void setMap (MapType *map)
    {
        oneDMap_ = map;
    }

private:
    MapType *oneDMap_;
```

Simulation results

Coupling microcirculation with embedding tissue



Simulation results

Coupling root network with embedding soil matrix