Mathematics in Applications


University of Graz, Medical University of Graz, King’s College London, ETH Zürich

CERN, Sept. 19, 2019
Bidomain equations:

\[
- \nabla \cdot (\sigma_i + \sigma_e) \nabla u_e - \nabla \cdot \sigma_i \nabla v = I_e(t)
\]

\[
\nabla \cdot \sigma_i \nabla v + \nabla \cdot \sigma_i \nabla u_e = C_m \frac{\partial v}{\partial t} + I_{ion}(v, w)
\]

\[
\frac{\partial w}{\partial t} = g(v, w)
\]

- \( v \): transmembrane potential
- \( l_e \): extracellular charge
- + nonlinear elasticity + CFD
"usefull" decomposition of discretized problem:
Fast solution algorithms

- "usefull" decomposition of discretized problem:

- Solve with appropriate algorithms on clusters of CPUs/GPUs

Moderate hardware:
- CPU cores: 128 IMSC; $10^3$
  KFU; $10^5$ VSC;
  $10^6$ PRACE
- GPU cluster;
- Knights Landing: $2 \times 64$ cores
- 15 min on 8192 CPU-cores
  (140 Mill. tets)

CARP: MUG+KFU+TUG;
toolbox (CPU □/GPU □) vs. Petsc (hypre) x on mephisto and hector [II/2012]
Hybrid Parallelization in the Solvers

- **Equally balanced domain interfaces** improve scalability.
  - new load balancing of interface nodes
  - eliminates inconsistent round-off errors

- **Reordering** of unknowns avoids buffers for communication.
  \[ \implies 3 \times \text{faster data exchange than standard} \]

- **Redistribution** of Algebraic Multigrid Hierarchy
  - coarse-grid domain redistribution (blocking)
  - inactive compute cores on coarser grids
  - less MPI communication

- Two-layer data distribution in the Toolbox for OpenMP + MPI
Improvements in strong scalability [2016]

- Blocking of coarse grids ($g=2$) + hybrid OpenMP/MPI
  - 6.9 Mill. dofs, elliptic part of bidomain equations, 200 solves (5 ms), 6500 AMG-PCG iterations
  - **blocking** pays off for $n_{procs} > 128$.
  - **hybrid** OpenMP/MPI pays off for $n_{procs} > 512$.
  - varies slightly with available hardware
Example: elasticity + potential problem

TBunnyC: 862,515 vertices \((n + 3 \times n \text{ dofs})\) on CPU [GPU: 10×]
Patient specific simulations

Tissue properties $\sigma_i, \sigma_e$ in
\[-\nabla \cdot (\sigma_i + \sigma_e) \nabla u_e - \nabla \cdot \sigma_i \nabla v = I_e(t)\]
- a priori known values have a huge uncertainty
- in vivo measurements for $\sigma_i, \sigma_e$ not possible with human
- tensor directions known from physiological maps

Compare calculated ECG with individually measured ECG:
- Idea: Change $\sigma \rightarrow \sigma_{\text{patient}}$ s.t. $\text{ECG}_{\text{calc}} \rightarrow \text{ECG}_{\text{meas}}$
- By Means of Eikonal solver (excitation pattern)
  - Math. optimization
  - Deep learning

10 mm = 1 mV

D. Ganellari [Graz → ETH Zürich]
Non-linear PDE encountered in problems of wave propagation.

\[
\begin{align*}
H(x, \nabla \phi) &= \sqrt{(\nabla \phi)^T M(\nabla \phi)} = 1, & \forall x \in \Omega \subset \mathbb{R}^3 \\
\phi(x) &= B(x), & \forall x \in B \subset \Omega
\end{align*}
\]

- $\Omega$ - 3D domain approximated by planar side-sided tetrahedralization.
- $\phi(x)$ - The shortest time needed to travel from the boundary to $x$ inside $\Omega$.
- $M(x)$ - 3x3 symmetric positive-definite matrix encoding speed information on $\Omega$.
- $B$ - set of smooth boundary conditions which adhere to the consistency requirements of the PDE.

Fast iterative method (FIM) to solve the Eikonal equation on tetrahedral domains [Fu, Kirby, Whitaker 2013].
Denote the travel time from $x_5$ to $x_4$:

$$\phi_{5,4} = \phi_4 - \phi_5 = \sqrt{e_{5,4}^T M e_{5,4}} \quad (3)$$

$$e_{5,4} = x_4 - x_5 \quad (4)$$

$$e_{5,4} = x_4 - (\lambda_1 x_1 + \lambda_2 x_2 + (1 - \lambda_1 - \lambda_2) x_3)$$

$$e_{5,4} = [e_{1,3}, e_{2,3}, e_{3,4}] \lambda \quad \text{with} \quad \lambda = [\lambda_1, \lambda_2, 1]^T$$

with common vertex $x_3$

$$e_{5,4}^T M e_{5,4} = \lambda^T [e_{1,3}^T e_{2,3}^T e_{3,4}^T] M [e_{1,3} e_{2,3} e_{3,4}] \lambda = \lambda^T M' \lambda \quad (5)$$

with $M' = \begin{cases} 
\alpha := [e_{1,3}^T M e_{1,3}, e_{2,3}^T M e_{1,3}, e_{3,4}^T M e_{1,3}]^T \\
\beta := [e_{1,3}^T M e_{2,3}, e_{2,3}^T M e_{2,3}, e_{3,4}^T M e_{2,3}]^T \\
\gamma := [e_{1,3}^T M e_{3,4}, e_{2,3}^T M e_{3,4}, e_{3,4}^T M e_{3,4}]^T 
\end{cases} \quad (6)$

Interpolating $\phi_5 = \lambda_1 \phi_1 + \lambda_2 \phi_2 + (1 - \lambda_1 - \lambda_2) \phi_3$ we can express $\phi_4$
Wave propagation in one tetrahedron II

We want to minimize the time

$$\phi_4(\lambda_1, \lambda_2) = -\lambda_1 \phi_{1,3} - \lambda_2 \phi_{2,3} + \phi_3 + \sqrt{\lambda^T M' \lambda}$$

wrt. $\lambda$, i.e., we are looking for $\min_{\lambda_1, \lambda_2} \phi_4$.

The necessary conditions $\frac{\partial \phi_4}{\partial \lambda_1} = 0$ and $\frac{\partial \phi_4}{\partial \lambda_2} = 0$ lead to the non-linear system of equations:

$$\begin{cases}
\phi_{1,3} \sqrt{\lambda^T M' \lambda} = \lambda^T \alpha \\
\phi_{2,3} \sqrt{\lambda^T M' \lambda} = \lambda^T \beta
\end{cases}$$

which can be reformulated as:

Determine $\lambda_1, \lambda_2$ from

$$\begin{cases}
\phi_{1,3} \sqrt{\lambda^T M' \lambda} = \lambda^T \alpha \\
\phi_{2,3} \lambda^T \alpha = \phi_{1,3} \lambda^T \beta
\end{cases} \quad (7)$$
Each solving of system (7) with matrix

\[
M' = \begin{cases} 
\alpha := [e_{1,3}^T Me_{1,3}, e_{2,3}^T Me_{1,3}, e_{3,4}^T Me_{1,3}]^T \\
\beta := [e_{1,3}^T Me_{2,3}, e_{2,3}^T Me_{2,3}, e_{3,4}^T Me_{2,3}]^T \\
\gamma := [e_{1,3}^T Me_{3,4}, e_{2,3}^T Me_{3,4}, e_{3,4}^T Me_{3,4}]^T 
\end{cases}
\]

requires coordinates of \( x_1, x_2, x_3, x_4 \) and symm. matrix \( M \).

- \((4 \times 3 + 6)\) floats from global memory + local storage for \( e_{i,j} \) + inner products
Memory footprint

Each solving of system (7) with matrix

\[
M' = \begin{cases} 
\alpha := [e_{1,3}^T Me_{1,3}, e_{2,3}^T Me_{1,3}, e_{3,4}^T Me_{1,3}]^T \\
\beta := [e_{1,3}^T Me_{2,3}, e_{2,3}^T Me_{2,3}, e_{3,4}^T Me_{2,3}]^T \\
\gamma := [e_{1,3}^T Me_{3,4}, e_{2,3}^T Me_{3,4}, e_{3,4}^T Me_{3,4}]^T 
\end{cases}
\]

requires coordinates of \(x_1, x_2, x_3, x_4\) and symm. matrix \(M\).

- \((4 \times 3 + 6)\) floats from global memory + local storage for \(e_{i,j}\)
  + inner products

- How to reduce the memory footprint?
- Why not precompute \(M'_{k,l}\)? \(\longrightarrow\) 6 floats from global memory
Each solving of system (7) with matrix

\[
M' = \begin{cases} 
\alpha := [e_{1,3}^T Me_{1,3}, e_{2,3}^T Me_{1,3}, e_{3,4}^T Me_{1,3}]^T \\
\beta := [e_{1,3}^T Me_{2,3}, e_{2,3}^T Me_{2,3}, e_{3,4}^T Me_{2,3}]^T \\
\gamma := [e_{1,3}^T Me_{3,4}, e_{2,3}^T Me_{3,4}, e_{3,4}^T Me_{3,4}]^T
\end{cases}
\]

requires coordinates of \(x_1, x_2, x_3, x_4\) and symm. matrix \(M\).

- \((4 \times 3 + 6)\) floats from global memory + local storage for \(e_{i,j}\)
  + inner products

How to reduce the memory footprint?

- Why not precompute \(M'_{k,l}\)? \(\Rightarrow\) 6 floats from global memory

- 4 different matrices \(M'\) depending on unknown value \(\Phi_j\).
  - Precompute the 18 inner products needed for all \(M'\)'s (\(4 \times 3\) angles + 6 edges) \(\rightarrow\) symmetric sparse \(6 \times 6\) matrix \(T_M\) per tetrahedron

- Mapping problem: Map \(j \mapsto (k, l)\) (vertex \(\mapsto\) edge numbers) without IF-statements or lookup tables.
Gray Code Numbering of Edges (common vertex $x_3$)

<table>
<thead>
<tr>
<th>$x_4$</th>
<th>$x_3$</th>
<th>$x_2$</th>
<th>$x_1$</th>
<th>edge: $d$</th>
<th>matrix: $f(d)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>5</td>
</tr>
</tbody>
</table>

$f(d) = \text{round}(0.547826 \times d - 1.608695)$

$f_{\text{int}}(d) = d/2 - 1$
Extract the submatrix for common vertex $x_3$
(reference configuration)

**Algorithm 1** Accessing $M'$ wrt. common vertex $j$

1: Get Gray-code edge numbers $d_0(j), d_1(j), d_2(j)$
2: Get edge indices $k_i := f_{\text{int}}(d_i)$ \quad $i = 0, 1, 2$
3: $M' := T_M(k, k)$ \quad $\triangleright$ extract $3 \times 3$ matrix

Extracts the required entries entries of $M'$.

| $T_M$ | 0 \hline 0 & $e_{1,2}^T Me_{1,2}$ & $e_{1,2}^T Me_{1,3}$ & $e_{1,2}^T Me_{2,3}$ & $e_{1,2}^T Me_{1,4}$ & $e_{1,2}^T Me_{2,4}$ & $X$ \\
1 & $e_{1,3}^T Me_{1,3}$ & $e_{1,3}^T Me_{2,3}$ & $e_{1,3}^T Me_{1,4}$ & $X$ & $X$ & $e_{1,3}^T Me_{3,4}$ \\
2 & $e_{2,3}^T Me_{2,3}$ & $X$ & $X$ & $e_{2,3}^T Me_{2,4}$ & $e_{2,3}^T Me_{3,4}$ &  \\
... & & & & & &  \\
5 & & & & & & $e_{3,4}^T Me_{3,4}$  |
Rotating from reference configuration

\[ \text{reference } (\phi_4) \quad \Rightarrow \quad \text{unknown } \phi_3 \]

- new edge numbers \( k \) are determined by bit rotations of Gray numbering
- edge direction changes: \( e_{4,2} = -e_{2,4} \)
  \[ \Rightarrow \text{sign change wrt. precomputed } -T_M(0,4) := e_{1,2}^T M e_{2,4} \]
### Rotation: numbers and signs

<table>
<thead>
<tr>
<th>$\phi$</th>
<th>Edges</th>
<th>Sign</th>
<th>Bit representation of edge and sign</th>
<th>Shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_4$</td>
<td>5,1,2</td>
<td>+,+,+</td>
<td>1100 0000 0101 0000 0110 0000</td>
<td></td>
</tr>
<tr>
<td>$\phi_3$</td>
<td>2,4,0</td>
<td>+,-,+</td>
<td>0110 0000 1010 1000 0011 0000</td>
<td>$\gg$ 1</td>
</tr>
<tr>
<td>$\phi_2$</td>
<td>0,1,3</td>
<td>+,-,-</td>
<td>0011 0000 0101 0100 1001 1000</td>
<td>$\gg$ 2</td>
</tr>
<tr>
<td>$\phi_1$</td>
<td>3,4,5</td>
<td>-,+,+</td>
<td>1001 1000 1010 1010 1100 1100</td>
<td>$\gg$ 3</td>
</tr>
</tbody>
</table>

- Reference configuration with $\phi_4$ and edge numbers in Gray-code:
  
  $k_0 0000 \ k_1 0000 \ k_2 0000$

- Gray code edge numbers: rotates the 4 bits to the right
  4-bit **ror** [X86 Assembly]

- Number of carry bits odd $\implies$ sign change for an edge

- Realized with 16 bit number $k_1 k_1 00000000$
only the 18 precomputed values in $T_M$ are needed for each tetrahedron (and only 6 of them are really used)

- mapping problem is solved without branching
- rotation of edge numbers is solved without branching
- sign changes are solved without branching
- better memory access pattern
- slightly accelerated code.

Is there a chance for precomputing less than 18 values per tetrahedron?
A triangle is uniquely defined
- by its three edges \((e_{1,2}, e_{1,3}, e_{2,3})\), or
- by two edges and their enclosed angle \((e_{1,2}, e_{1,3}, \alpha)\).

With our \(M\)-inner product we can express
- \(\alpha\) via \(\langle e_{1,2}, e_{1,3} \rangle_M \equiv (e_{1,2})^T M e_{1,3} \), and
- an edge length as \(\| e_{1,2} \|_M^2 \equiv (e_{1,2})^T M (e_{1,2})\)

which allows to substitute the mixed inner product \(e_{1,2}^T M e_{1,3}\):

\[
e_{2,3} = x_3 - x_2 = x_3 - x_1 + x_1 - x_2 = e_{1,3} - e_{1,2}
\]

\[
e_{2,3}^T M e_{2,3} = (e_{1,3} - e_{1,2})^T M (e_{1,3} - e_{1,2})
\]

\[
= e_{1,3}^T M e_{1,3} + e_{1,2}^T M e_{1,2} - 2 e_{1,2}^T M e_{1,3}
\]

\[
e_{1,2}^T M e_{1,3} = \frac{1}{2} (e_{1,3}^T M e_{1,3} + e_{1,2}^T M e_{1,2} - e_{2,3}^T M e_{2,3}) \quad (\text{8})
\]
Reduced memory footprint

Thanks to (8) we have to precalculate/store only the 6 main diagonal entries of $T_M$ and calculate the remaining mixed inner $M$-products on demand.

- Only 6 floats per tetrahedron.
- Much less memory transfer.
- The sign change of the mixed products after rotation is the same as before.
- No additional branching.

Remaining entries of $T_M$:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_{1,2}^T M e_{1,2}$</td>
<td>$e_{1,3}^T M e_{1,3}$</td>
<td>$e_{2,3}^T M e_{2,3}$</td>
<td>$e_{1,4}^T M e_{1,4}$</td>
<td>$e_{2,4}^T M e_{2,4}$</td>
<td>$e_{3,4}^T M e_{3,4}$</td>
<td></td>
</tr>
</tbody>
</table>
Results on the workstation

- Intel Core i7-4700MQ CPU @2.40GHz x 4 + HT
- GeForce GTX 1080

**Table:** Run times in sec. on the workstation.

<table>
<thead>
<tr>
<th>Implementations</th>
<th># Tests</th>
<th>CUDA 8 threads</th>
<th>OpenMP 8 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without Gray-code</td>
<td>3,073,529</td>
<td>1.49</td>
<td>5.66</td>
</tr>
<tr>
<td>With Gray-code</td>
<td>3,073,529</td>
<td>0.73</td>
<td>3.65</td>
</tr>
<tr>
<td>Without Gray-code</td>
<td>24,400,999</td>
<td>11.48</td>
<td>56.63</td>
</tr>
<tr>
<td>With Gray-code</td>
<td>24,400,999</td>
<td>5.16</td>
<td>36.43</td>
</tr>
</tbody>
</table>

- CPU implementation: 33% faster
- CUDA implementation: 50% faster.
Results on mobile device

- Nvidia Shield tablet
- 2.2 GHz quad ARM Cortex A15 CPU and 2 GB RAM
- Tegra K1 with 192 core Kepler GPU

Table: Run times on the Shield Tablet (Android).

<table>
<thead>
<tr>
<th>Implementations</th>
<th># Tetrahedrons</th>
<th>CUDA sec.</th>
<th>OpenMP sec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without Gray-code</td>
<td>3,073,529</td>
<td>30.26</td>
<td>205.22</td>
</tr>
<tr>
<td>With Gray-code</td>
<td>3,073,529</td>
<td>20.30</td>
<td>97.19</td>
</tr>
</tbody>
</table>

- CPU implementation: 52% faster
- CUDA implementation: 33% faster.

Embedded systems as this tablet with limited memory and registers benefit more from the lower memory footprint.
We achieved an acceleration by 35% for the OpenMP implementation and by 50% for the CUDA implementation.

A profiling results show that the number of loads has been reduced to 70% and the number of stores dropped to 40% in the Gray-code version.

The last level cache (LLC) miss count has been reduced to 84%.

The significant reduction in stores is caused by avoiding many local temporary variables from the original approach.
High local memory overhead which accounted for 54% of total memory traffic which indicated excessive register spilling disappeared totally.

We have only 60 bytes spill stores and 84 bytes spill loads in the new main kernel in contrast to the old kernel with 352 bytes spill stores and 472 bytes spill loads.

The L2-bandwidth is doubled and achieves now 952.08 GB/s, compared to the version with only 456.103 GB/s.

The number of loads and stores is also decreased for the L2 cache as well as for the device memory increasing the memory bandwidth to above 90% compared to below 60% which indicating latency issues.

The divergent execution dropped and improved the warp execution efficiency from 75% to 80%.
Conclusions on memory reduction

- Significant reduction of the memory footprint.
- Less memory transfer + increased computations.
- Branching avoided.
- \( \Rightarrow \) 33% - 52% faster.
- weak hardware \( \Rightarrow \) better acceleration
- It can be adopted to any problem of that kind.

Future:: DD approach for mult. GPUs
Future: OpenACC
Future: adjoint problem
Figure: Computational domain $\Omega$ and non-overlapping sub-domains $\Omega_i$.

#subdomains $\gg$ #GPU SMs
dynamic mapping of active subdomains to SMs.
many kernels
- mult. separate kernels: scan, scatter, comp., ...
- one subdomain → mult. thread blocks
- host synchronizations after each kernel
- **less** shared memory needed per thread block
- one data buffer

one kernel
- one **large** kernel incl. scan, scatter, comp., ...
- one subdomain ↔ one thread block
- one host synchronization
- reaches shared memory limitations per thread block
- dynamic memory allocation in kernel

**Figure:** SMs allocated by GTX1080 to solve on 3 active domains (mult. threads blocks per domain)

**Figure:** SMs allocated by GTX1080 to solve on 3 active domains (one thread block per domain).
DD results: coarse mesh

Eikonal on GTX 1080 with 3 Mill. tets

- mult. kernels
- one kernel

Graph showing the comparison of execution time (Sec.) with the number of subdomains. The graph indicates a decrease in execution time as the number of subdomains increases, with 'one kernel' performing better than 'mult. kernels'.
DD results: fine mesh

Eikonal on GTX 1080 with 24 Mill. tets

- mult. kernels
- one kernel

Graph showing the performance of Eikonal on GTX 1080 with 24 million tetrahedra, comparing single and multiple kernel approaches.

D. Ganellari [Graz → ETH Zürich]
DD conclusions

- DD on one GPU
- **faster** than without DD (CUB Scan in shared mem)

- **one kernel**: better scaling for small example
  (no global memory limitation)
- larger #subdomains $\Rightarrow$ better load balancing
- poor scaling for large example
  (not enough GPU memory $\Rightarrow$ many dynamic allc/dealloc)
  $\Rightarrow$ next step: clusters of GPUs
- **one kernel**: increased granularity,
  take care for coalesc data access!
- Similar behavior on Titan X (8 GB $\rightarrow$ 12 GB), 20% faster.
- **one kernel**: much better code structure
A fast iterative method for eikonal equations.

A fast iterative method for solving the eikonal equation on triangulated surfaces.

Zhisong Fu, Robert M. Kirby, and Ross T. Whitaker (2013)
Fast iterative method for solving the eikonal equation on tetrahedral domains.

Fast sweeping methods for eikonal equations on triangulated meshes.

A massively parallel Eikonal solver on unstructured meshes.
Computing and Visualization in Science, accepted.
This work has been supported by

- FWF project F32-N18,
- Erasmus Mundus JoinEUsee PENTA scholarship
- NVIDIA hardware donation
The Bigger Picture: ILearnHeart

GOAL
Patient-specific model of total heart function
Physics involved:
- Electrophysiology
- Mechanics
- Fluid Dynamics

Code basis:
http://getdrawings.com/

Cardiac Arrhythmia Research Package
J. Fuchsberger (Graz), E. Karabelas (King's College)

http://getdrawings.com/
GOAL
Patient-specific model of total heart function
The Bigger Picture: ILearnHeart

GOAL
Patient-specific model of total heart function

Physics involved:
- Electrophysiology
- Mechanics
- Fluid Dynamics

http://getdrawings.com/
GOAL
Patient-specific model of total heart function

Physics involved:
- Electrophysiology
- Mechanics
- Fluid Dynamics

Code basis:
Cardiac Arrhythmia Research Package

http://getdrawings.com/
Objective

GOAL
Model the impact of a valve in the human heart

\[^{1}\text{Khadra2000.}\]
Objective

GOAL
Model the impact of a valve in the human heart

ISSUES
- complex multiphysics problem
- interface conditions
- projecting boundary data from one domain into the other
- costly in terms of computation time

\[ ^1 \text{Khadra2000.} \]
GOAL
Model the impact of a valve in the human heart

ISSUES
- complex multiphysics problem
- interface conditions
- projecting boundary data from one domain into the other
- costly in terms of computation time

IDEA
Penalize the Navier-Stokes equations to model obstacles in the fluid domain (fictitious domain approach\(^1\))

\(^1\)Khadra2000.
BENEFITS

- flexible and robust computational technique
- straight forward implementation
- saving computation time
- different media are accounted for by their physical characteristics assigned to the grid points
- time-dependent geometry and material properties can easily be taken into account
Navier-Stokes-Brinkman (NSB) Model

Domain and Boundary:
\[ \Omega = \Omega_f \cup \Omega_p \cup \Omega_s \]
\[ \Gamma = \Gamma_e \cup \Gamma_n \]  \hspace{1cm} (3)

Navier-Stokes-Brinkmann Model:
\[
\begin{aligned}
\partial_t (\rho u_i) + \partial_j (\rho u_i u_j - \sigma_{ij}) + \frac{\mu}{K} u_i &= \rho f_i & \text{in } \mathbb{R}^+ \times \Omega \\
\partial_i u_i &= 0, & \text{in } \mathbb{R}^+ \times \Omega \\
\u_i(0, \cdot) &= u_i^0 & \text{in } \Omega \\
\u_i &= g_i, & \text{on } \Gamma_e \\
\sigma_{ij} n_j &= h_i, & \text{on } \Gamma_n
\end{aligned}
\]  \hspace{1cm} (4)

Stress Tensor \( \sigma_{ij} \):
\[
\sigma_{ij} = -\rho \delta_{ij} + 2\mu \varepsilon_{ij}
\]
\[
\varepsilon_{ij} = \frac{1}{2} (\partial_i u_j + \partial_j u_i)
\]  \hspace{1cm} (5)
The Permeability $K$

\[ K(t, x_i) = \begin{cases} 
K_f \to +\infty, & \forall x_i \in \Omega_f \\
K_p, & \forall x_i \in \Omega_p \\
K_s \to 0^+, & \forall x_i \in \Omega_s 
\end{cases} \quad (6) \]

- $\Omega_f$: $\frac{\mu}{K}u_i \to 0$ classical Navier–Stokes equations
- $\Omega_p$: full Navier-Stokes-Brinkmann equations
- $\Omega_s$: $\frac{\mu}{K}u_i$ forces $u_i \to 0$ Darcy’s law (7)

\[ \partial_i p + \frac{\mu}{K}u_i = \rho f_i \quad (7) \]

Note: No-slip condition on $\partial \Omega_s$ is asymptotically satisfied
Variational Arbitrary Lagrangian Eulerian (ALE) Formulation

The Navier-Stokes-Brinkman Model for a moving domain:

Find $u_i \in S_u$ and $p \in S_p$, such that $\forall w_i \in V_u$ and $q \in V_p$:

$$
\int_{\Omega} w_i \rho \left( \partial_t u_i + (\partial_j u_i) (u_j - \hat{u}_j) - f_i + \frac{\nu}{K} u_i \right) d\Omega \\
+ \int_{\Omega} \varepsilon_{ij}(w_i) \sigma_{ij}(u_i, p) d\Omega - \int_{\Gamma_n} w_i h_i d\Gamma + \int_{\Omega} q \partial_i u_i d\Omega = 0
$$

(8)

$u_i$ ... fluid velocity

$\hat{u}_i$ ... fluid domain velocity respective to a reference domain
Residual-Based Variational Multiscale Formulation

RBVMS\(^2\) formulation of the ALE NSB equations:

Decomposition into coarse (\(h\)) and fine (\('\)) scale subspaces:

\[
\begin{align*}
S_u &= S_u^h \oplus S_u' \\
S_p &= S_p^h \oplus S_p' \\
\mathcal{V}_u &= \mathcal{V}_u^h \oplus \mathcal{V}_u' \\
\mathcal{V}_p &= \mathcal{V}_p^h \oplus \mathcal{V}_p'
\end{align*}
\] (9)

Model the fine scale velocity and the fine scale pressure:

\[
\begin{align*}
\vec{u}'_i &= -\frac{\tau_{SUPS}}{\rho} r_i^M(u_i^h, p^h) \\
p' &= -\rho \nu_{LSIC} r_i^C(u_i^h)
\end{align*}
\] (10)

Residuals \(r_i^M\) and \(r_i^C\):

\[
\begin{align*}
 r_i^M(u_i^h, p^h) &= \rho \left( \partial_t u_i^h + (\partial_j u_i^h) \left( u_j^h - \hat{u}_j^h \right) - f_i^h + \frac{\nu}{K} u_i^h \right) - \partial_j \sigma_{ij}(u_i^h, p^h) \\
r_i^C(u_i^h) &= \partial_i u_i^h
\end{align*}
\] (11)

\(^2\)Bazilevs.
Find \( u^h_i \in S^h_u \) and \( p^h \in S^h_p \), such that \( \forall w^h_i \in \mathcal{V}^h_u \) and \( q^h \in \mathcal{V}^h_p \):

\[
\int_{\Omega} w^h_i \rho \left( \partial_t u^h_i + (\partial_j u^h_i) \left( u^h_j - \hat{u}^h_j \right) - f^h_i + \frac{v}{K} u^h_i \right) d\Omega + \int_{\Omega} \varepsilon_{ij}(w^h_i) \sigma_{ij}(u^h_i, p^h) d\Omega \\
- \int_{\Gamma_n} w^h_i h^h_i d\Gamma + \int_{\Omega_t} q^h \partial_i u^h_i d\Omega \\
+ \sum_{e=1}^{n_{el}} \int_{\Omega_e} \tau_{SUPS} \left( (\partial_j w^h_i) \left( u^h_j - \hat{u}^h_j \right) + \frac{\partial_i q^h}{\rho} \left( - \frac{v}{K} w^h_i \right) \right) r^M_i(u^h_i, p^h) d\Omega \\
+ \sum_{e=1}^{n_{el}} \int_{\Omega_e} \rho \nu_{LSIC} \partial_i w^h_i r^C(u^h_i) d\Omega \\
- \sum_{e=1}^{n_{el}} \int_{\Omega_e} \tau_{SUPS} w^h_i \left( r^M_i(u^h_i, p^h) (\partial_i u^h_i) \right) d\Omega \\
- \sum_{e=1}^{n_{el}} \int_{\Omega_e} \frac{\partial_j w^h_i}{\rho} \left( \tau_{SUPS} r^M_i(u^h_i, p^h) \right) \left( \tau_{SUPS} r^M_j(u^h_i, p^h) \right) d\Omega = 0
\]  

(12)
How to choose the permeability parameters to model the valve?
How to choose the permeability parameters to model the valve?

- NS-domain
- NS-Brinkmann
- Darcy
Moving Valve
Moving Valve

Moving Valve + Moving Mesh
See also heart pump!
Moving valve: Conclusions

PRO

- suitable to model the impact of the valve
- straightforward implementation
- robust and fast (on 128 cores)

CONTRA

- two discretized domains (fluid; valve)
  - intersection between elements $\Rightarrow$ volume fraction,
  - interpolate (RBF, simple linear?) pre-described boundary moving to the interior of valve.
- modelling error in the direct vicinity of the valve
  - use finer discretization in the small subdomain around the valve.

Financially supported by BioTechMed Graz and NAWI Graz.
Reinforcement Learning for a HEV System Controller
Manfred Liebmann, Uni Graz

GOAL
Design a hybrid electric vehicle (HEV) system controller using reinforcement learning (RL).

Figure: Left: Hybrid electric vehicle. Right: Agent-environment feedback loop of reinforcement learning

- **Agent**: System controller (Deep neural network)
- **Environment**: Hybrid electric vehicle model (AVL CRUISE software)
- Optimize: Fuel economy, performance, emissions, or drivability

Project supported by FVV (Frankfurt) and AVL List GmbH (Graz)
Figure: Zotter chocolate factory (Bean To Bar) near Graz