Going Multi-Physics

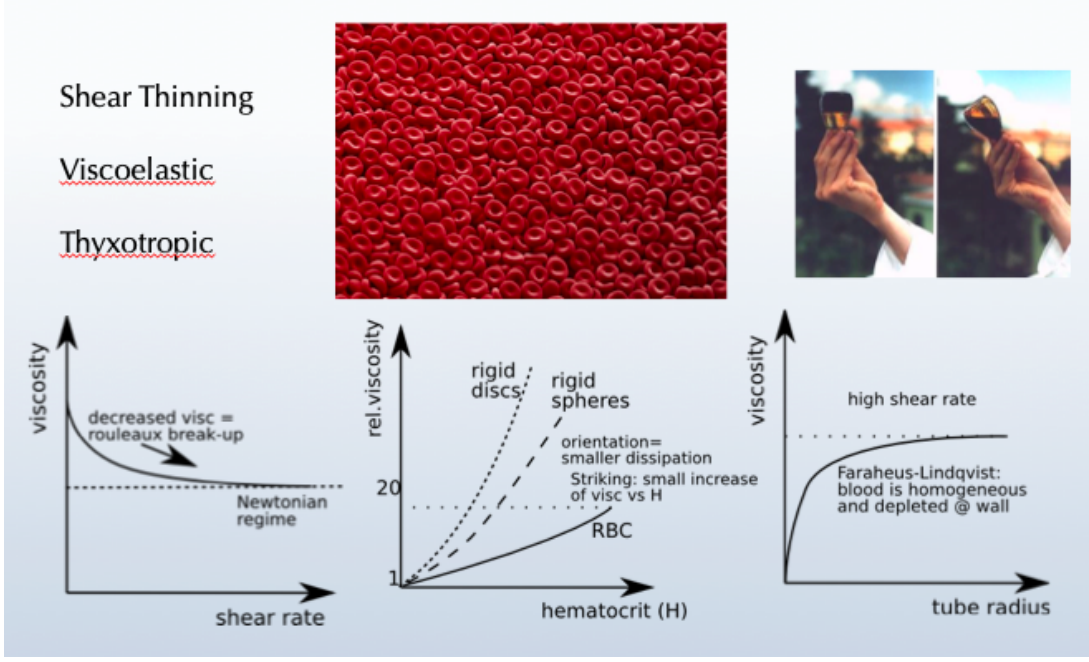
AC290r - Simone Melchionna

National Research Council Italy & Lexma Technology LLC

April 18, 2019

Hemorheology (recap)

Besides blood, so many fluids show non-Newtonian behavior (paints, suspensions, biofluids, emulsions,....)

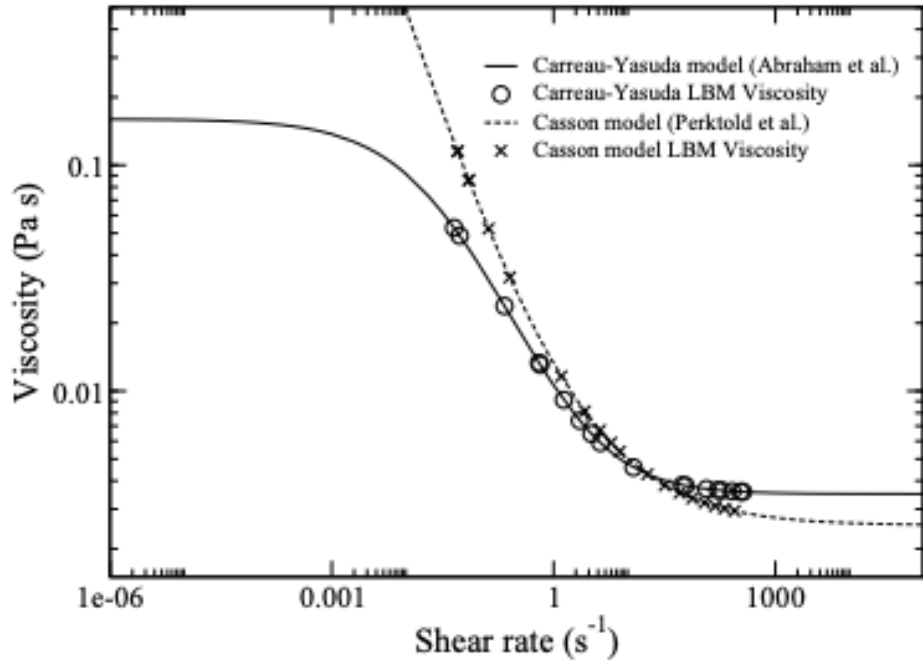


Can we get away with a simplified model for blood rheology?

Non-Newtonian models

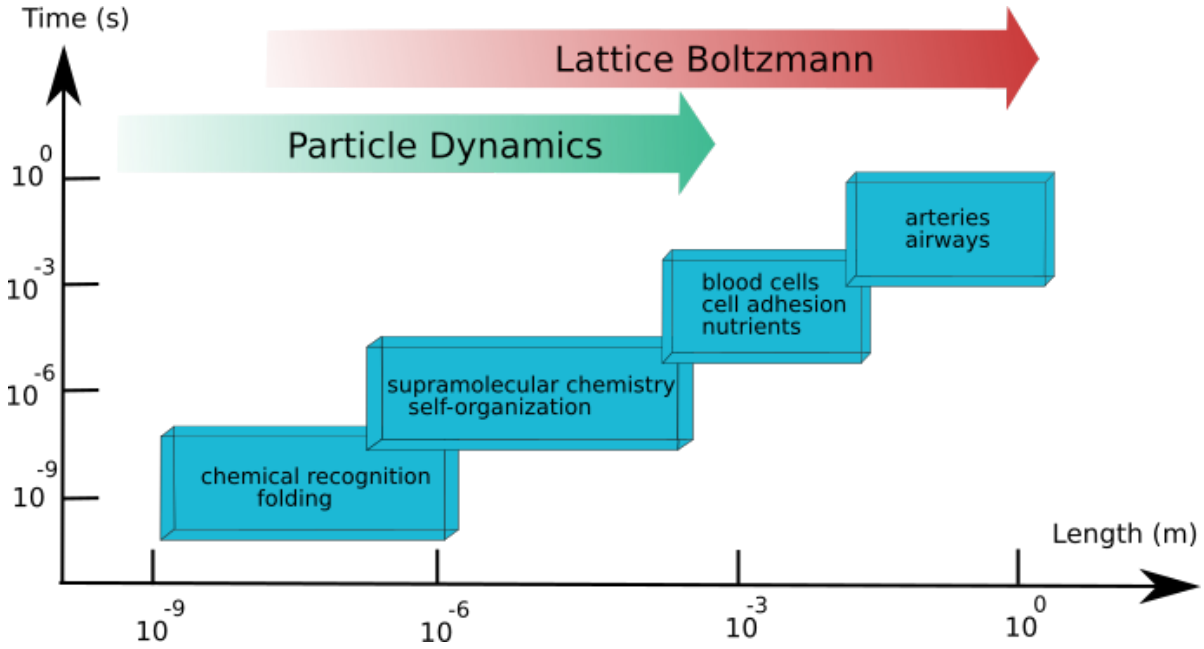
To reproduce shear thinning/ thickening, etc, we don't necessarily need to represent multiple elements (eg plasma + cells)
Stress tensor: $\sigma = -p\mathbf{I} + 2\rho\nu\mathbf{S}$
Strain rate tensor: $\mathbf{S} = \frac{1}{2}\left(\nabla u + \nabla u^T\right)$ and the strain rate $\dot{\gamma} = 2\sqrt{\mathbf{S} : \mathbf{S}}$
which can be computed directly from LBM :
$\mathbf{S} = -\frac{3\omega}{2}\sum_p \mathbf{c}_p\mathbf{c}_p(f_p - f_p^{eq})$

To reproduce a non-Newtonian rheology, it is possible to use different analytical forms $\nu(\dot{\gamma})$, from models present in the literature (Casson, Carreau-Yasuda, etc.)



Multi-Physics

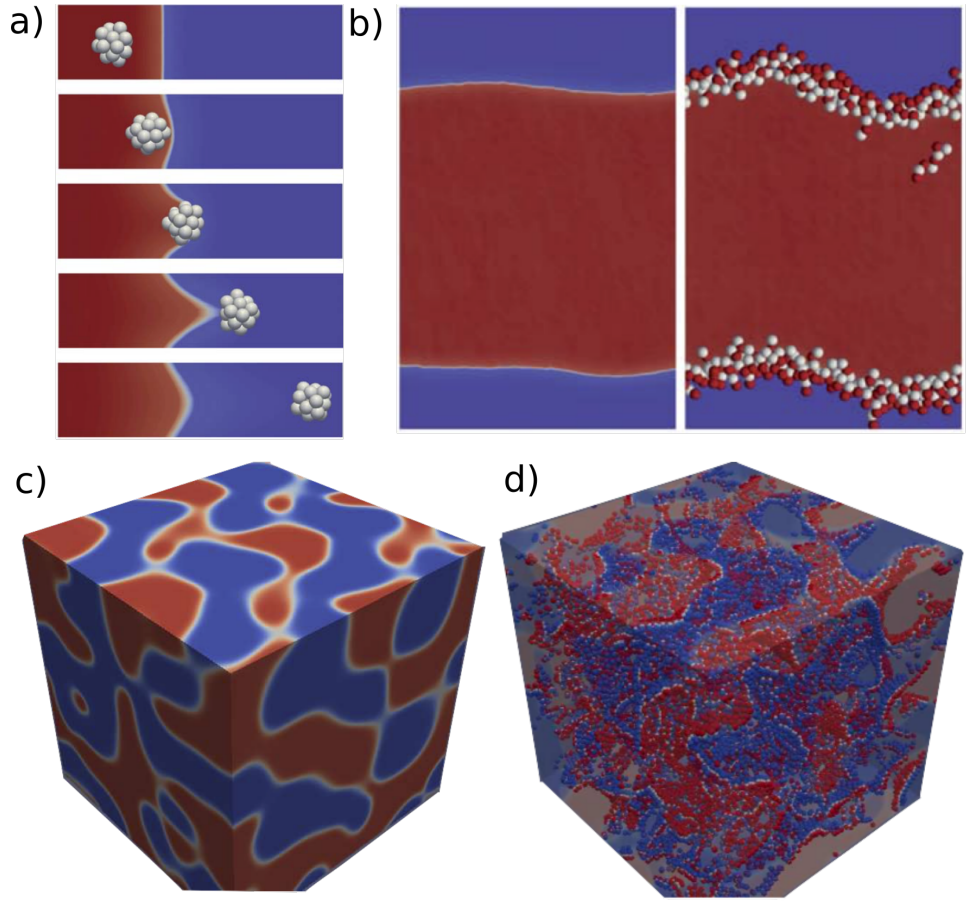
But if the goal is to tackle complex problems where one single picture is insufficient, a more flexible representation is needed.



where:

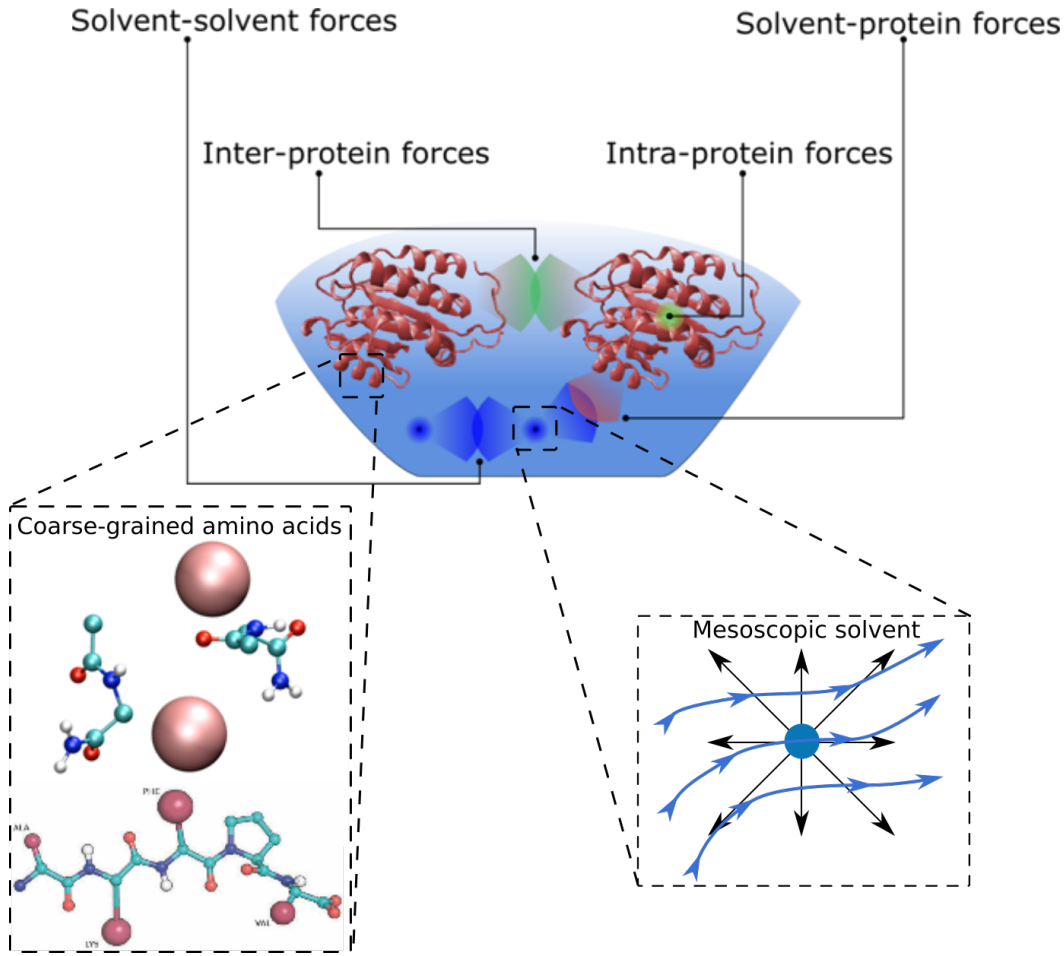
- Continuum pictures can hold and coexist with the particle-like picture
- Actors can be evolved on a common timescale and a common "resolution"

An example: bigels



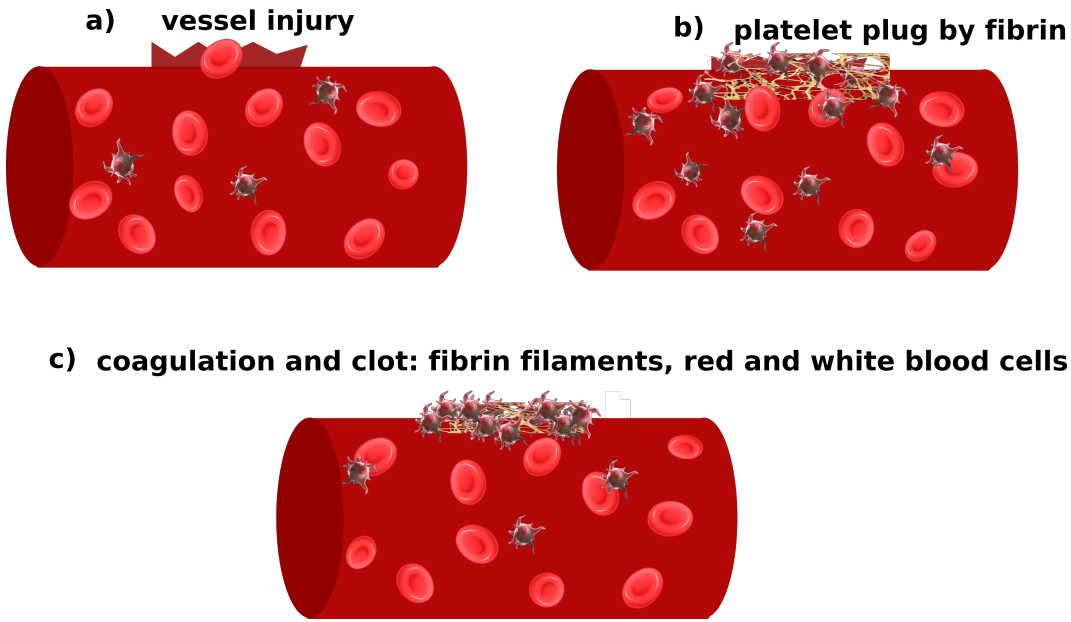
A multiphase fluid in presence of amphiphiles that sit at the interface and actually modify its interfacial properties (eg surface tension)

An example: proteins



Macromolecules in solution can be handled as mesoscopic objects, with no need to reproduce all degrees of freedom in detail

An example: hemostasis



Hemostasis arises from the interplay of various blood components. By neglecting the biochemical cascade, the process can be modelled at a single resolution.

Advection-Diffusion-Reaction with LBM

$$\partial_t \phi + \nabla \cdot \mathbf{u}\phi = \nabla(D\nabla\phi) + S(\phi)$$

$$f_p(x + hc_p, t + h) = f_p(x, t) + \omega h \left[f_p^{eq}(\phi, \mathbf{u}) - f_p \right](x, t) + w_p \frac{\mathbf{c}_p \cdot \mathbf{g}}{c_s^2}$$

$$f_p^{eq}(\phi, \mathbf{u}) = w_p \phi \left[1 + \frac{\mathbf{u} \cdot \mathbf{c}_p}{c_S^2} \right]$$

with $\mathbf{u}$ imposed externally: the advector.

$$D = c_s^2 \left(\frac{1}{\omega} - \frac{1}{2} \right)$$

Boundary Conditions (recap)

In general boundary conditions fall under the following generic condition: $b_1 \frac{\partial \phi}{\partial n}\Big|_{x_b,t} + b_2 \phi(x_b, t) = b_3$ with ϕ density or velocity, n the wall normal, b_1, b_2, b_3 constants.

Choosing $\phi = \mathbf{u} = 0$ is the no-slip boundary conditions for impenetrable walls.
Dirichlet BC : $b_1 = 0$ and $b_2 \neq 0$
Neumann BC : $b_1 \neq 0$ and $b_2 = 0$ (imposing the flux)

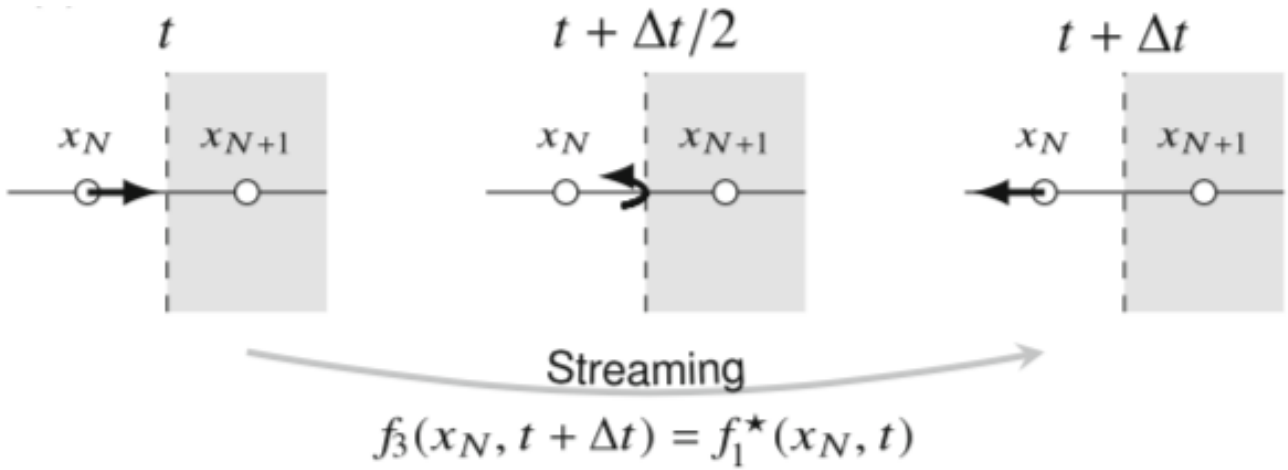
Boundary Conditions: Dirichlet

Dirichlet BC: at the wall

$$\phi = \phi_w$$

typically imposed via the "anti-bounce back" scheme:

$$f_{\bar{p}}(x_N, t+1) = f_p^*(x_N, t) + 2f_p^{eq}(x_N, t+1)$$



As for bounce-back, this is 2nd order accurate for aligned walls.

Boundary Conditions: Neumann

The flux is imposed from the walls (eg heat thermal coefficient, dielectric condition, etc.)

$$\hat{\mathbf{n}} \cdot \nabla \phi = \psi_w$$

The condition on gradient is transformed into a Dirichlet type for a scalar. By selecting $\hat{\mathbf{n}}$ to be aligned with a given discrete direction

$$\hat{\mathbf{n}} \cdot \nabla \phi \simeq \frac{\phi(x_{N+1}) - \phi(x_N)}{|x_{N+1} - x_N|} = \psi_w$$

with $(x_{N+1} - x_N)$ aligned with c_p, then the BC is imposed via the "anti-bounce back" scheme on f_p (generic directions are also possible but the scheme becomes fairly more involved)

Project:

How many fluids do we have ?
2: one for the advector (BGK) and one for the ADR.
In python terms
How to impose the "advector"
Do we need to impose any special condition on walls ?
If we don't do anything, diffusion will create a leak. If there is no mass exchange with the wall:

$$\hat{\mathbf{n}} \cdot \nabla c = 0$$

the usual bounce-back is already sufficient to create an impenetrable wall.

MUPHY script for ADR

```
from MagicUniverse import *
MagicBegins()

u = Universe()
f = Fluid()      # for carrier
c = Fluid()      # for drug
... other actors instantiated...

u.create()
...
f.setViscosity( ... )
...
c.setDiffusivity( .... )
...
u.decorate()
...
...time cycle
```

This will create a DIRDATA with the two actors (and in paraview only the scalar quantity will be shown on the ADR mesh).

File .ios for multiple fluid actors

Each fluid will have its own *.ios.

For default behavior, files are named in order:
bgkflag.1.ios —> first fluid actor (header + nodes)
bgkflag.2.ios —> second fluid actor - most likely, the drug specie (only contains the header).
Also, BC conditions can be changed from python:

f.setIOValue('inlet', 1, U) f.setIOValue('outlet', 2, U)
c.setIOValue('inlet', 1, C0) d.setIOValue('outlet', 2, C0)

In this way we can impose a given time-dependent singla, eg pulsatile signal at the inlet.

Imposing an initial profile

```
f = Fluid()      # for carrier
c = Fluid()      # for drug
... other actors instantiated...

u.create()
...
u.decorate()
...
nx, ny, nz = m.getBox()
nx = int(nx); ny = int(ny); nz = int(nz)

profile2 = c.getArray(nx*ny*nz)
for k in xrange(1,nz+1):
    for j in xrange(1,ny+1):
        for i in xrange(1,nx+1):
            ifl = m.getLocator(i,j,k)
            if k > nz/8 - 3 and k < nz/8. + 3: # create BOLUS
                profile2[ifl] = C1
            else:
                profile2[ifl] = C0
c.setDensityProfile(profile2)

for itime in u.cycle():
    ...
```

Freezing fluid actors

To release the drug, you want to first have stabilized fluid flow for the carrier.

```
f = Fluid()      # for carrier
c = Fluid()      # for drug
... other actors instantiated...

u.create()
...
f.setFreeze( False )
c.setFreeze( True )
...
u.decorate()
for itime in u.cycle():
    if itime == 1000:
        f.setFreeze( True )
        c.setFreeze( False )

    u.animate()
```

Regular Domain Decomposition in slabs

To use a parallel decomposition along the x,y,z directions, you can use any number of domains and thus, of parallel tasks.

```
m = Mesh()
... other actors instantiated...

u.create()
...
s.set( name='MyScale', mesh=m, actors=[f,c,t] )
m.setRegularMesh( True )
m.setPeriodicity( '111')
m.setDomainDecomposition( 3 )          # 1: z, 2: y, 3: x-wise domain decomposition
...
u.decorate()
```

Regular Domain Decomposition in parallelepipeds

You can use only a number of domains that is a perfect cube, e.g. 8, 27, 64, ..., 512, 1000 (not 1024)

```
m = Mesh()
... other actors instantiated...

u.create()
...
s.set( name='MyScale', mesh=m, actors=[f,c,t] )
m.setRegularMesh( True )
m.setPeriodicity( '111')
m.setDomainDecomposition( 7 )          # decompisition in parallelepipeds
...
u.decorate()
```

A note on visualization

- Showing ADR data is typically done by showing the underlying flow field (pressure or velocity) and the ADR quantity

Paraview ca be used in combination with:

- Intel OSPRay (photorealistic rendering)
- NVIDIA Index plugin (volume rendering on unstructured grids)