Performance optimizations in large-scale biomedical computations

Gábor Závodszky[1,2]

[1]University of Amsterdam, Computational Science Institute, Amsterdam, The Netherlands

[2]Budapest University of Technology and Economics, Department of Hydrodynamic Systems, Budapest, Hungary

Preface & outline

Disclaimer

- **Performance** can refer to several things – in this talk I simply mean **wall clock run-time performance**

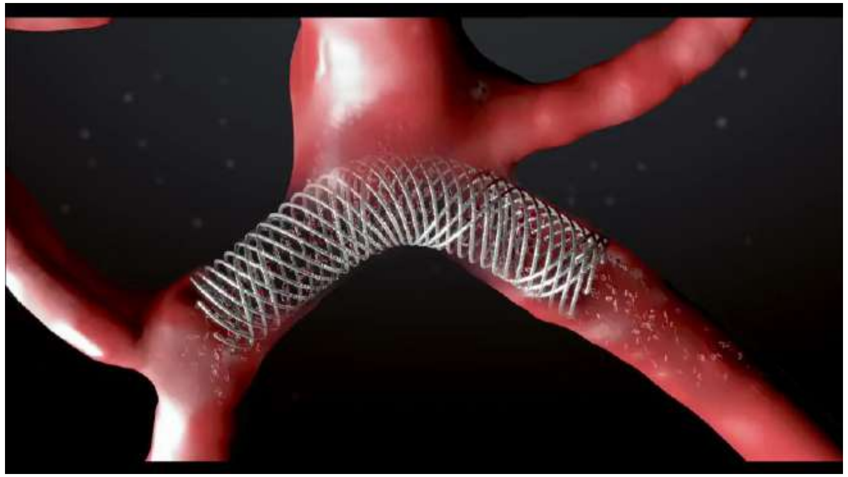
Outline

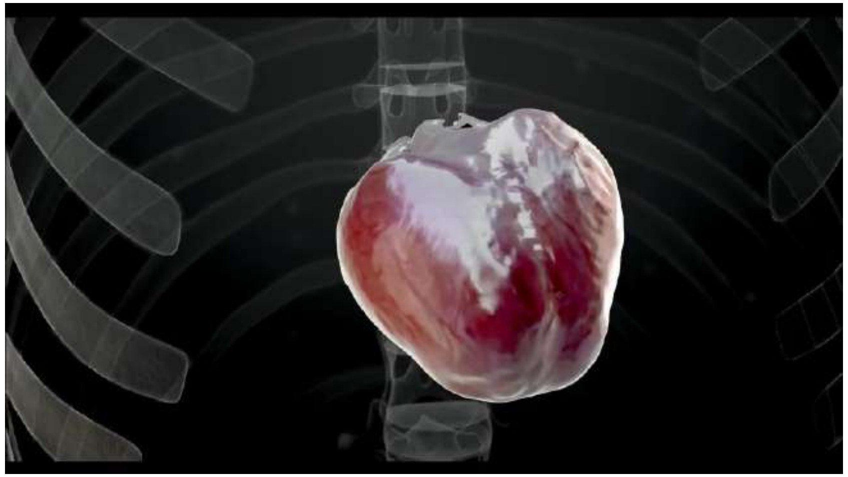
- Brief introduction to numerical models. What happens at large-scale deployments?
- Cherry-picking from the most often used techniques to improve computational performance.
  - These are just a few examples.
  - High-level overview.
  - From the perspective of a modeler.
- These techniques in the current talk are divided into three levels:
  - Low-level / architecture dependent programming techniques
  - Medium-level / algorithm implementation level techniques
  - High-level / conceptual-level considerations

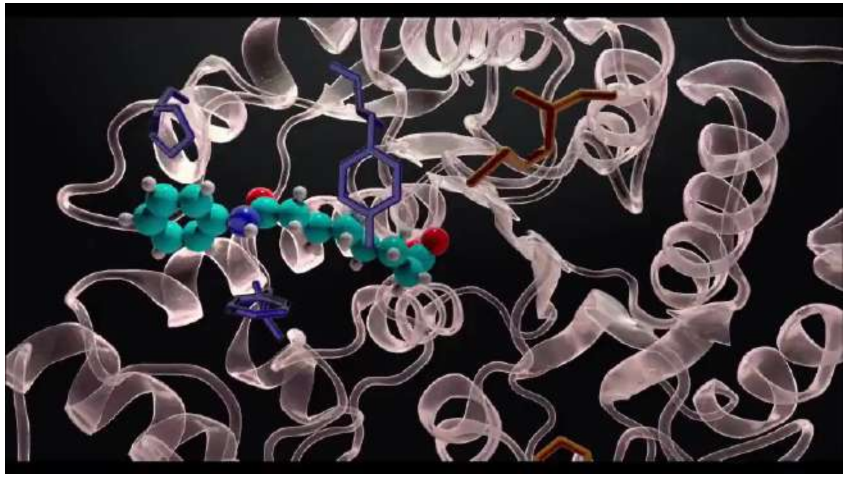
Virtual Physiological Human

https://youtu.be/1FvRSJ9W734

Some example models used in research

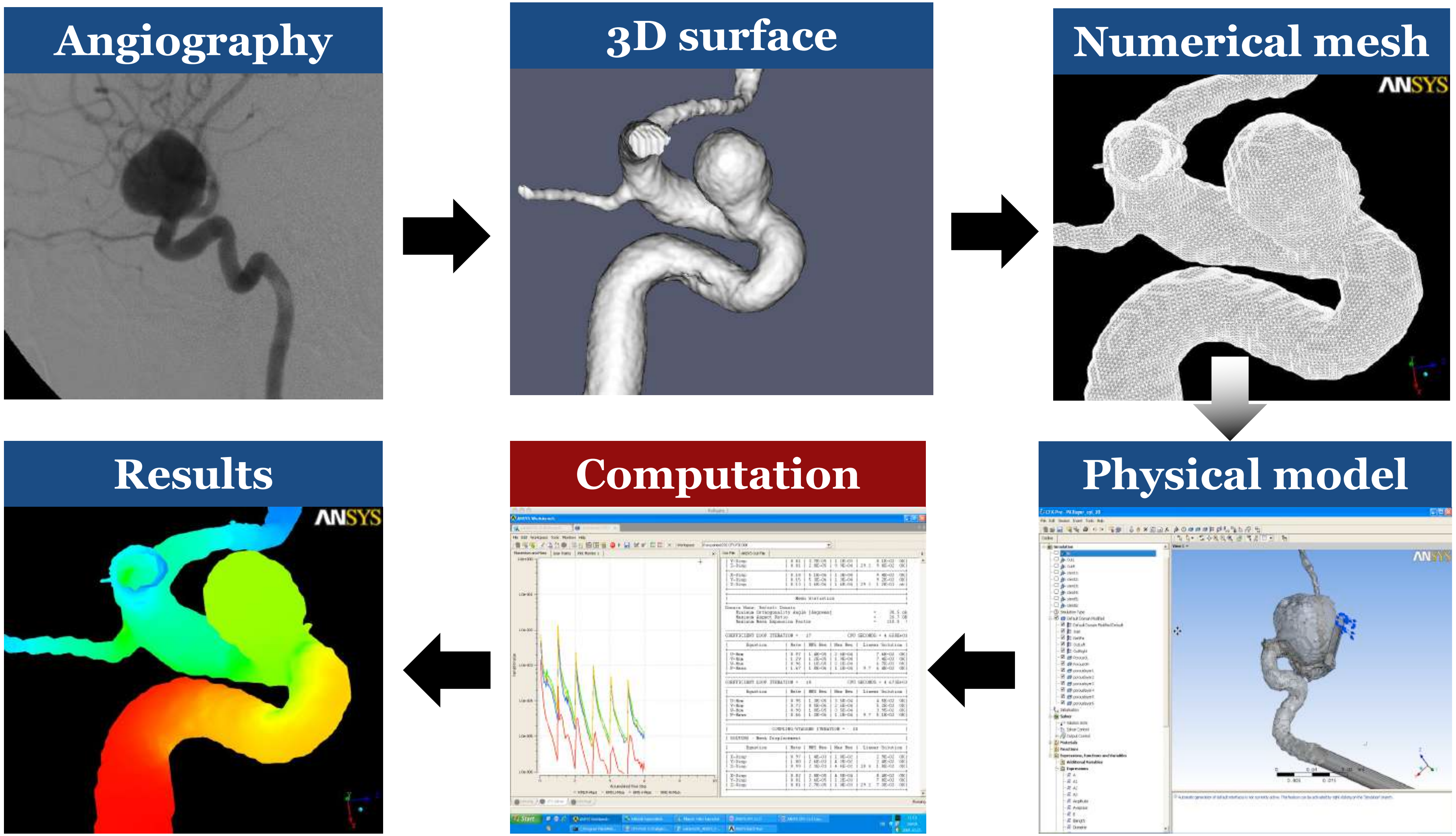






Intro – numerical models

Modelling workflow



Angiography

3D surface

Numerical mesh

Physical model

Computation

Results

Numerical methods

Physical system

$$\begin{cases} \rho\dfrac{\partial \boldsymbol{u}}{\partial t} + \rho(\boldsymbol{u}\cdot\nabla)\boldsymbol{u} - \nabla\cdot\boldsymbol{\sigma}(\boldsymbol{u},p) = \boldsymbol{f} \\ \nabla\cdot\boldsymbol{u}=0 \\ \boldsymbol{u}=\boldsymbol{g} \\ \sigma(\boldsymbol{u},p)\hat{\boldsymbol{n}} = \boldsymbol{h} \\ \boldsymbol{u}(0)=\boldsymbol{u}_0 \end{cases}$$

PDE

Analytic technique / approximation

ODE

$$\begin{pmatrix} y_1^{(n)} \\ y_2^{(n)} \\ \vdots \\ y_m^{(n)} \end{pmatrix} = \begin{pmatrix} f_1\left(x,\mathbf{y},\mathbf{y}',\mathbf{y}'',\ldots,\mathbf{y}^{(n-1)}\right) \\ f_2\left(x,\mathbf{y},\mathbf{y}',\mathbf{y}'',\ldots,\mathbf{y}^{(n-1)}\right) \\ \vdots \\ f_m\left(x,\mathbf{y},\mathbf{y}',\mathbf{y}'',\ldots,\mathbf{y}^{(n-1)}\right) \end{pmatrix}$$

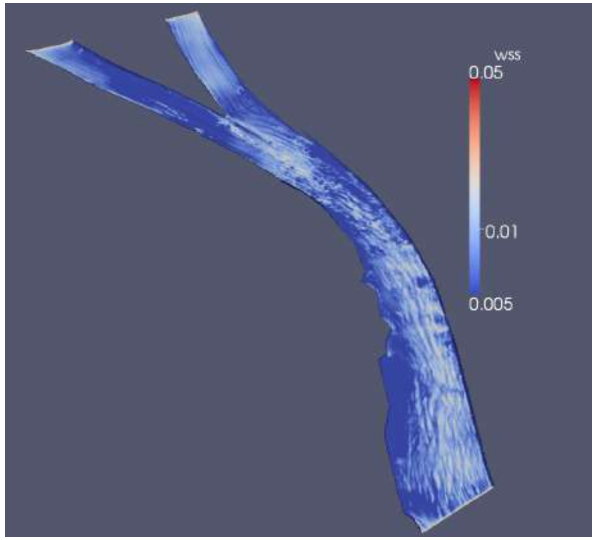
Numerical method

Algebraic

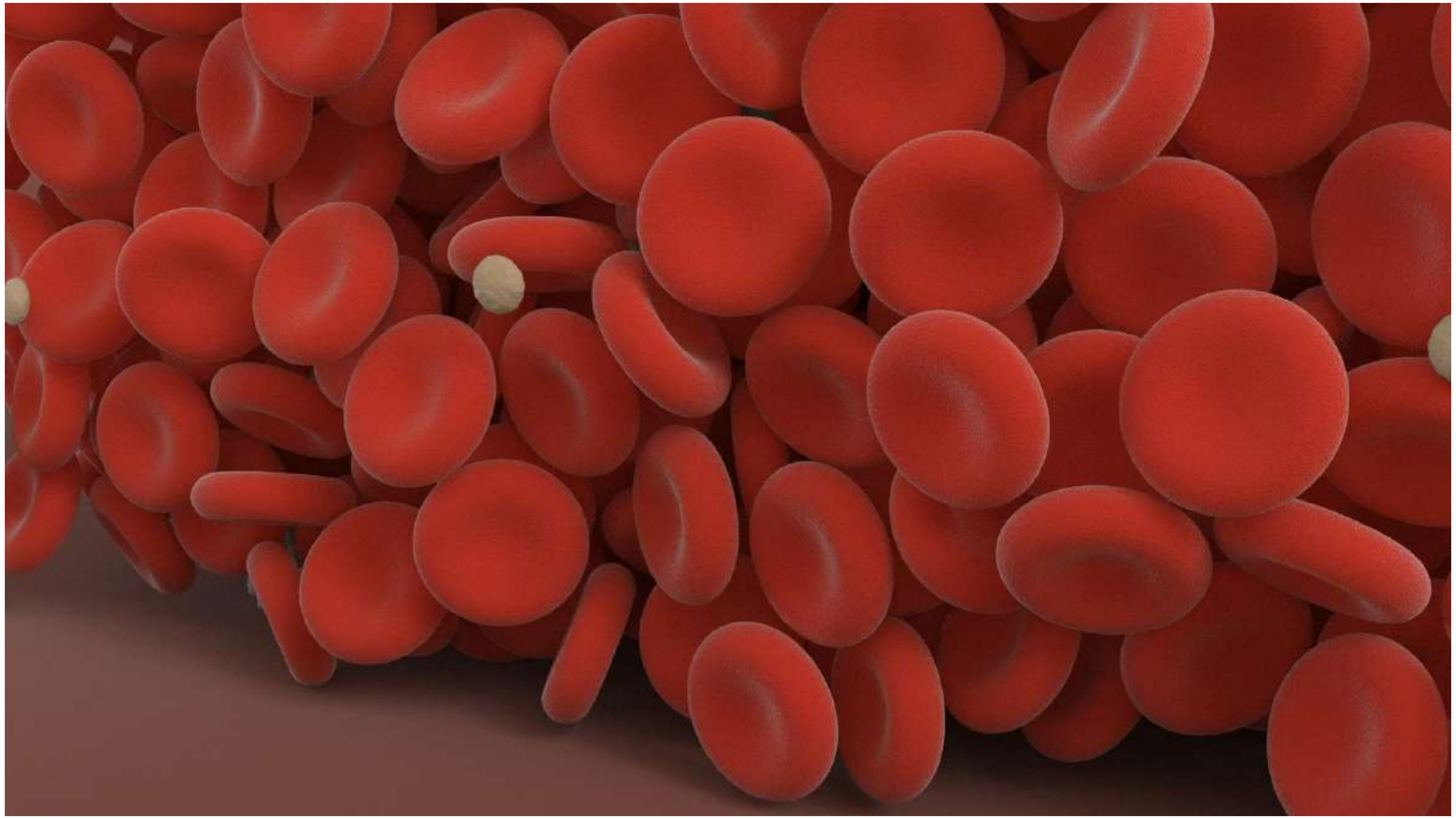
$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

where a_{ij} and b_i are constants, $i=1,2,\ldots,m, j=1,2,\ldots,n.$

Numerical solution



HemoCell



- Open source (www.hemocell.eu)
- Fully validated
- Currently used for research in:
    - Malaria
    - Diabetes
    - Hemolysis
    - Retinal aneurysms
    - Thrombus formation
    - Cellular and drug transport
    - Platelet aggregation

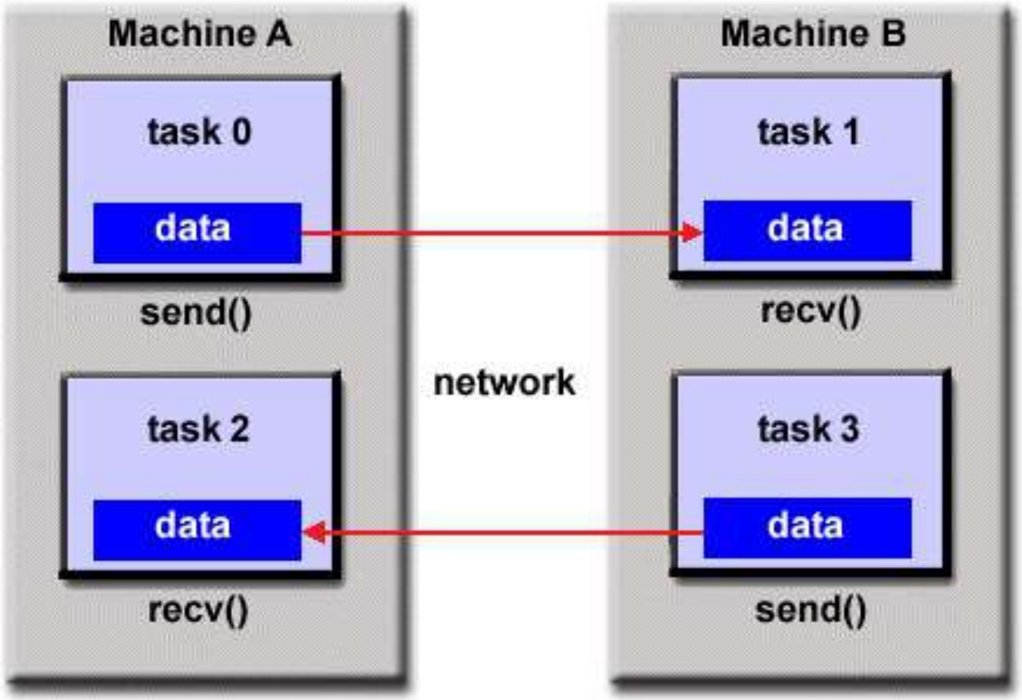
MPI and OpenMP

This model demonstrates the following characteristics:

- A set of tasks that use their own local memory during computation. Multiple tasks can reside on the same physical machine and/or across an arbitrary number of machines.

- Tasks exchange data through communications by sending and receiving messages.

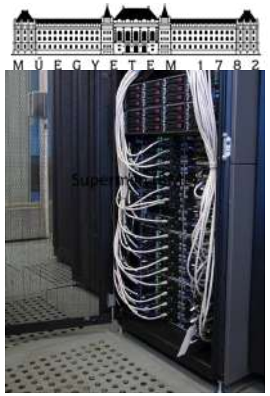
- Data transfer usually requires cooperative operations to be performed by each process. For example, a send operation must have a matching receive operation.



Barney, B. (2010). Introduction to parallel computing. Lawrence Livermore National Laboratory, 6(13), 10.

Note: in some cases shared memory parallelism can be preferable. I.e.: OpenMP can be more portable (GPU support), faster to implement (decorator over a loop).

HPC deployment



Superman (BME, Budapest)

Lomonosov (MSU, Moscow)

BSC
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Marenostrum (BSC, Barcelona)

SGI Altrix (QUT, Brisbane)

Eagle (PSNC, Poznan)

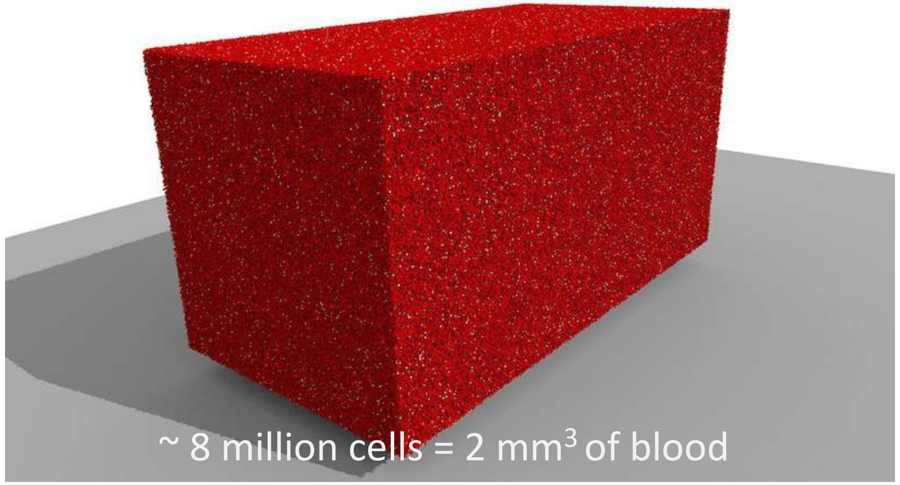
Cartesius and Lisa (SURFsara, Amsterdam)

Sanam (KACST)

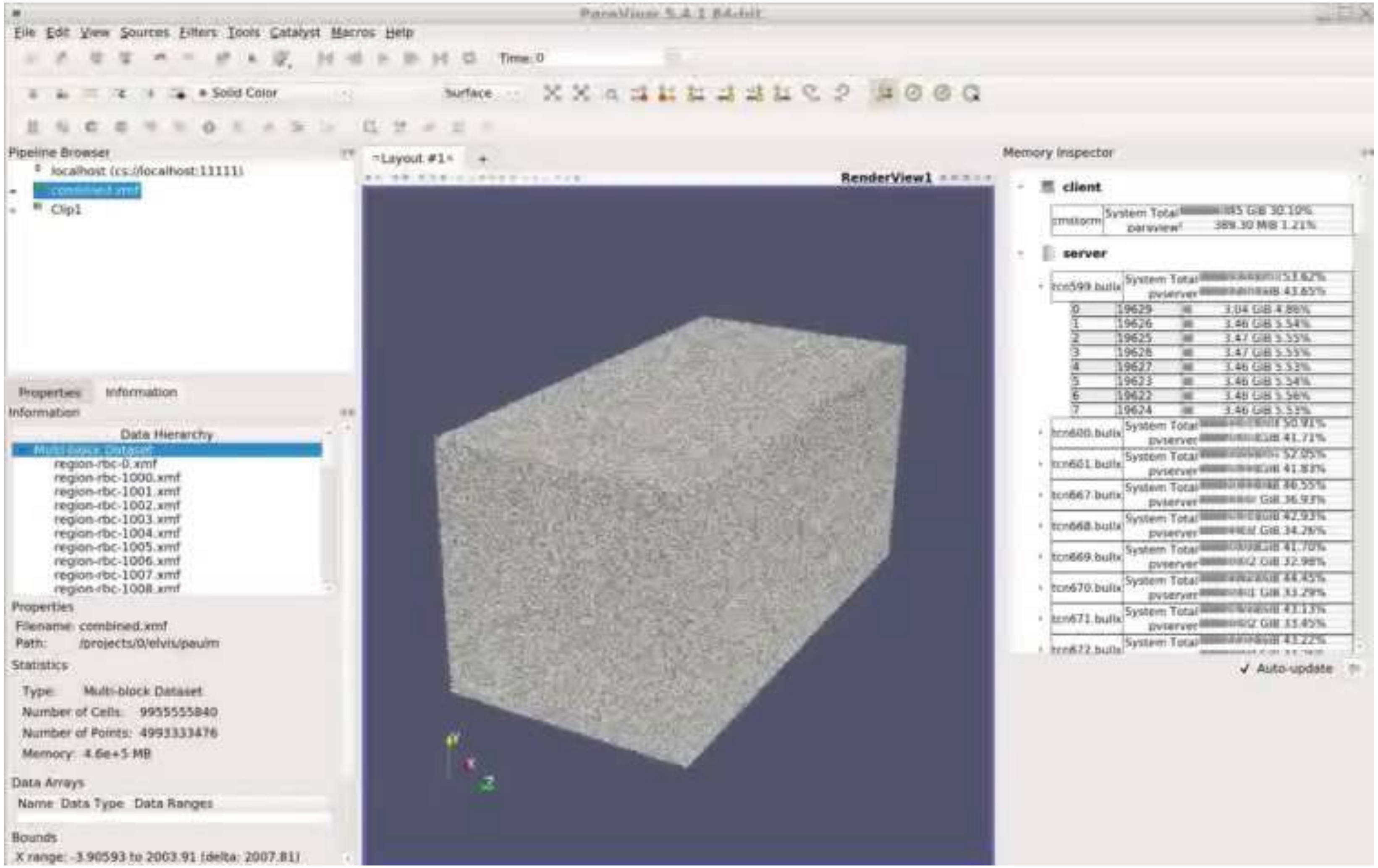
Supermuc (LRZ, Munich)

Aspire I (NSCC, Singapore)

HPC



~ 8 million cells = 2 mm^3 of blood



Pipeline Browser
- localhost (cs://localhost:11111)
- combined.xmf
- Clip1

Properties | Information

Information

Data Hierarchy
- Multi-block Dataset
  - region-rbc-0.xmf
  - region-rbc-1000.xmf
  - region-rbc-1001.xmf
  - region-rbc-1002.xmf
  - region-rbc-1003.xmf
  - region-rbc-1004.xmf
  - region-rbc-1005.xmf
  - region-rbc-1006.xmf
  - region-rbc-1007.xmf
  - region-rbc-1008.xmf

Properties

Filename: combined.xmf
Path: /projects/0/elvis/paulm

Statistics

Type: Multi-block Dataset
Number of Cells: 9955555840
Number of Points: 4993333476
Memory: 4.6e+5 MB

Data Arrays

Name Data Type Data Ranges

Bounds

X range: -3.90593 to 2003.91 (delta: 2007.81)

Memory Inspector

client

cmstorm	System Total	[illegible]5 GiB 30.10%
	paraview	389.30 MiB 1.21%

server

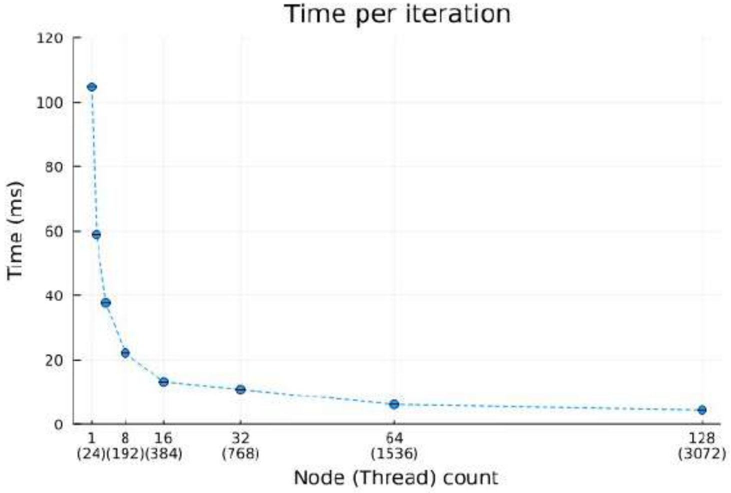
tcn599.bullx	System Total		[illegible] 53.62%
	pvserver		[illegible] GiB 43.65%
0	19629		3.04 GiB 4.86%
1	19626		3.46 GiB 5.54%
2	19625		3.47 GiB 5.55%
3	19628		3.47 GiB 5.55%
4	19627		3.46 GiB 5.53%
5	19623		3.46 GiB 5.54%
6	19622		3.48 GiB 5.56%
7	19624		3.46 GiB 5.53%
tcn600.bullx	System Total		[illegible] 50.91%
	pvserver		[illegible] GiB 41.71%
tcn601.bullx	System Total		[illegible] 52.05%
	pvserver		[illegible] GiB 41.83%
tcn667.bullx	System Total		[illegible] GiB 46.55%
	pvserver		[illegible] GiB 36.93%
tcn668.bullx	System Total		[illegible] GiB 42.93%
	pvserver		[illegible] GiB 34.28%
tcn669.bullx	System Total		[illegible] GiB 41.70%
	pvserver		[illegible] GiB 32.98%
tcn670.bullx	System Total		[illegible] GiB 44.45%
	pvserver		[illegible] GiB 33.29%
tcn671.bullx	System Total		[illegible] GiB 43.13%
	pvserver		[illegible] GiB 33.45%
tcn672.bullx	System Total		[illegible] GiB 43.22%

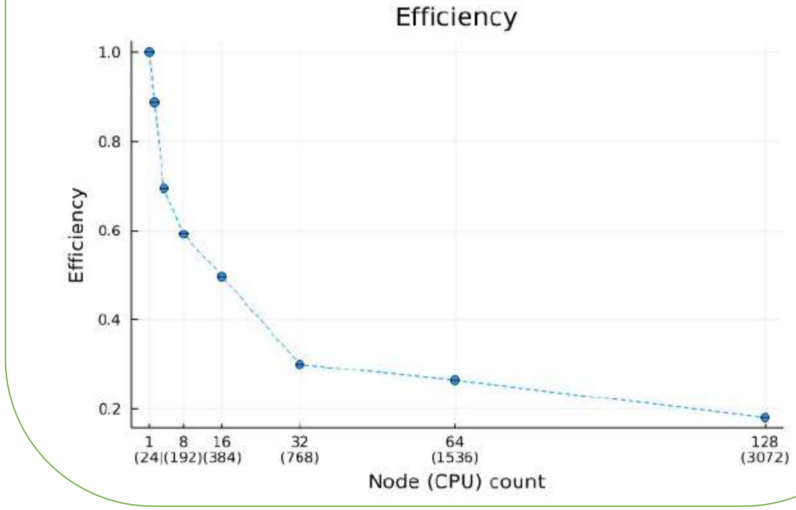
✓ Auto-update

Paul Melis, SURF

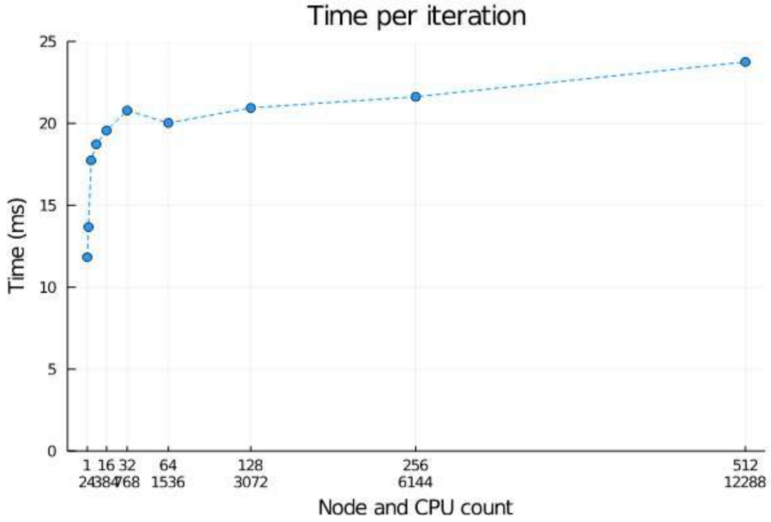
HPC – Scaling and performance

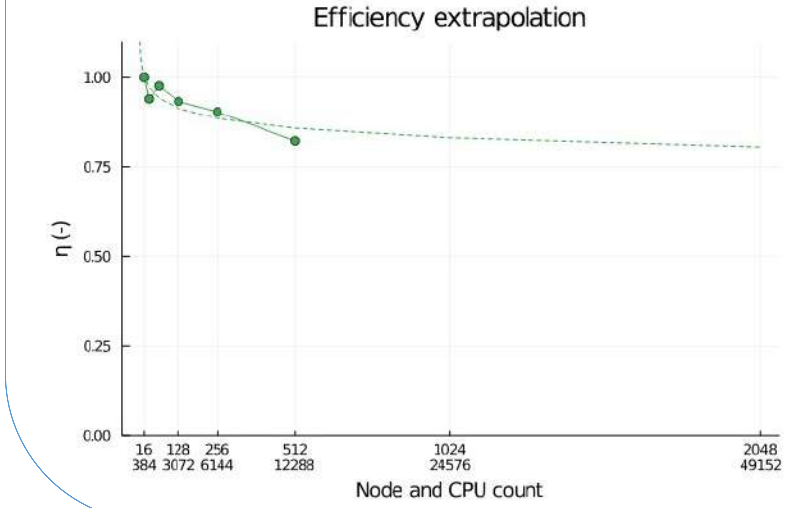
Strong-scaling





Weak-scaling



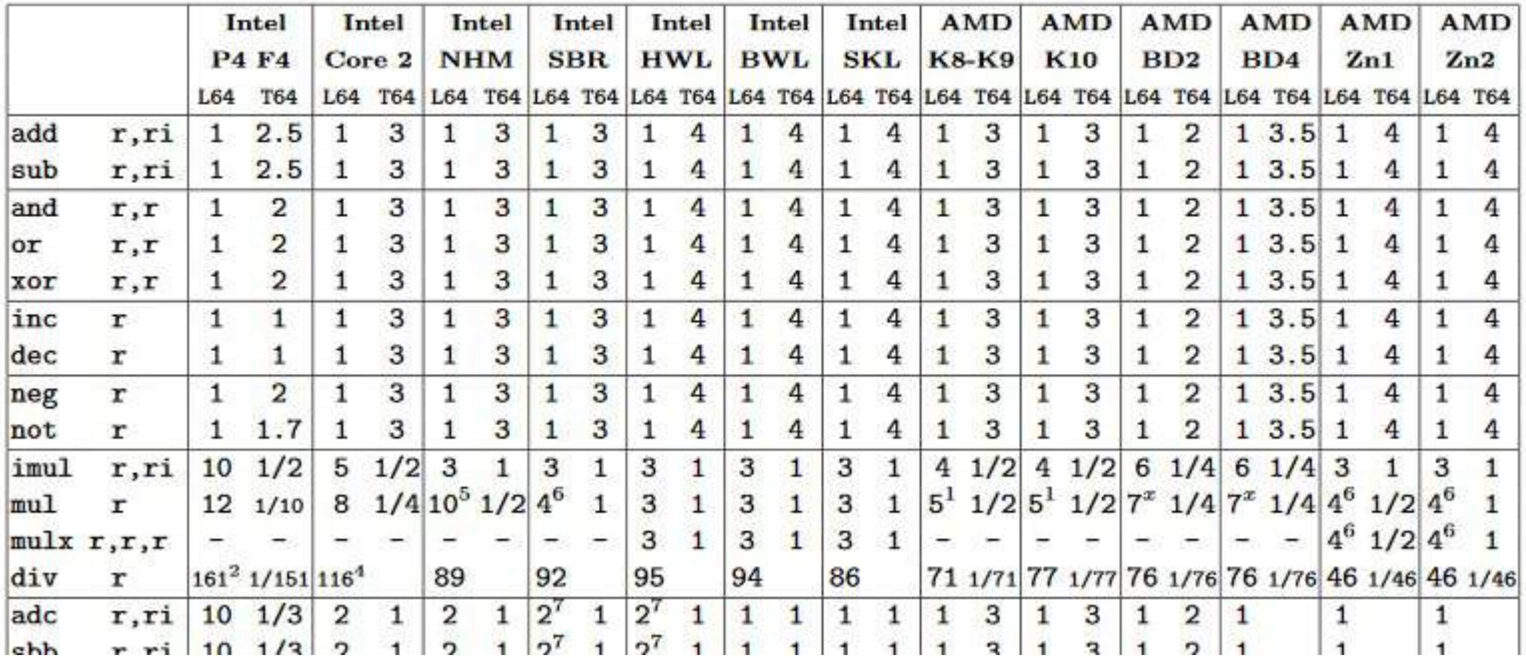


Why is it important to improve scaling?
Which scaling is more important?
Which on is more difficult to improve?

Low-level techniques

(Implementation techniques targeting architectural properties = close to metal)

Algebraic operations: x / 2 vs. 0.5 * x ?



		Intel		Intel		Intel		Intel		Intel		Intel		Intel		AMD		AMD		AMD		AMD		AMD		AMD	
		P4 F4		Core 2		NHM		SBR		HWL		BWL		SKL		K8-K9		K10		BD2		BD4		Zn1		Zn2	
		L64	T64	L64	T64	L64	T64	L64	T64	L64	T64	L64	T64	L64	T64	L64	T64	L64	T64	L64	T64	L64	T64	L64	T64	L64	T64
add	r,ri	1	2.5	1	3	1	3	1	3	1	4	1	4	1	4	1	3	1	3	1	2	1	3.5	1	4	1	4
sub	r,ri	1	2.5	1	3	1	3	1	3	1	4	1	4	1	4	1	3	1	3	1	2	1	3.5	1	4	1	4
and	r,r	1	2	1	3	1	3	1	3	1	4	1	4	1	4	1	3	1	3	1	2	1	3.5	1	4	1	4
or	r,r	1	2	1	3	1	3	1	3	1	4	1	4	1	4	1	3	1	3	1	2	1	3.5	1	4	1	4
xor	r,r	1	2	1	3	1	3	1	3	1	4	1	4	1	4	1	3	1	3	1	2	1	3.5	1	4	1	4
inc	r	1	1	1	3	1	3	1	3	1	4	1	4	1	4	1	3	1	3	1	2	1	3.5	1	4	1	4
dec	r	1	1	1	3	1	3	1	3	1	4	1	4	1	4	1	3	1	3	1	2	1	3.5	1	4	1	4
neg	r	1	2	1	3	1	3	1	3	1	4	1	4	1	4	1	3	1	3	1	2	1	3.5	1	4	1	4
not	r	1	1.7	1	3	1	3	1	3	1	4	1	4	1	4	1	3	1	3	1	2	1	3.5	1	4	1	4
imul	r,ri	10	1/2	5	1/2	3	1	3	1	3	1	3	1	3	1	4	1/2	4	1/2	6	1/4	6	1/4	3	1	3	1
mul	r	12	1/10	8	1/4	10[5]	1/2	4[6]	1	3	1	3	1	3	1	5[1]	1/2	5[1]	1/2	7[x]	1/4	7[x]	1/4	4[6]	1/2	4[6]	1
mulx	r,r,r	–	–	–	–	–	–	–	–	3	1	3	1	3	1	–	–	–	–	–	–	–	–	4[6]	1/2	4[6]	1
div	r	161[2]	1/151	116[4]		89		92		95		94		86		71	1/71	77	1/77	76	1/76	76	1/76	46	1/46	46	1/46
adc	r,ri	10	1/3	2	1	2	1	2[7]	1	2[7]	1	1	1	1	1	1	3	1	3	1	2	1		1		1	
sbb	r,ri	10	1/3	2	1	2	1	2[7]	1	2[7]	1	1	1	1	1	1	3	1	3	1	2	1		1		1	

*Note: ***0.5** has a different round-off error than **/ 2**!*

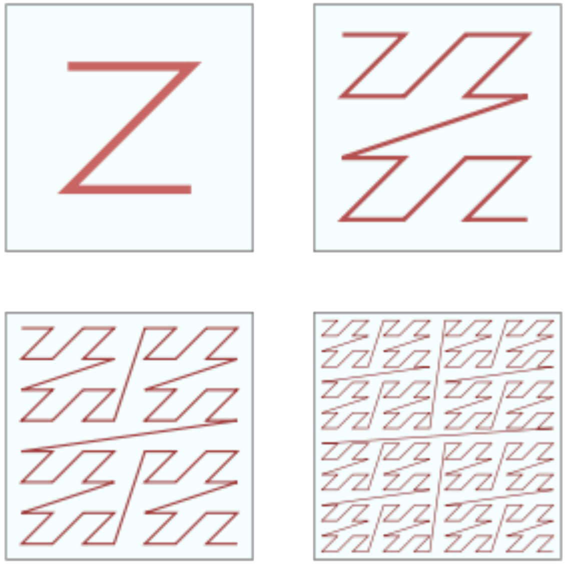
https://gmplib.org/~tege/x86-timing.pdf

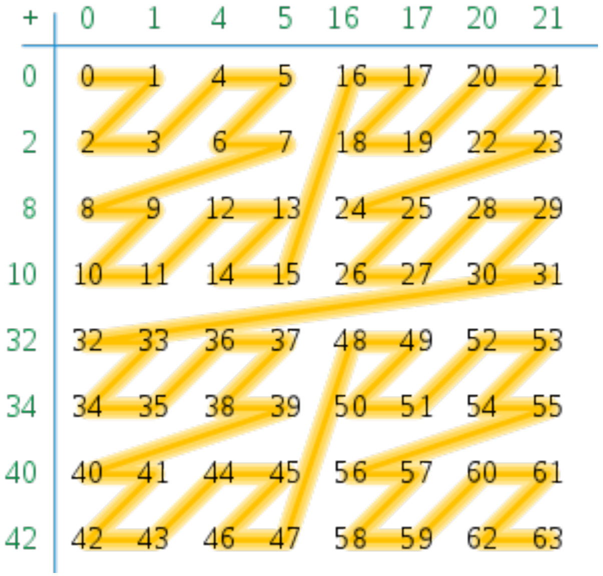
Improving data locality – spatial indexing

There are numerous other space-filling curves.

Heavily used e.g. in GPU related applications.

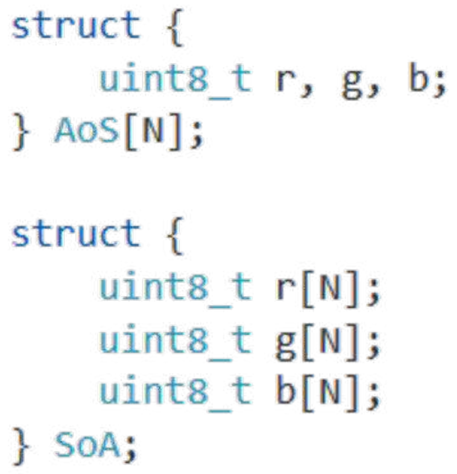
It can improve cache hits significantly.





+	0	1	4	5	16	17	20	21
0	0	1	4	5	16	17	20	21
2	2	3	6	7	18	19	22	23
8	8	9	12	13	24	25	28	29
10	10	11	14	15	26	27	30	31
32	32	33	36	37	48	49	52	53
34	34	35	38	39	50	51	54	55
40	40	41	44	45	56	57	60	61
42	42	43	46	47	58	59	62	63

Other basic programming techniques



```
struct {
    uint8_t r, g, b;
} AoS[N];

struct {
    uint8_t r[N];
    uint8_t g[N];
    uint8_t b[N];
} SoA;
```

- Appropriate data structures (e.g. AoS vs. SoA) that improve data locality.
  This depends on the access pattern of the values.

- Loop optimizations (e.g. via loop transformations).
  Via various transformations fusing loops together can improve performance substantially, plus it allows for additional optimizations (e.g., data reuse).

- Aim for data reuse.
  Commonly used values can be stored instead of being recomputed.

Medium-level techniques

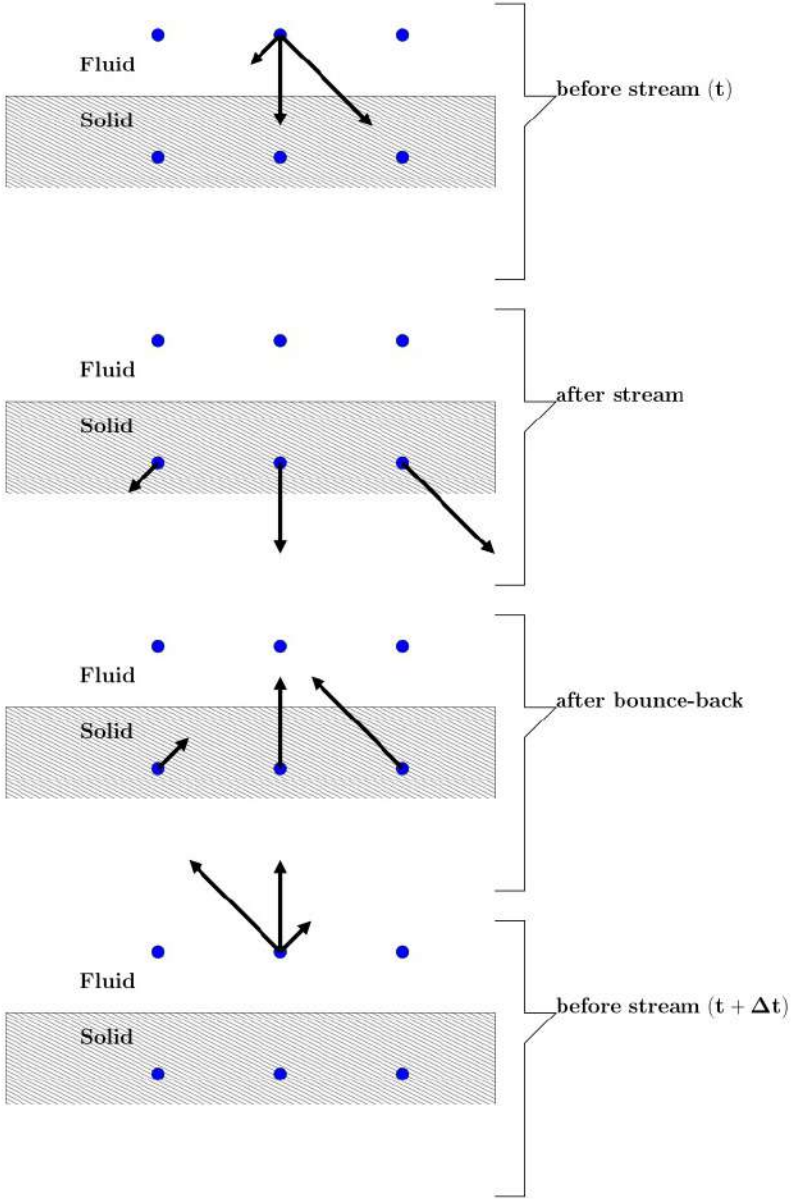
(improvements in the implementation methods)

Metaprogramming example

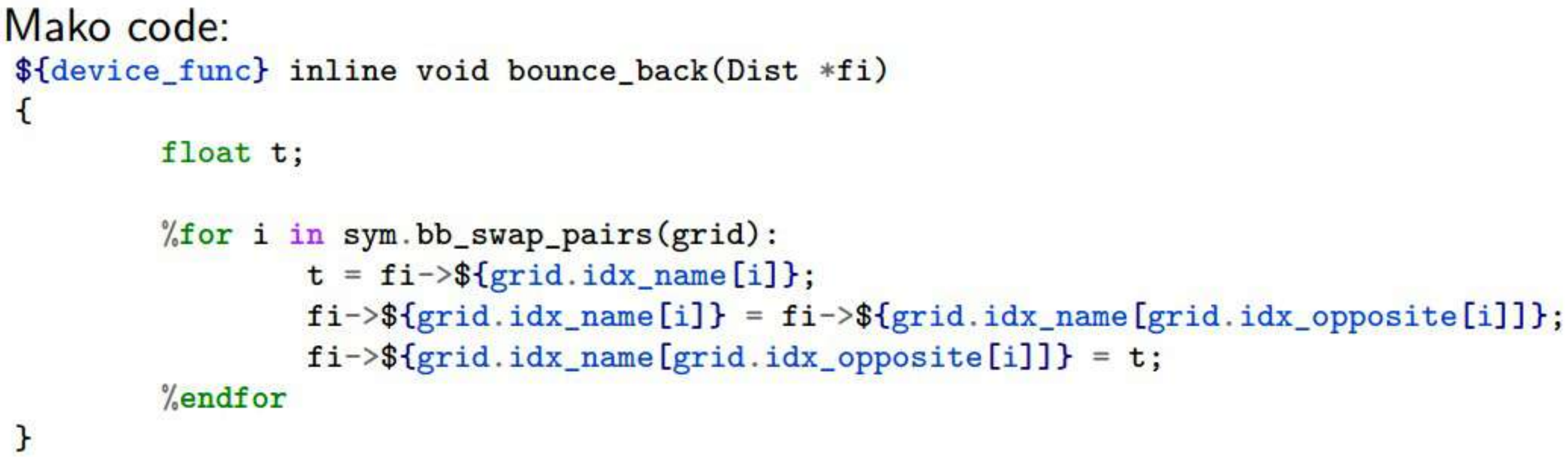
Can be used to (amongst others):

- Keep the model description close to math form.
- Generate code from symbolic math expressions conveniently.
- Generate code for multiple backends (e.g. CPU, GPU).
- Keep the code flexible and general.

Two example algorithms (from a fluid flow simulation):
A.) streaming data on multiple discretized grids
B.) implement a mathematical vector equation



Metaprogramming allows for general code that works in 2D or 3D.

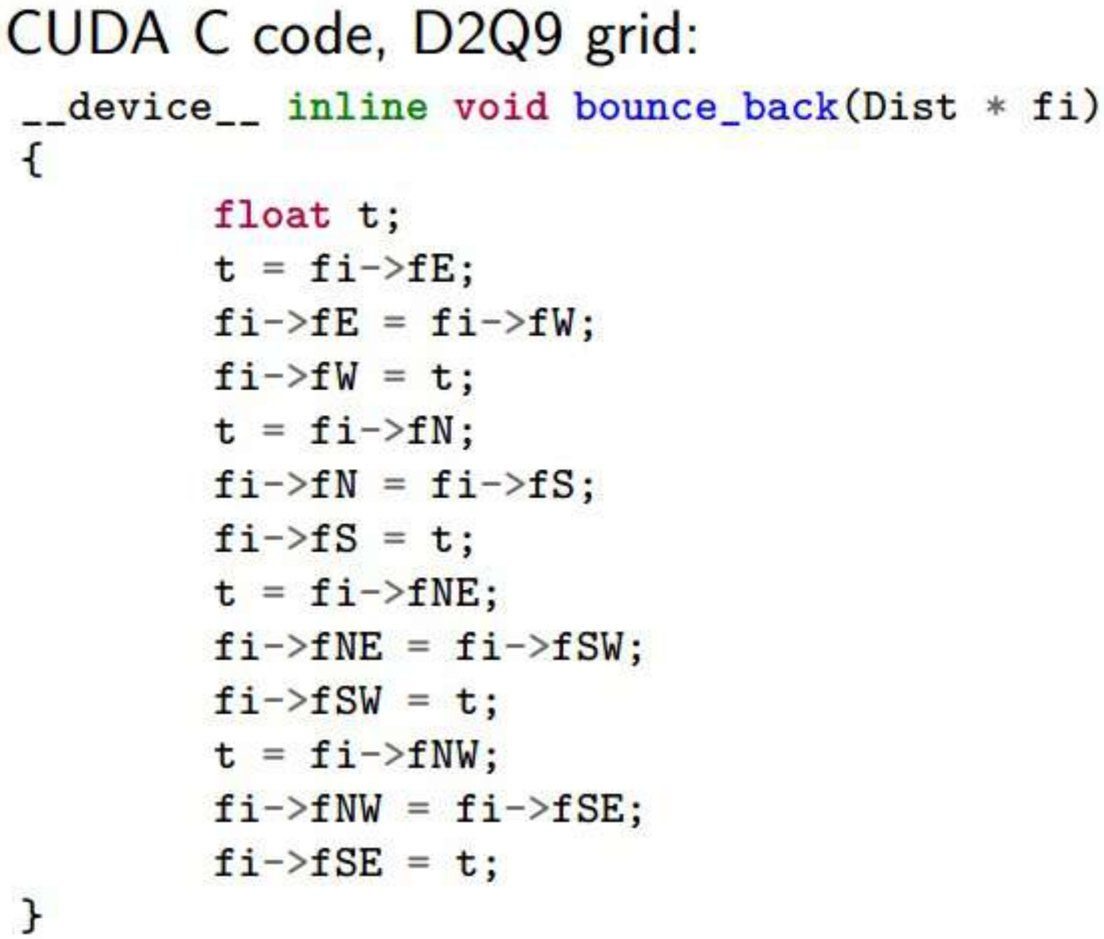


Mako code:

```
${device_func} inline void bounce_back(Dist *fi)
{
        float t;

        %for i in sym.bb_swap_pairs(grid):
                t = fi->${grid.idx_name[i]};
                fi->${grid.idx_name[i]} = fi->${grid.idx_name[grid.idx_opposite[i]]};
                fi->${grid.idx_name[grid.idx_opposite[i]]} = t;
        %endfor
}
```

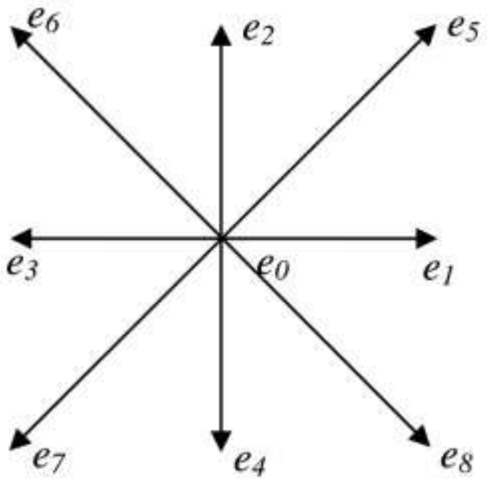
Metaprogramming implementation of the bounce-back method in Python, using the Mako templating language. This technique is trivial in some languages (e.g., lisp), and quite difficult in others (e.g., fortran).

Generated code – CUDA, 2D mesh, Q9



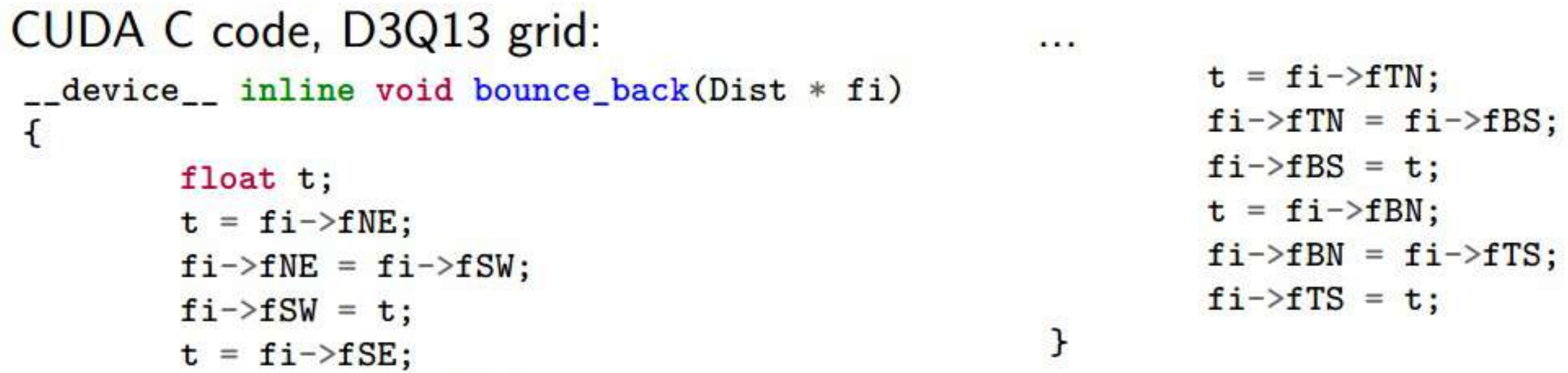
CUDA C code, D2Q9 grid:

```
__device__ inline void bounce_back(Dist * fi)
{
        float t;
        t = fi->fE;
        fi->fE = fi->fW;
        fi->fW = t;
        t = fi->fN;
        fi->fN = fi->fS;
        fi->fS = t;
        t = fi->fNE;
        fi->fNE = fi->fSW;
        fi->fSW = t;
        t = fi->fNW;
        fi->fNW = fi->fSE;
        fi->fSE = t;
}
```



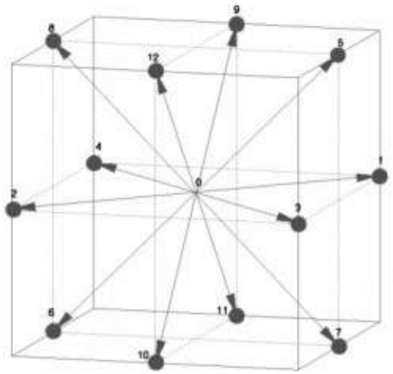
D2Q9

Generated code – CUDA, 3D mesh, Q13



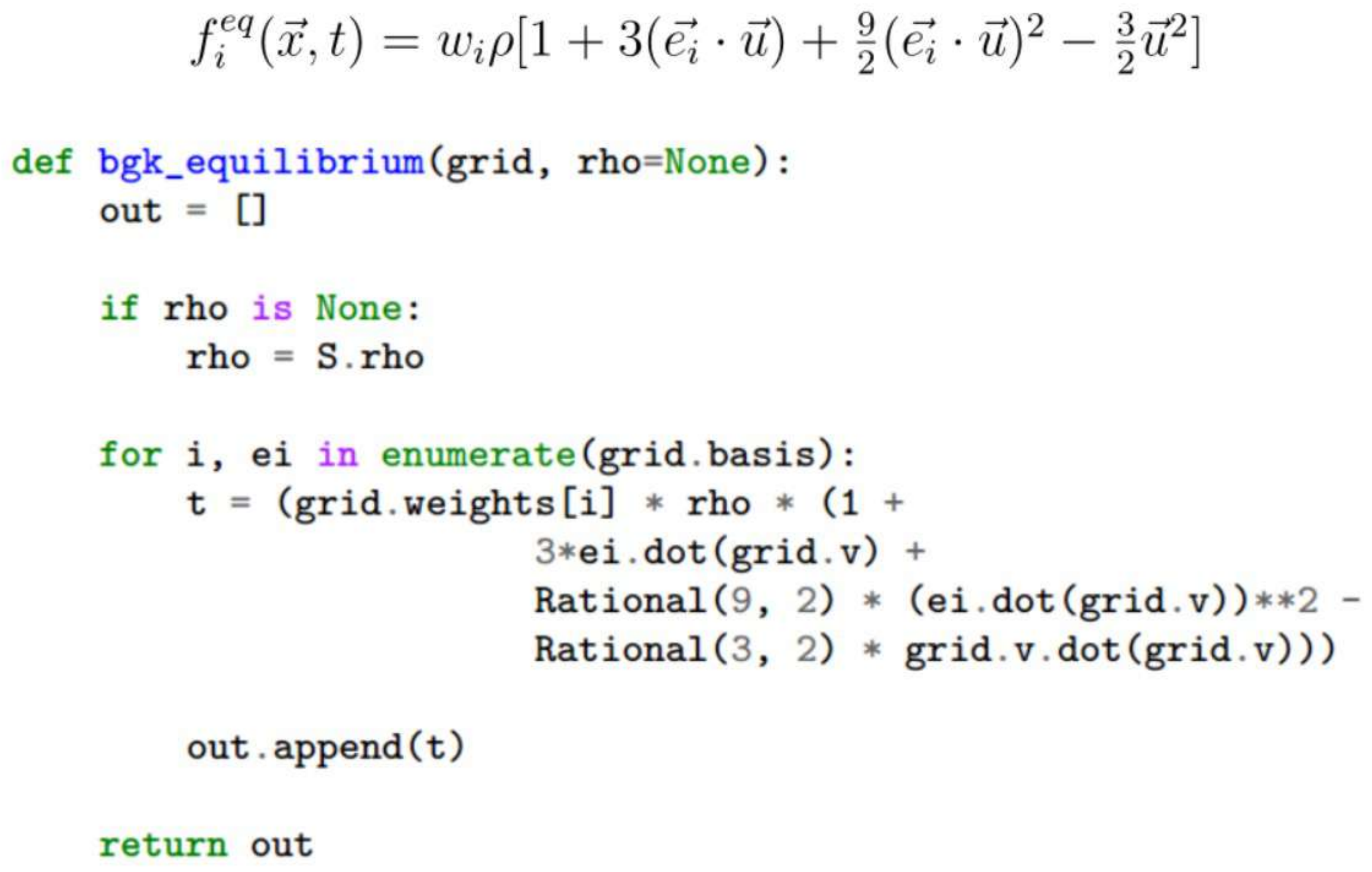
CUDA C code, D3Q13 grid:

```
__device__ inline void bounce_back(Dist * fi)
{
        float t;
        t = fi->fNE;
        fi->fNE = fi->fSW;
        fi->fSW = t;
        t = fi->fSE;
        fi->fSE = fi->fNW;
        fi->fNW = t;
        t = fi->fTE;
        fi->fTE = fi->fBW;
        fi->fBW = t;
        t = fi->fBE;
        fi->fBE = fi->fTW;
        fi->fTW = t;
...
        t = fi->fTN;
        fi->fTN = fi->fBS;
        fi->fBS = t;
        t = fi->fBN;
        fi->fBN = fi->fTS;
        fi->fTS = t;
}
```



D3Q13

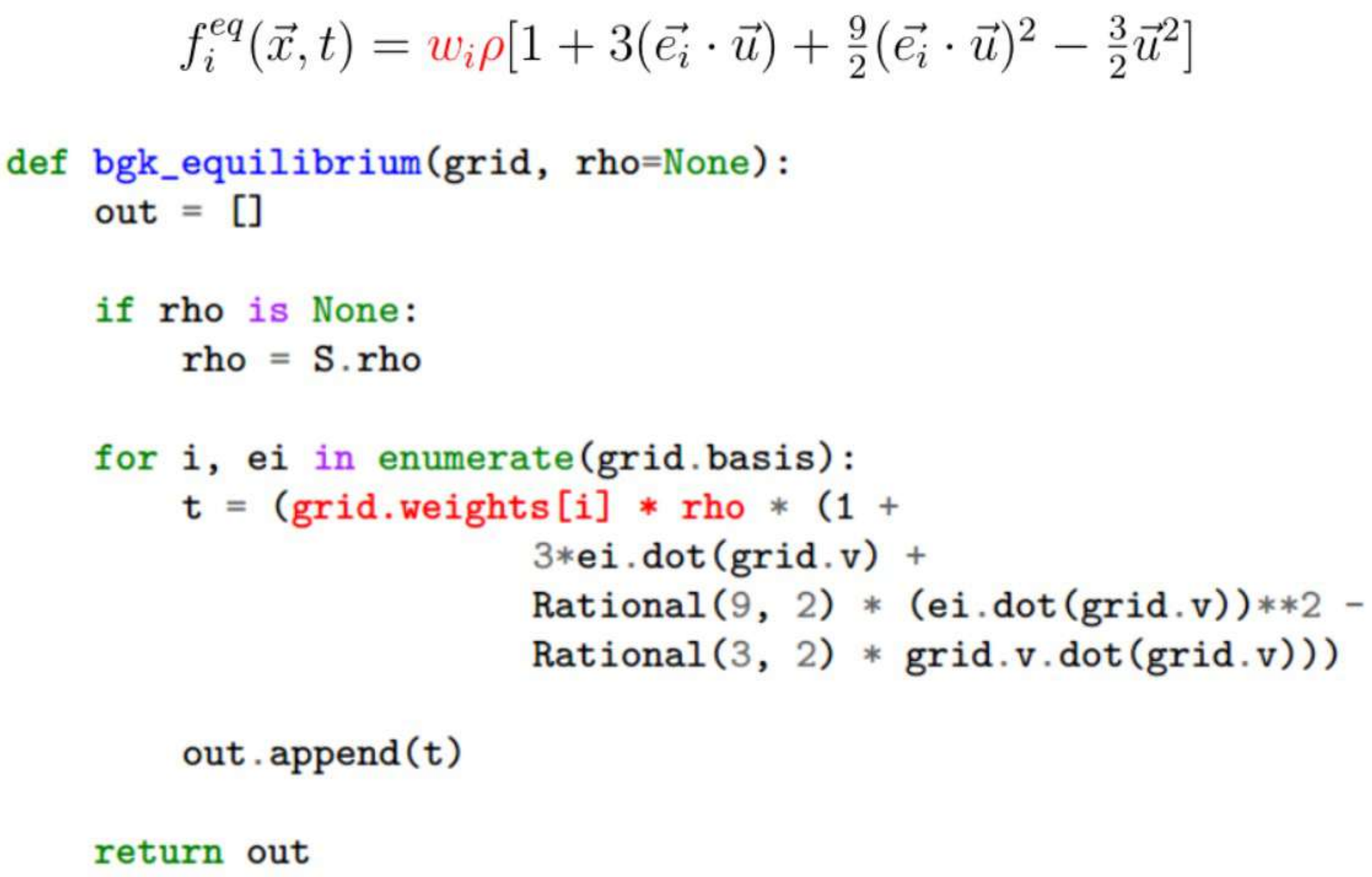
Metaprogramming allows for abstractions, to keep code close to math equations



$$f_i^{eq}(\vec{x},t) = w_i\rho[1 + 3(\vec{e_i}\cdot\vec{u}) + \frac{9}{2}(\vec{e_i}\cdot\vec{u})^2 - \frac{3}{2}\vec{u}^2]$$

```
def bgk_equilibrium(grid, rho=None):
    out = []

    if rho is None:
        rho = S.rho

    for i, ei in enumerate(grid.basis):
        t = (grid.weights[i] * rho * (1 +
                        3*ei.dot(grid.v) +
                        Rational(9, 2) * (ei.dot(grid.v))**2 -
                        Rational(3, 2) * grid.v.dot(grid.v)))

        out.append(t)

    return out
```

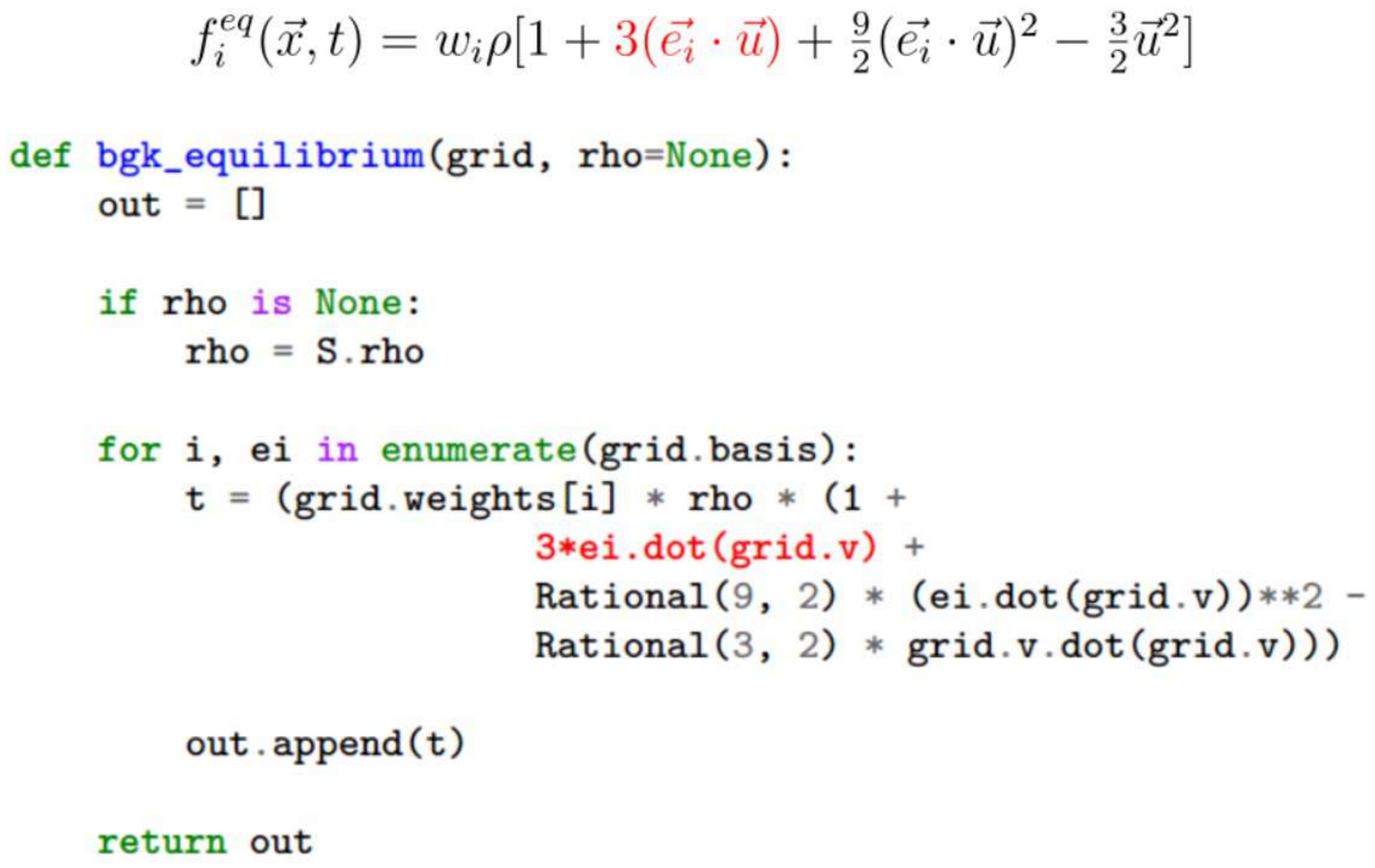
Code close to math equations



$$f_i^{eq}(\vec{x},t) = w_i\rho[1 + 3(\vec{e_i}\cdot\vec{u}) + \frac{9}{2}(\vec{e_i}\cdot\vec{u})^2 - \frac{3}{2}\vec{u}^2]$$

```
def bgk_equilibrium(grid, rho=None):
    out = []

    if rho is None:
        rho = S.rho

    for i, ei in enumerate(grid.basis):
        t = (grid.weights[i] * rho * (1 +
                        3*ei.dot(grid.v) +
                        Rational(9, 2) * (ei.dot(grid.v))**2 -
                        Rational(3, 2) * grid.v.dot(grid.v)))

        out.append(t)

    return out
```

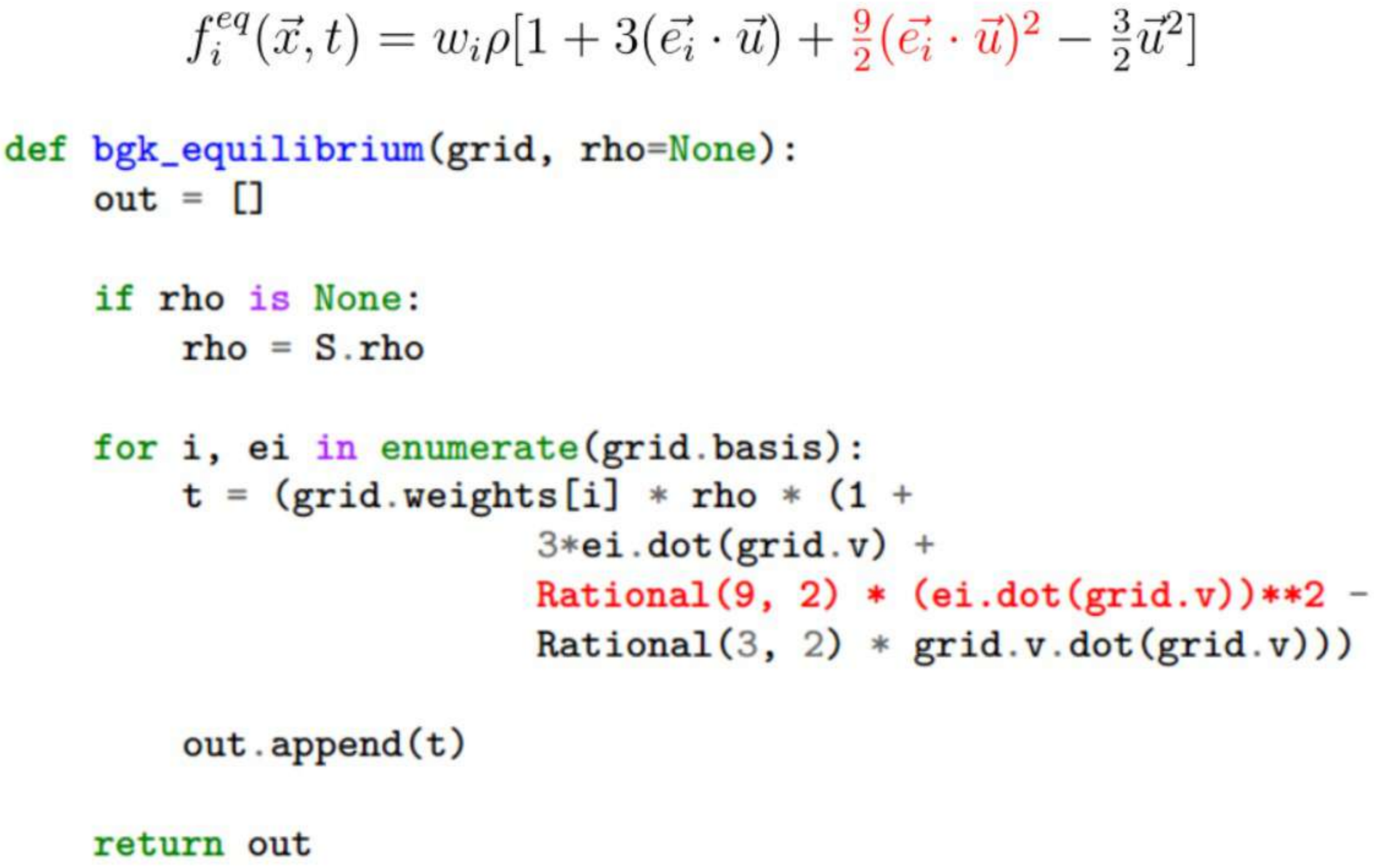
Code close to math equations



$$f_i^{eq}(\vec{x},t) = w_i\rho[1 + 3(\vec{e_i}\cdot\vec{u}) + \frac{9}{2}(\vec{e_i}\cdot\vec{u})^2 - \frac{3}{2}\vec{u}^2]$$

```
def bgk_equilibrium(grid, rho=None):
    out = []

    if rho is None:
        rho = S.rho

    for i, ei in enumerate(grid.basis):
        t = (grid.weights[i] * rho * (1 +
                        3*ei.dot(grid.v) +
                        Rational(9, 2) * (ei.dot(grid.v))**2 -
                        Rational(3, 2) * grid.v.dot(grid.v)))

        out.append(t)

    return out
```

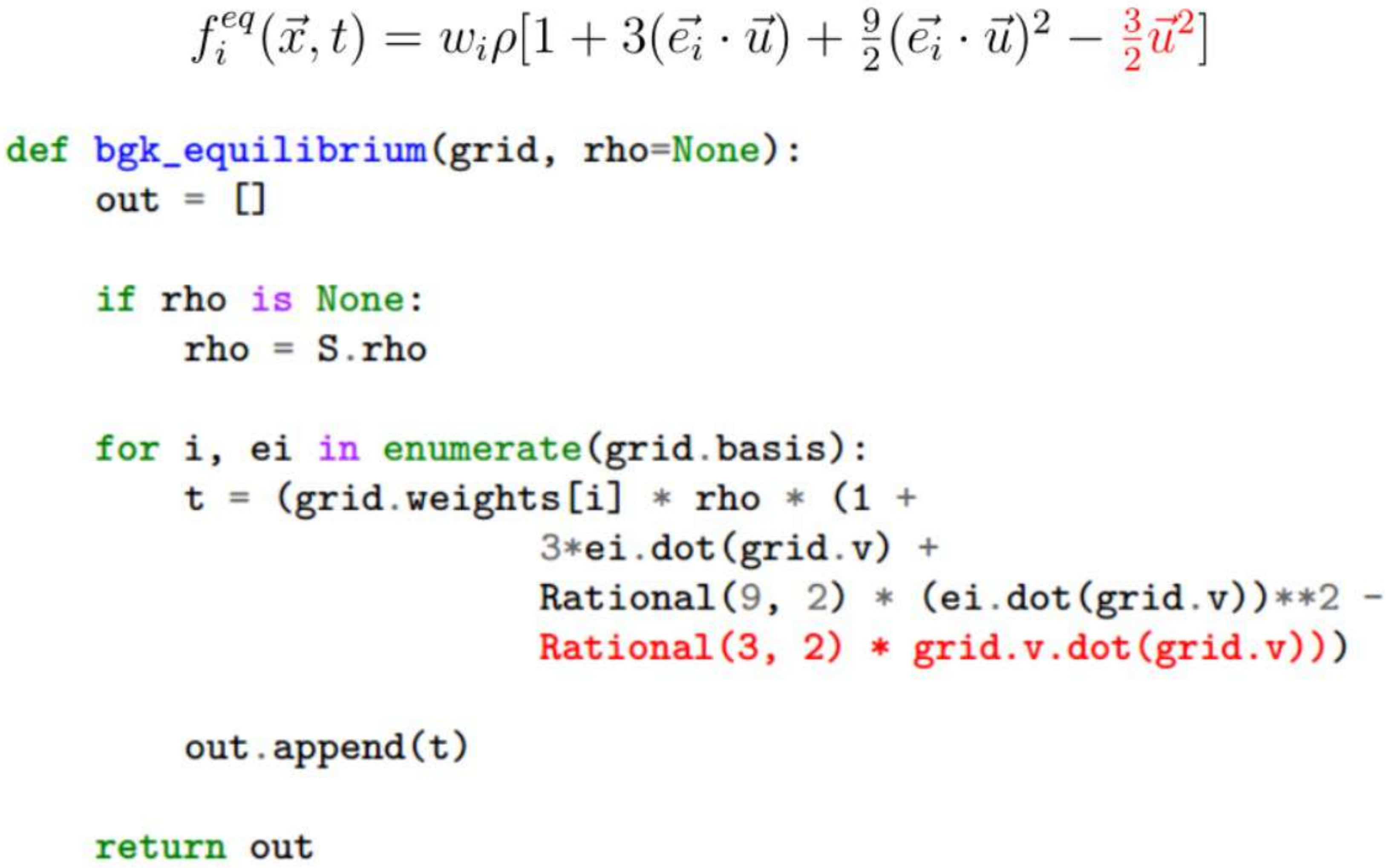
Code close to math equations



$$f_i^{eq}(\vec{x},t) = w_i\rho[1 + 3(\vec{e_i}\cdot\vec{u}) + \frac{9}{2}(\vec{e_i}\cdot\vec{u})^2 - \frac{3}{2}\vec{u}^2]$$

```
def bgk_equilibrium(grid, rho=None):
    out = []

    if rho is None:
        rho = S.rho

    for i, ei in enumerate(grid.basis):
        t = (grid.weights[i] * rho * (1 +
                                     3*ei.dot(grid.v) +
                                     Rational(9, 2) * (ei.dot(grid.v))**2 -
                                     Rational(3, 2) * grid.v.dot(grid.v)))

        out.append(t)

    return out
```

Code close to math equations



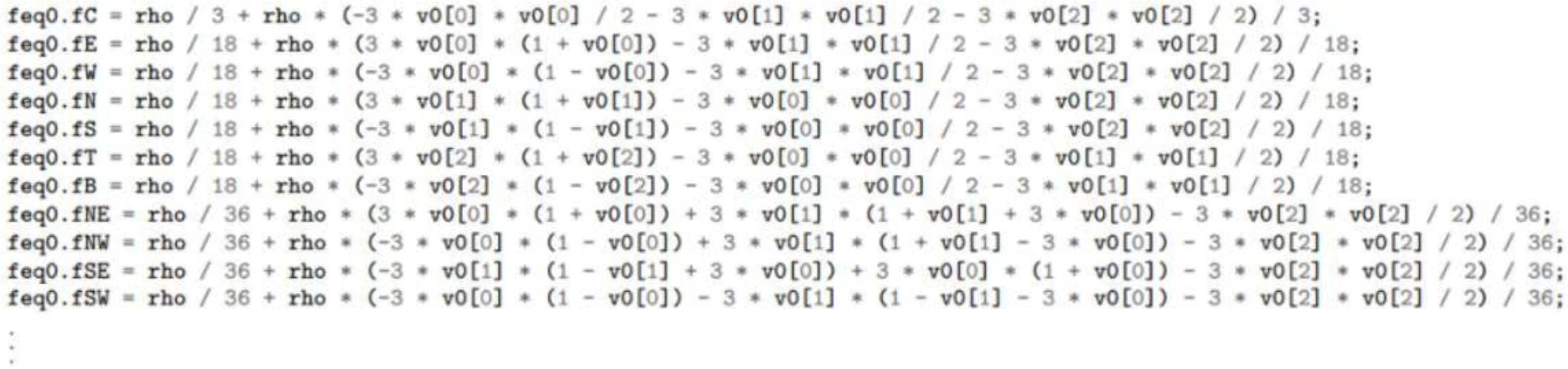
$$f_i^{eq}(\vec{x},t) = w_i\rho[1 + 3(\vec{e_i}\cdot\vec{u}) + \frac{9}{2}(\vec{e_i}\cdot\vec{u})^2 - \frac{3}{2}\vec{u}^2]$$

```
def bgk_equilibrium(grid, rho=None):
    out = []

    if rho is None:
        rho = S.rho

    for i, ei in enumerate(grid.basis):
        t = (grid.weights[i] * rho * (1 +
                                      3*ei.dot(grid.v) +
                                      Rational(9, 2) * (ei.dot(grid.v))**2 -
                                      Rational(3, 2) * grid.v.dot(grid.v)))

        out.append(t)

    return out
```

Code close to math equations

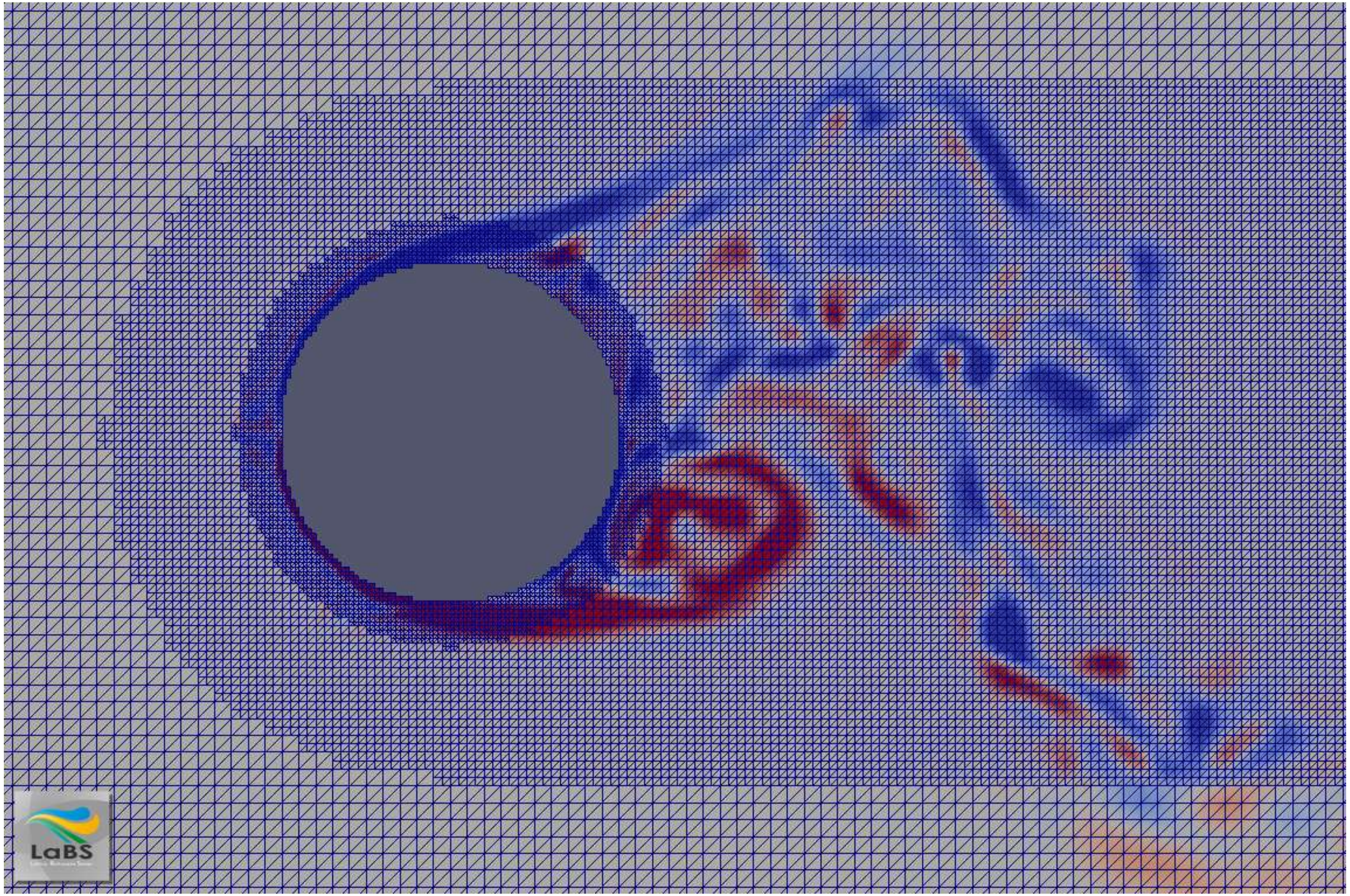
$$f_i^{eq}(\vec{x},t) = w_i\rho[1 + 3(\vec{e_i}\cdot\vec{u}) + \tfrac{9}{2}(\vec{e_i}\cdot\vec{u})^2 - \tfrac{3}{2}\vec{u}^2]$$

The generated code:



```
feq0.fC = rho / 3 + rho * (-3 * v0[0] * v0[0] / 2 - 3 * v0[1] * v0[1] / 2 - 3 * v0[2] * v0[2] / 2) / 3;
feq0.fE = rho / 18 + rho * (3 * v0[0] * (1 + v0[0]) - 3 * v0[1] * v0[1] / 2 - 3 * v0[2] * v0[2] / 2) / 18;
feq0.fW = rho / 18 + rho * (-3 * v0[0] * (1 - v0[0]) - 3 * v0[1] * v0[1] / 2 - 3 * v0[2] * v0[2] / 2) / 18;
feq0.fN = rho / 18 + rho * (3 * v0[1] * (1 + v0[1]) - 3 * v0[0] * v0[0] / 2 - 3 * v0[2] * v0[2] / 2) / 18;
feq0.fS = rho / 18 + rho * (-3 * v0[1] * (1 - v0[1]) - 3 * v0[0] * v0[0] / 2 - 3 * v0[2] * v0[2] / 2) / 18;
feq0.fT = rho / 18 + rho * (3 * v0[2] * (1 + v0[2]) - 3 * v0[0] * v0[0] / 2 - 3 * v0[1] * v0[1] / 2) / 18;
feq0.fB = rho / 18 + rho * (-3 * v0[2] * (1 - v0[2]) - 3 * v0[0] * v0[0] / 2 - 3 * v0[1] * v0[1] / 2) / 18;
feq0.fNE = rho / 36 + rho * (3 * v0[0] * (1 + v0[0]) + 3 * v0[1] * (1 + v0[1] + 3 * v0[0]) - 3 * v0[2] * v0[2] / 2) / 36;
feq0.fNW = rho / 36 + rho * (-3 * v0[0] * (1 - v0[0]) + 3 * v0[1] * (1 + v0[1] - 3 * v0[0]) - 3 * v0[2] * v0[2] / 2) / 36;
feq0.fSE = rho / 36 + rho * (-3 * v0[1] * (1 - v0[1] + 3 * v0[0]) + 3 * v0[0] * (1 + v0[0]) - 3 * v0[2] * v0[2] / 2) / 36;
feq0.fSW = rho / 36 + rho * (-3 * v0[0] * (1 - v0[0]) - 3 * v0[1] * (1 - v0[1] - 3 * v0[0]) - 3 * v0[2] * v0[2] / 2) / 36;
⋮
```

Grid refinement - static



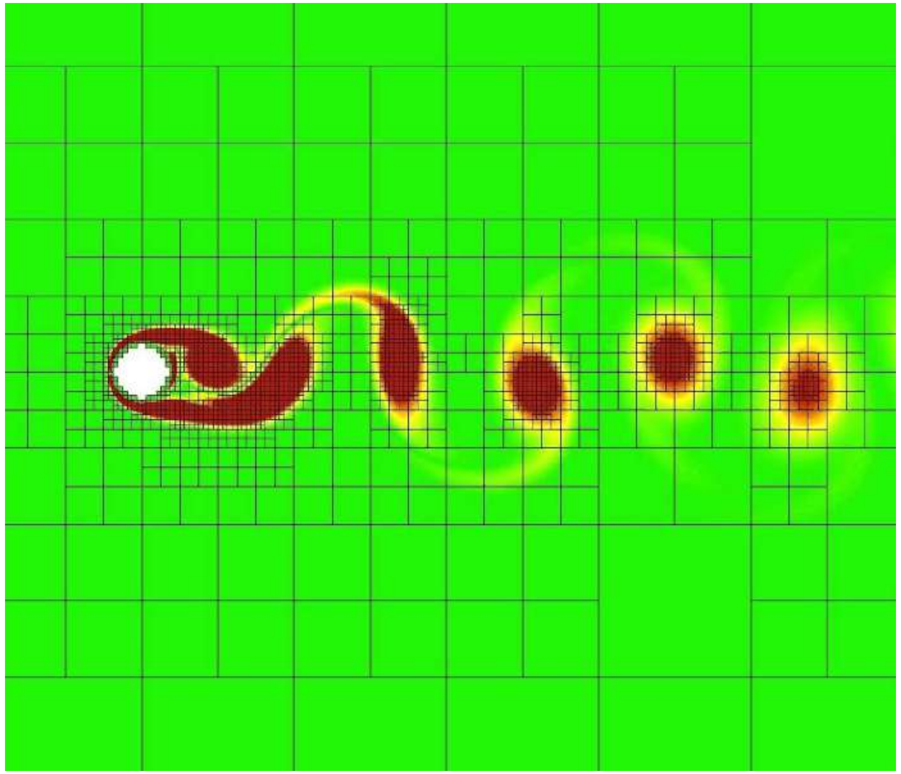
We employ this method

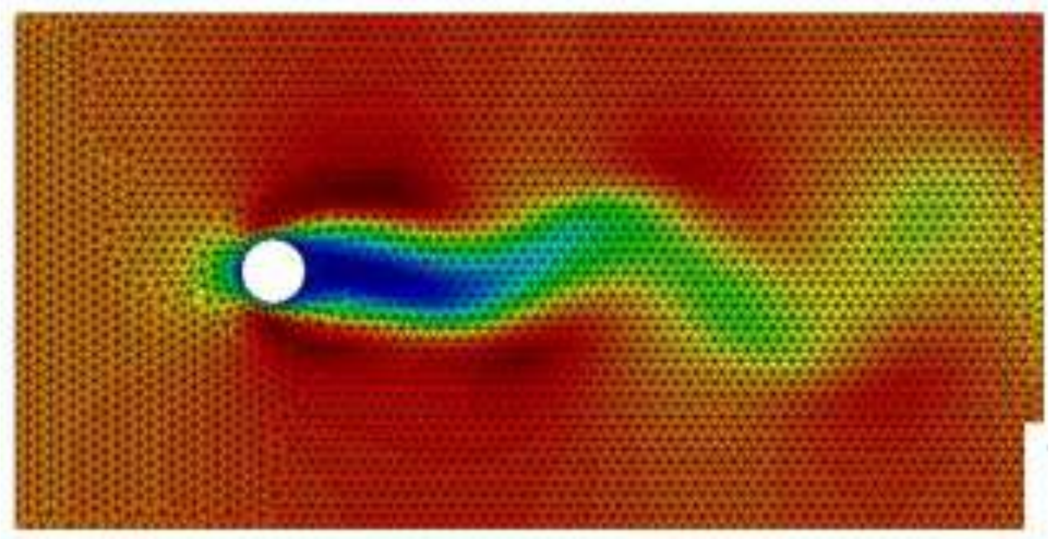
- to increase accuracy or
- to increase performance.

Think about what this means for parallel programming!

- The load-imbalance can be mitigated at the initial domain decomposition.
- At the cost of significant complexity!

Grid refinement - adaptive





- Dynamic (adaptive) grid refinement presents dynamic load-imbalance challenges!

- This is an order of magnitude more difficult problem, that is currently heavily researched.

High-level techniques

(conceptual level of the numerical model – we use information about the specific problem being solved)

Dynamic iteration step-size in a coupled simulation

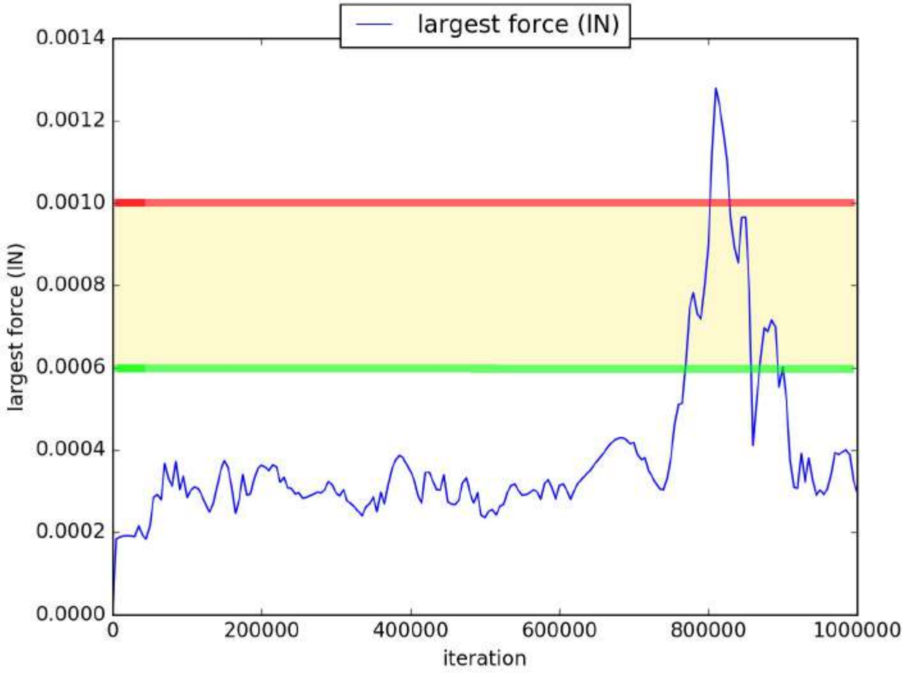
Time steps:

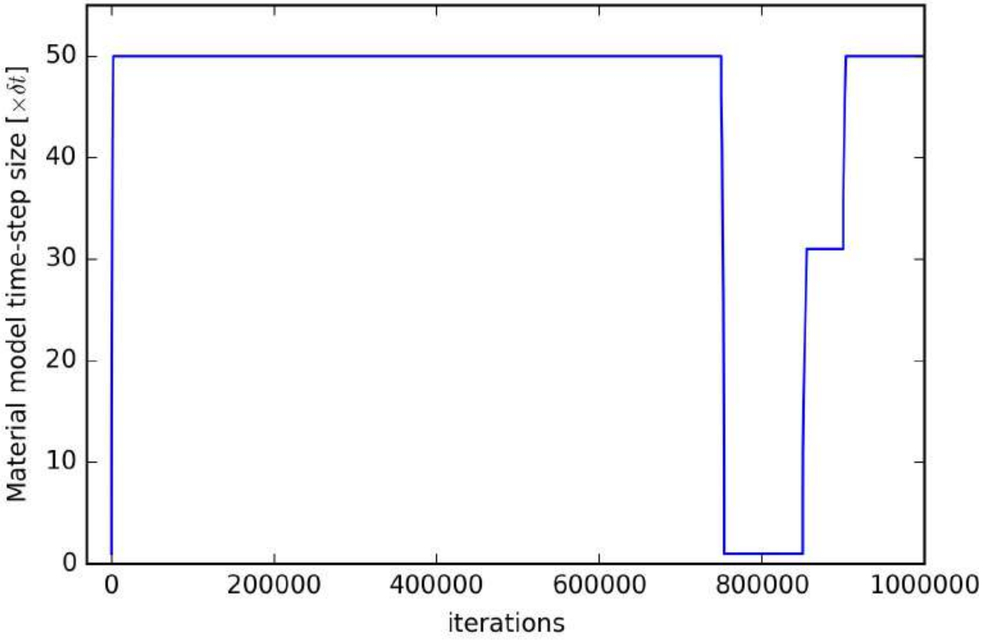
- Fluid field (LBM): dt ~ 1e-7 s
- FSI coupling (IBM): dt ~ 1-6 dt
- Cell model (DEM): dt ~ 1-100 dt

dt ≤ dt ≤ dt

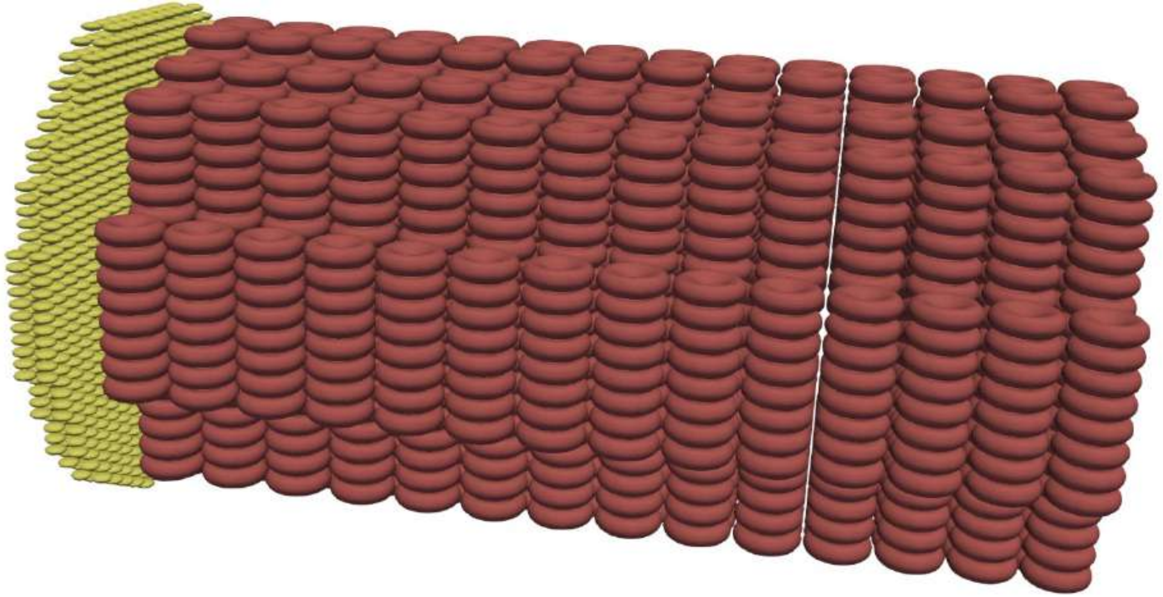
Technically it is an adaptive grid in time with different resolution for the different coupled components.

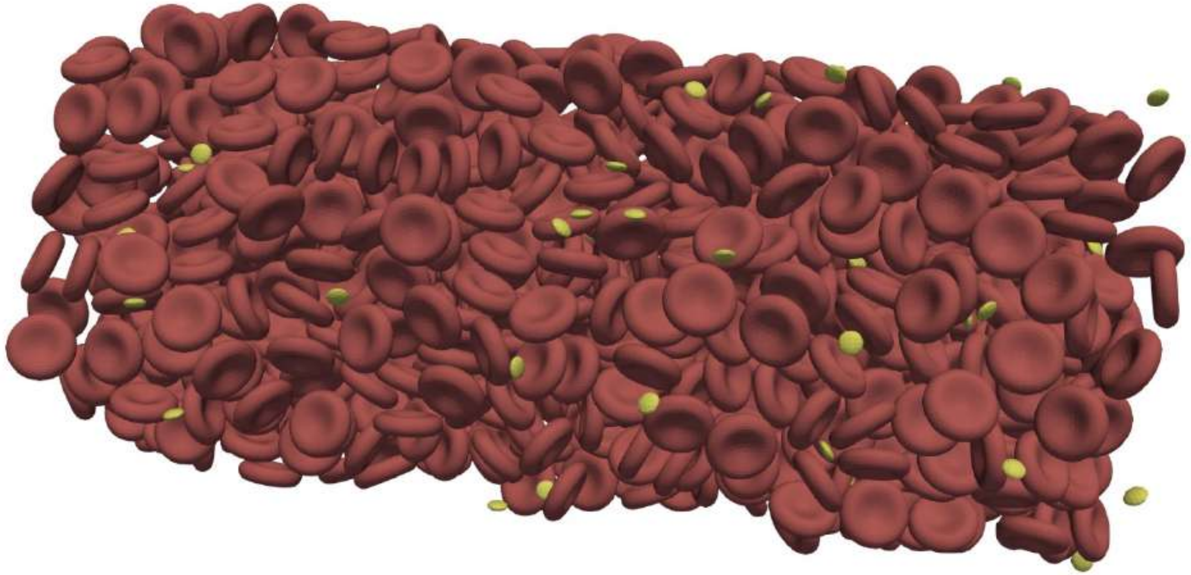
dt:



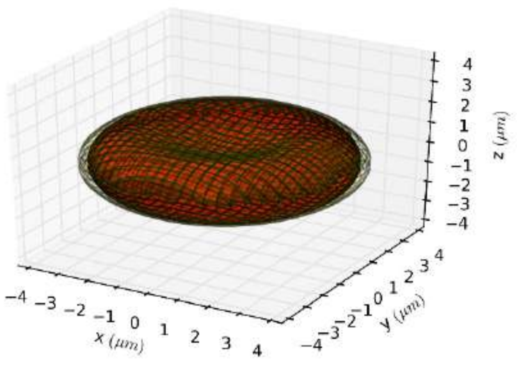


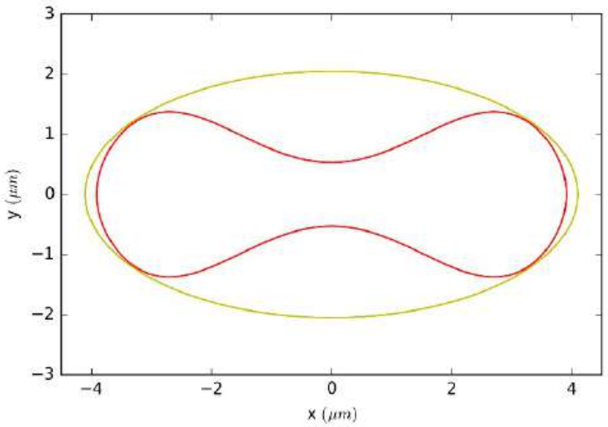
Initial condition optimizations – better mixed cells

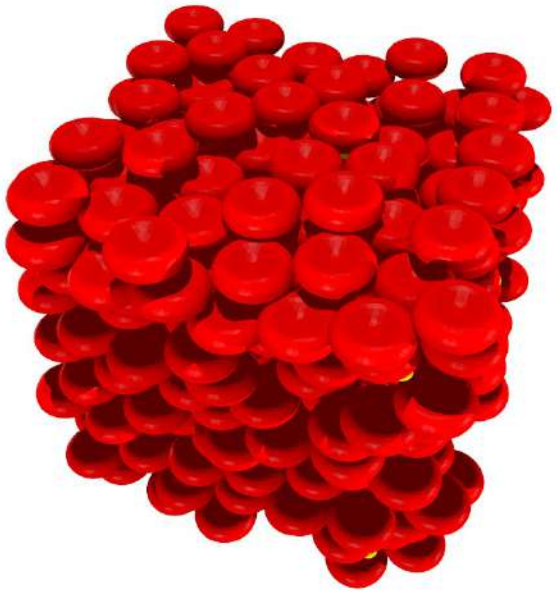


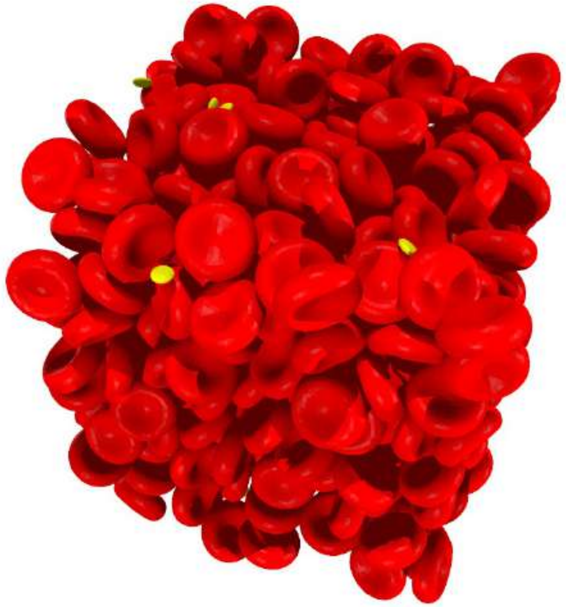


Fast way to compute the initial cell packing









Force-bias method:

$$\vec{F}_{ij} = \boldsymbol{\delta_{ij}} p_{ij} \frac{\vec{r}_j - \vec{r}_i}{|\vec{r}_j - \vec{r}_i|}$$

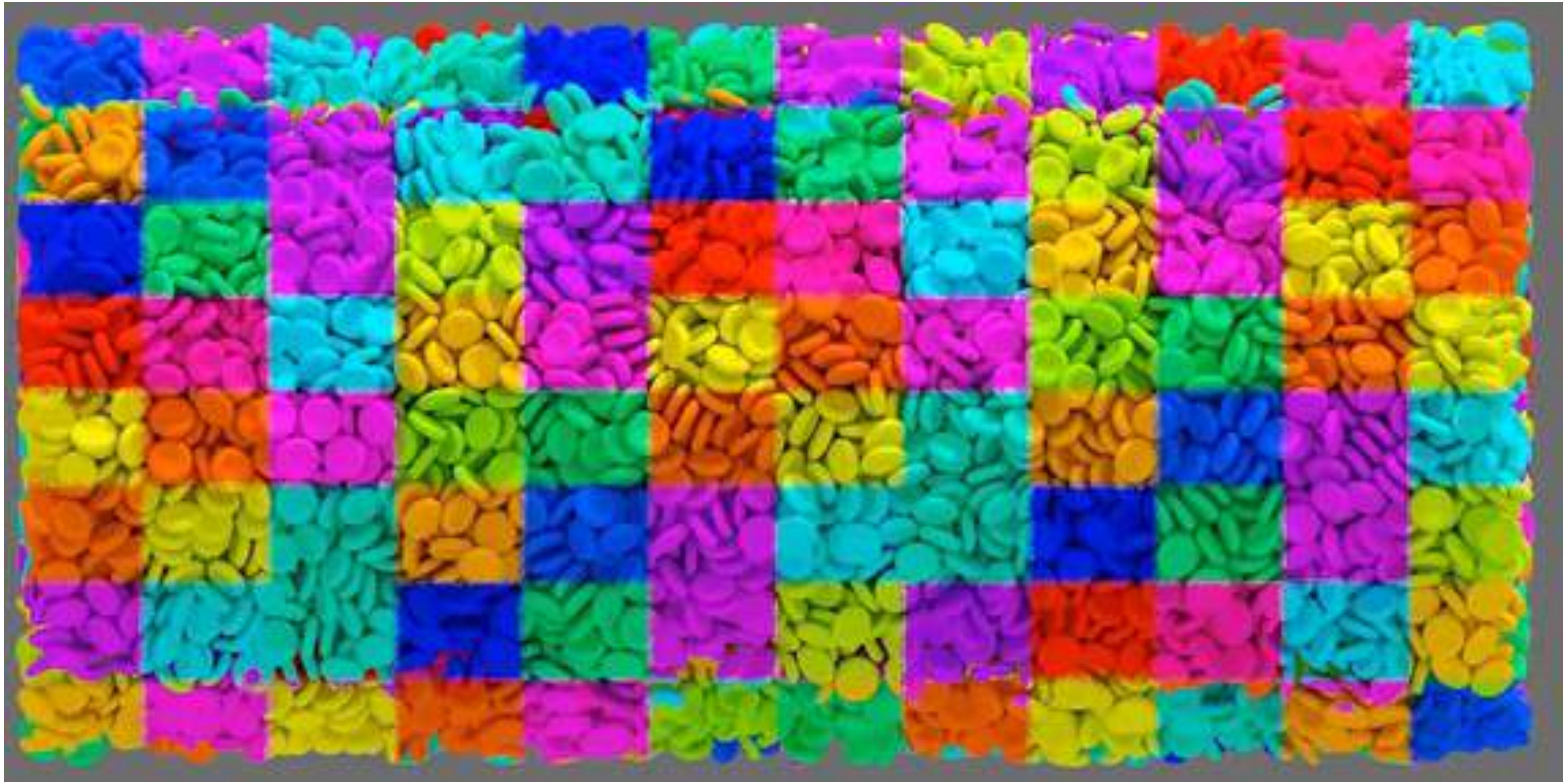
- Potential function is proportional to overlapping volumes
- Regular grid space-partitioning
- OpenMP implementation
- On a 16 core machine it can position millions of cells at 40% hematocrit within an hour of computation wall-time

Gain through some optimization

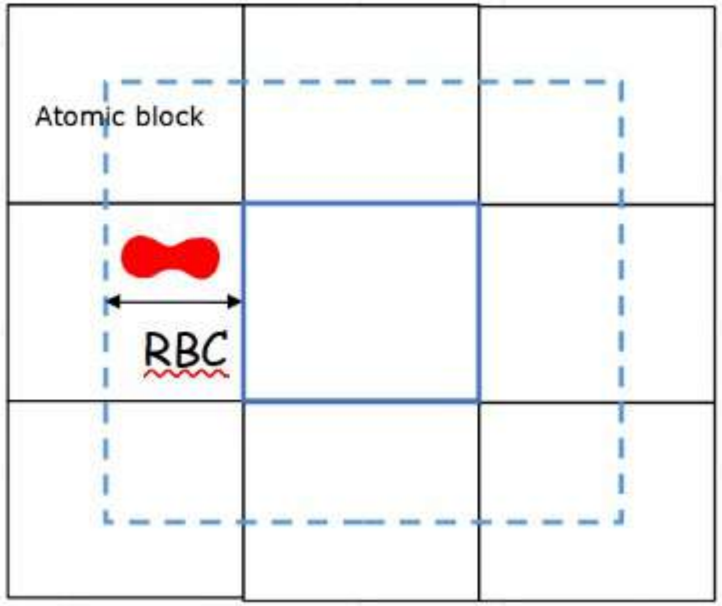
	Wall-time / iteration	Warmup iterations	Simulation iterations	Total time (expected)
original	2.3 s	10M	2M	319.5 days
+ low-level code opt.	0.9 s	10M	2M	125 days
+ better initial conditions	0.9 s	1M	2M	31.25 days
+ adaptive time steps	0.06 s	1M	2M	2.1 days

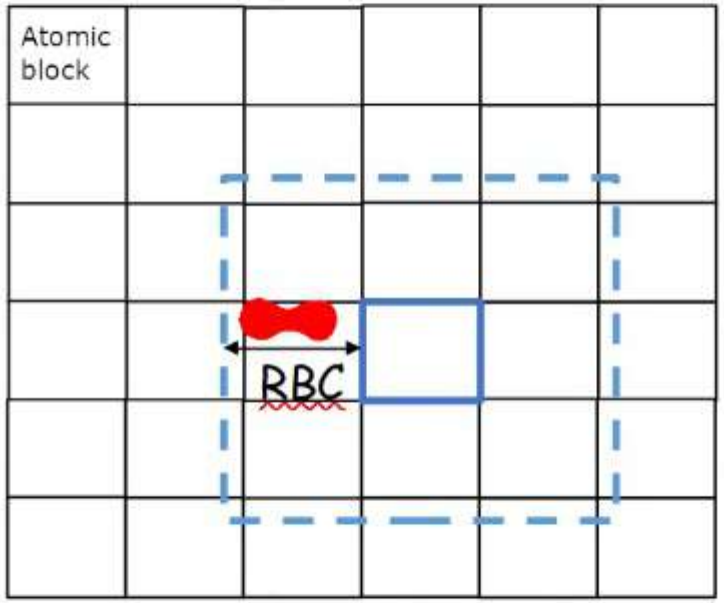
Conceptual level optimization (15x). **Think before you optimize!**

Domain decomposition of coupled codes



Communication as a bottleneck





The communication envelope size is constant (dictated by the simulated system).

The number of neighbours grow as $(2N+1)^3-1$ in a 3D grid, where N is the number of neighbours in one direction.

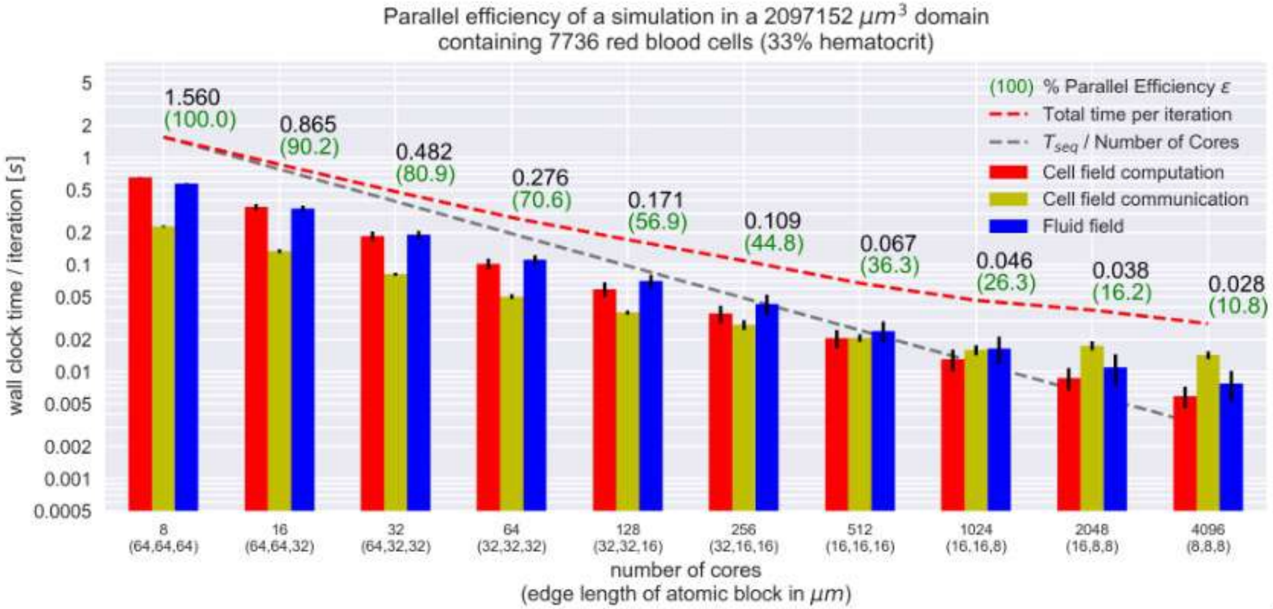
From N=1 to N=2 -> 124–26 = 98 new neightbours.

This is rather typical with simulations, and can cause a serious bottleneck for performance scaling. On an HPC this is a serious issue!
(Think! Is this a strong- or weak-scaling issue?)

In HemoCell to mitigate this particular issue we implemented a two-step communication:

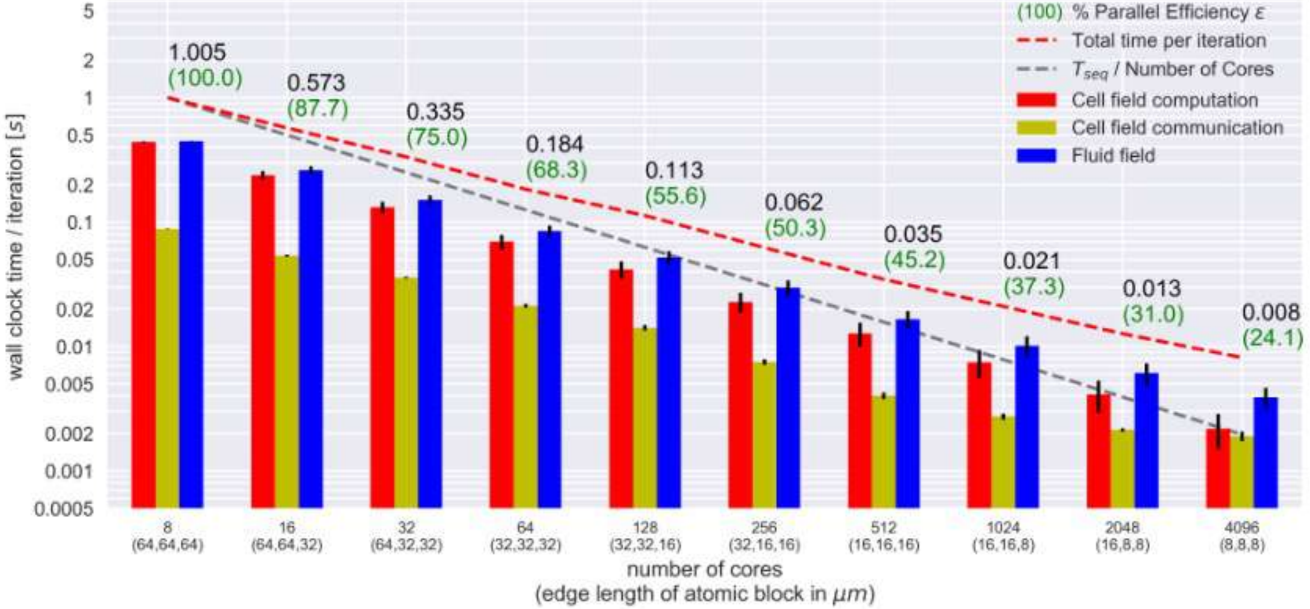
1. Distribute what we specifically need from neighbours
2. Transfer only that

Communication as a bottleneck for scaling

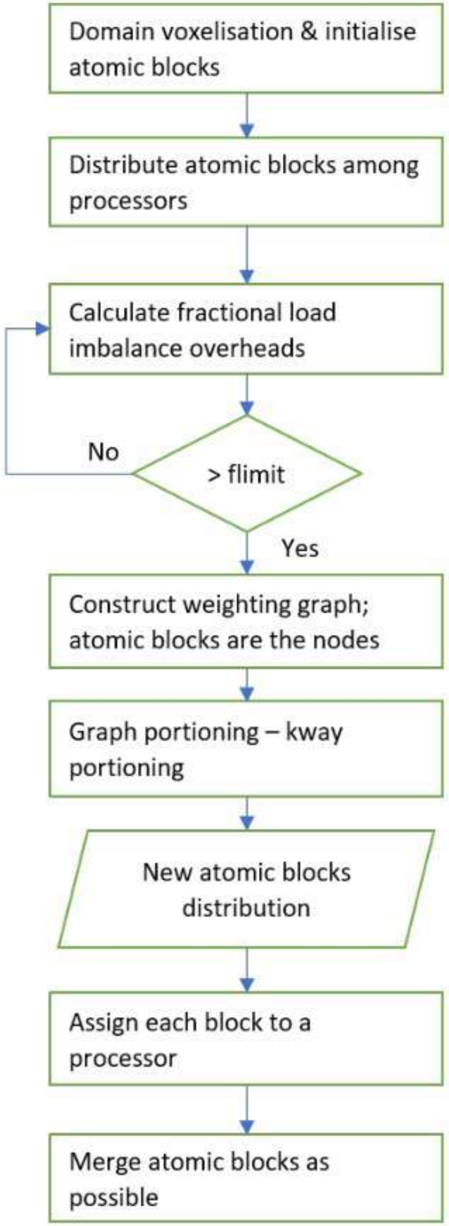


<- Single-step communication

Two-step communication ->



Load-balancing spatial domain distribution



Domain voxelisation & initialise atomic blocks

Distribute atomic blocks among processors

Calculate fractional load imbalance overheads

> flimit

No

Yes

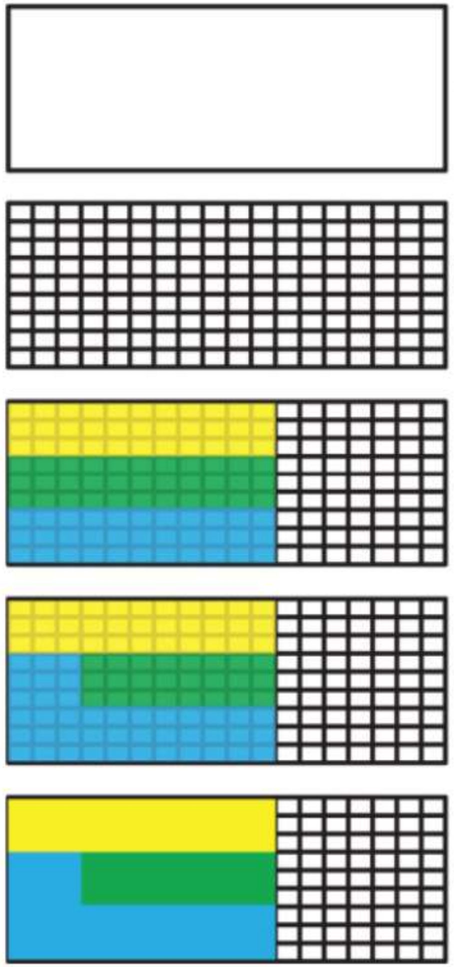
Construct weighting graph; atomic blocks are the nodes

Graph portioning – kway portioning

New atomic blocks distribution

Assign each block to a processor

Merge atomic blocks as possible



a- *Initialize the domain*

b- *Build atomic blocks*

c- *Initial atomic block assignments*

d- *Rebalance atomic blocks*

e- *Merge to large rectangular atomic blocks*

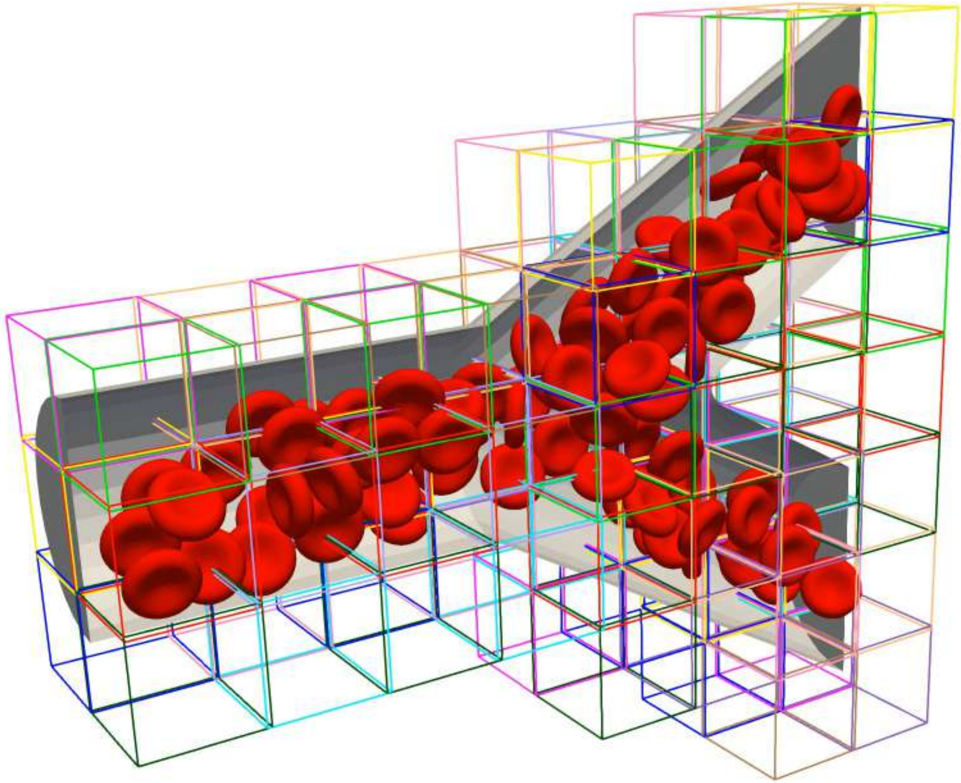
How does it look in the cell simulation?

- Initialise with N atomic blocks
  - Here N = 80
  - Minimal size of atomic block determined by max RBC size
  - We use Palabos functionality to do this

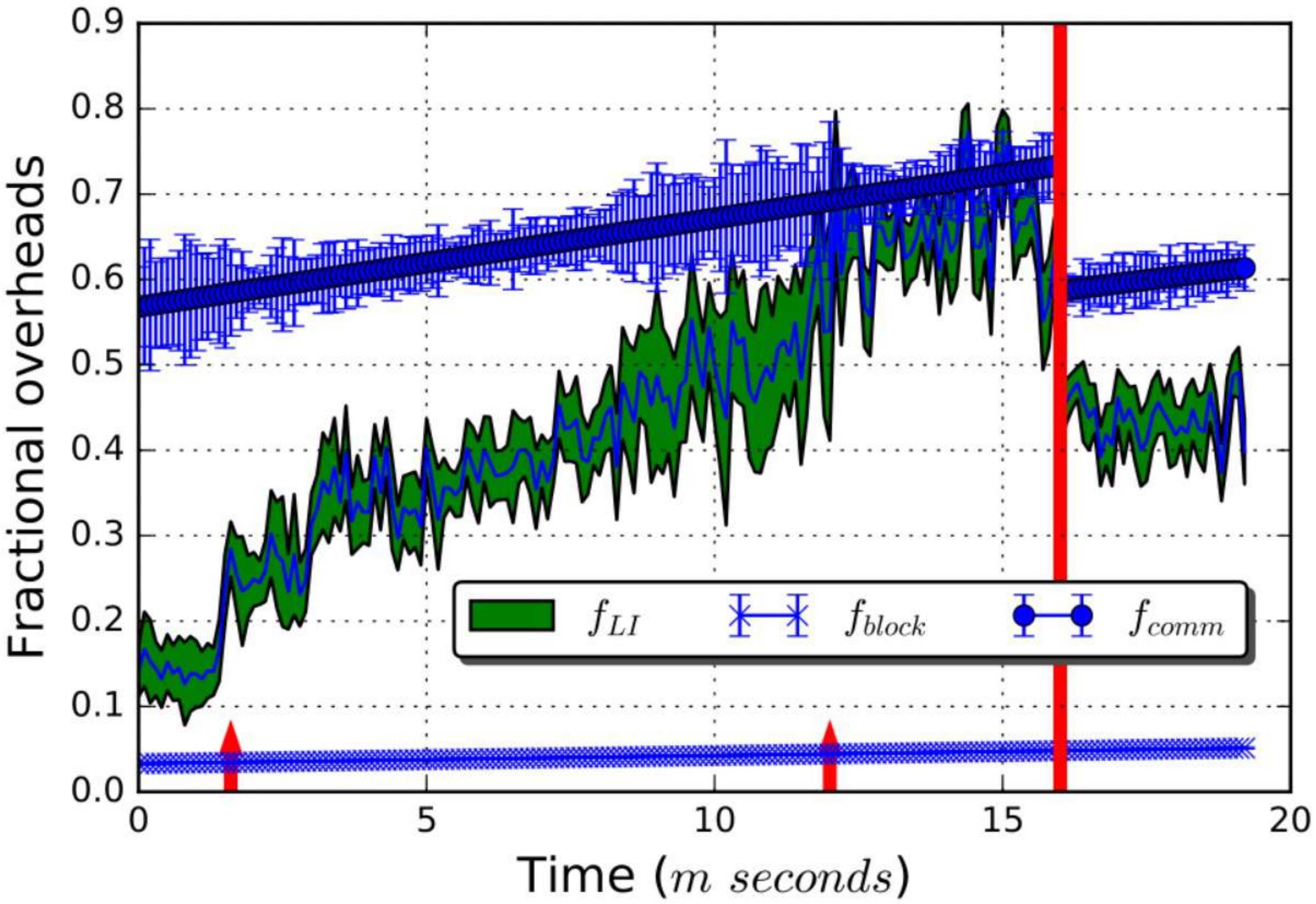
- Load balance over available processors, we give atomic blocks to processors
  - Metis, using weights on the nodes, applying the multilevel KL.
  - Here p = 12

- Merge atomic blocks when possible (when in the same processor) to save on “communication”

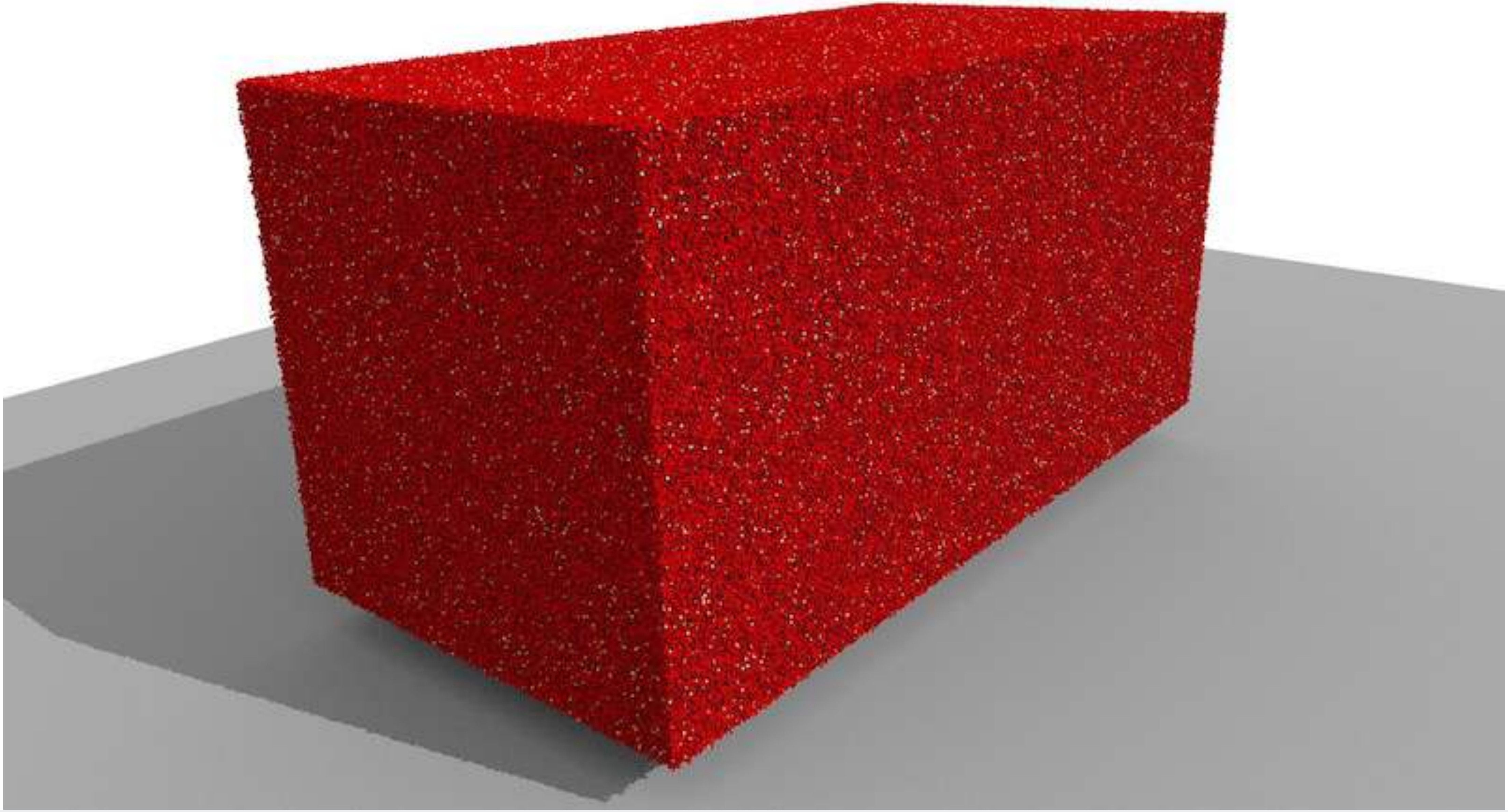
- Then, at run time the load balance changes, so we rebalance every 2000 timesteps



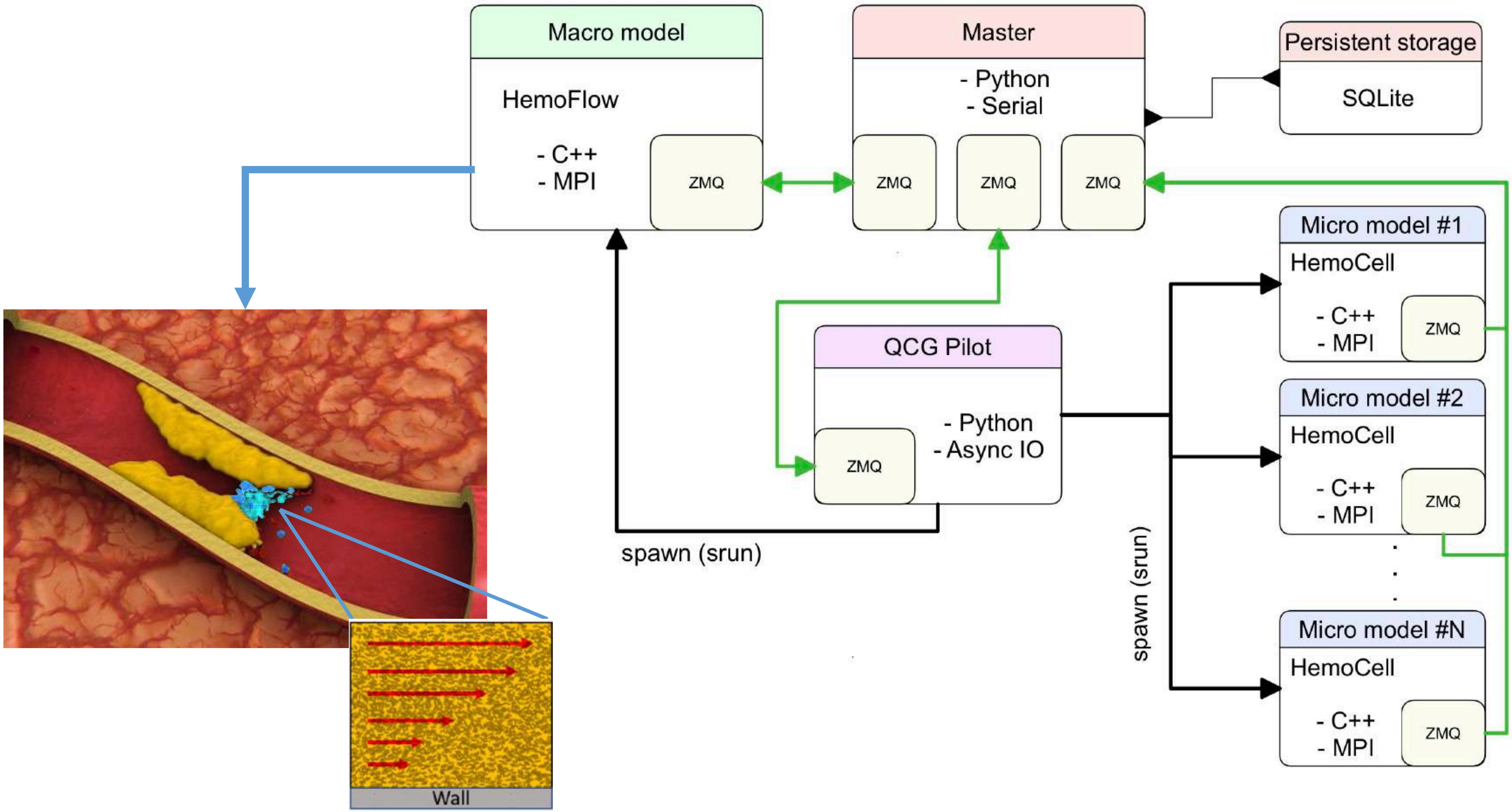
Effects of load-balancing in a large-scale simulation



These techniques make larger domains and more details possible



Multi-scale coupled models (in development)



Team, friends & partners in development

Amsterdam UMC
Universitair Medische Centra

UNIVERSITÉ
DE GENÈVE

COMPAT
Computing Patterns
for High Performance
Multiscale Computing

CompBioMed

1755

PRACE

HDR

Erasmus MC
Universitair Medisch Centrum Rotterdam

SURF

UNIVERSITY OF
OXFORD