Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

A Load Balancing Library for Particle Simulations

ALL

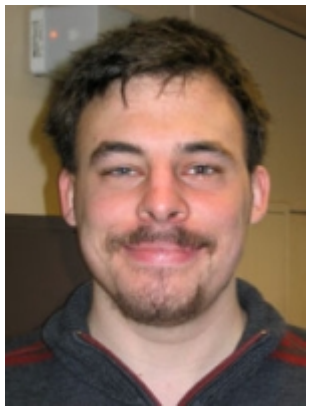
Godehard Sutmann

[g.sutmann@fz-juelich.de]

Institute for Advanced Simulation (IAS)
Jülich Supercomputing Centre (JSC)
Research Centre Jülich

Acknowledgements

- LB developments at Jülich Supercomputing Centre and ICAMS



Rene Halver
(JSC)

Tensor Product
Staggered Grid
LB Library

Jens Freche
(JSC)

Unstructured
Meshes
(Fundamentals)

Marvin Tegeler
(ICAMS)

Phase Field
Models

Carlos Teijeiro
(ICAMS)

LB library

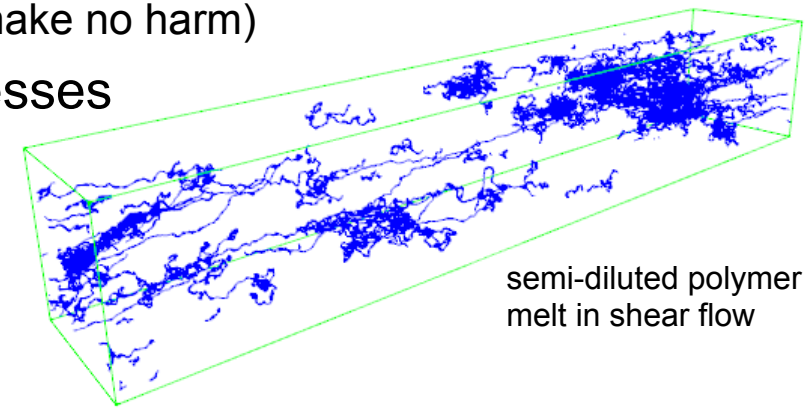
Christoph Begau
(ICAMS)

Unstructured
Meshes
(Implementation)

Load balancing

- Scalability on parallel computers is often limited due to work imbalances on the processors (might be heterogeneous)

- Problem occurs, e.g., if particle distributions within simulation get strongly inhomogeneous
  - some processors might be idle (make no harm)
  - speedup limited by slowest processes



semi-diluted polymer melt in shear flow

- Solutions
  - distribute workload
    e.g. work stealing
    - *larger data-traffic but simple geometries*
  - adjust volume of domains
    - *more complicated geometries and more involved communication patterns*

Load balancing in practice

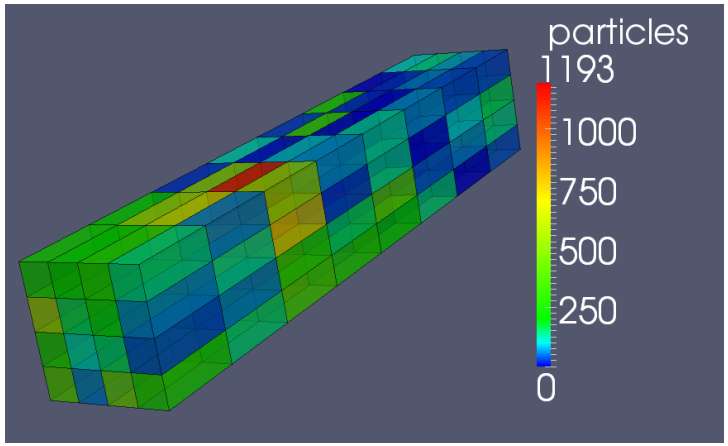
- Performance of homogeneous domain size will be limited by slowest processor

- Speedup is limited by processor with highest deviation of load from average

$$\delta W_i(p) = W_i(p) - \langle W \rangle_P$$

$$\sigma = \frac{P}{1 + P\dfrac{\max_P\{\delta W(P)\}}{W_p}}$$

Amdahl's law for non-homogeneous systems



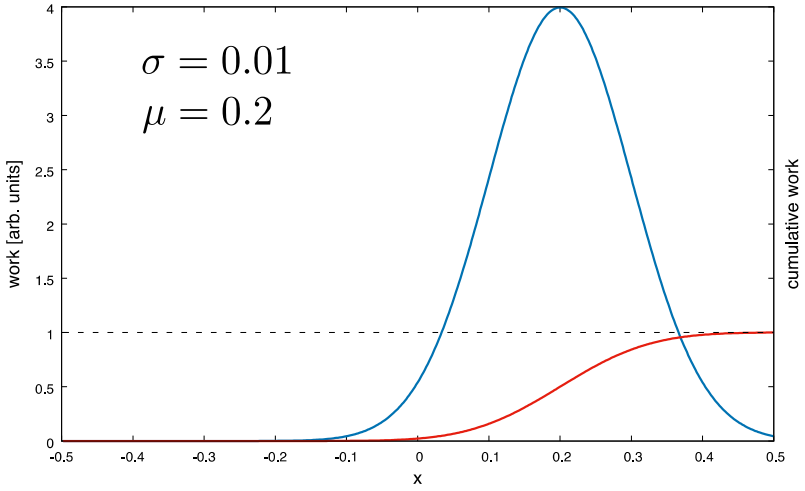
- If characteristic length scale of highest work concentration is close to domain length, linear scaling can be recovered
  - prefactor <1
  - condition: $\delta W_{i,max}(2P) \approx \frac{1}{2}\,\delta W_{i,max}(P)$

if $\max_P\{\delta W(P)\} \propto \frac{1}{P}$

linear scaling is recovered

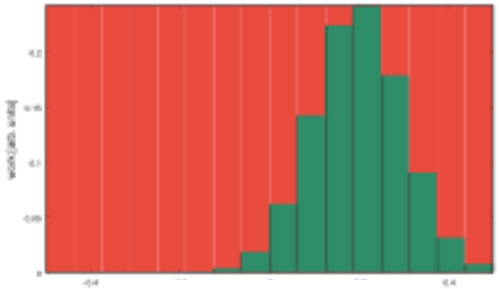
Model for Work Distribution

- e.g. Model problem: 1d-Gaussian distribution of work (or particles)



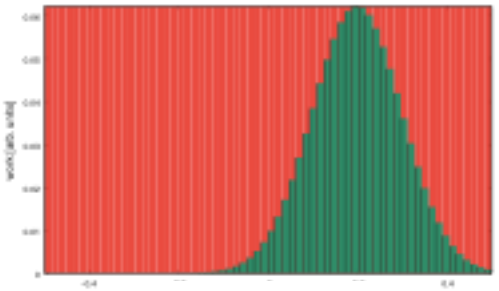
process domain

P = 4



process domain

P = 16

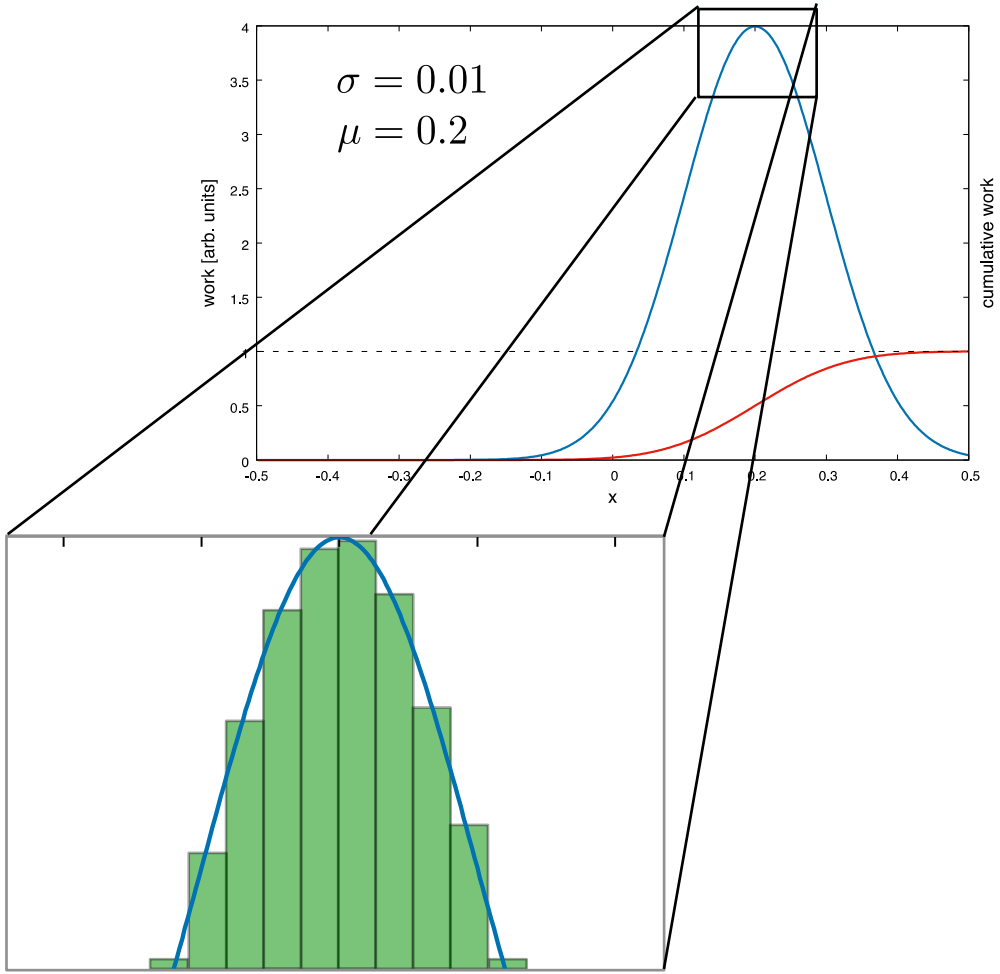


process domain

P = 64

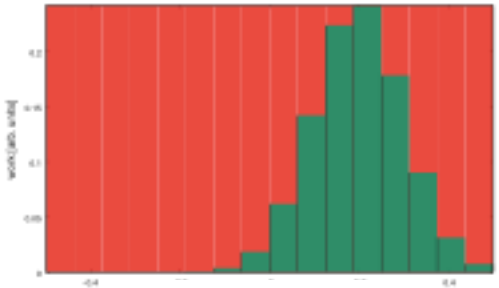
Model for Work Distribution

- e.g. Model problem: 1d-Gaussian distribution of work (or particles)



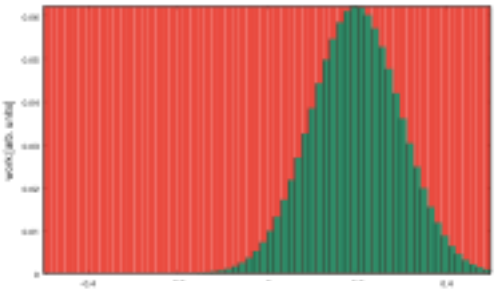
process domain

P = 4



process domain

P = 16

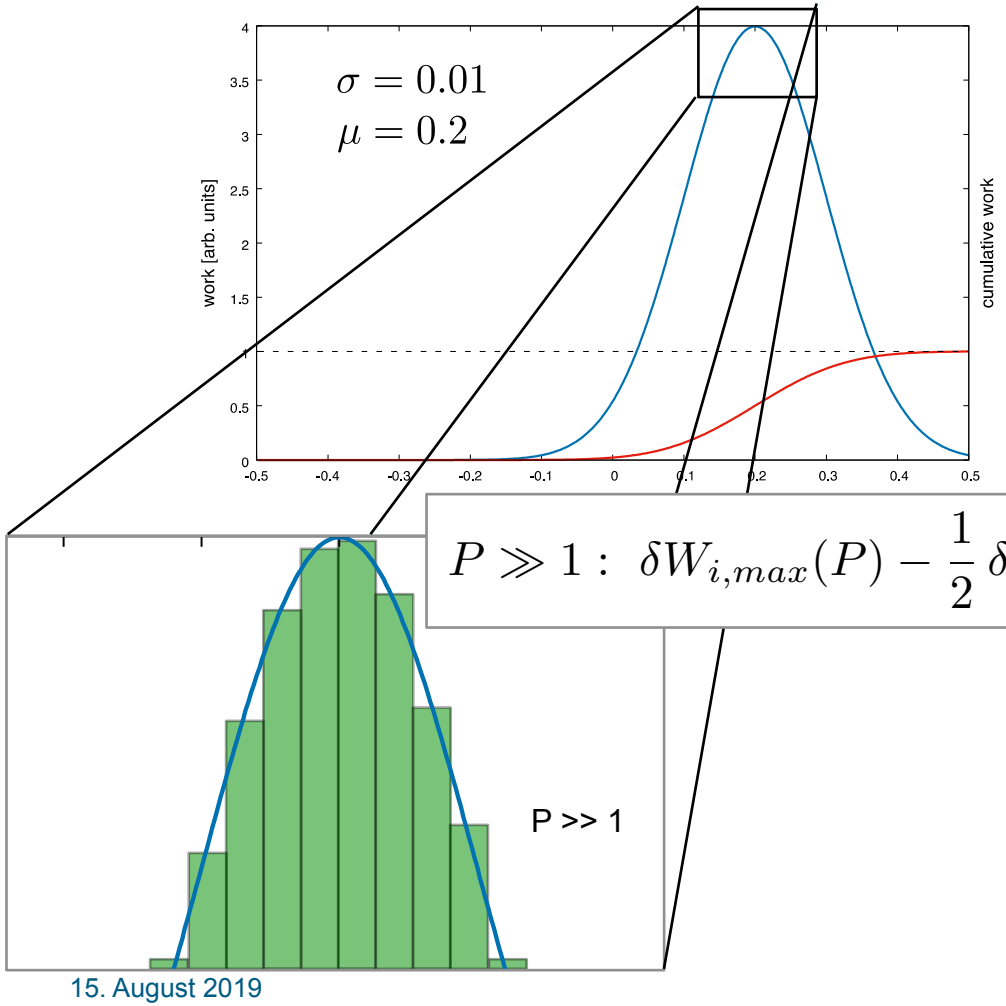


process domain

P = 64

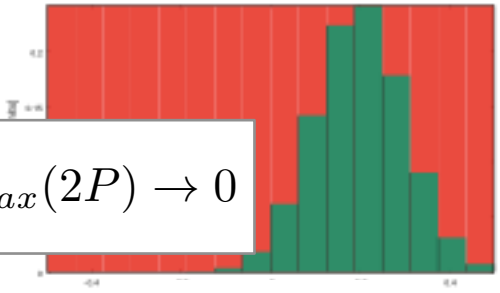
Model for Work Distribution

- e.g. Model problem: 1d-Gaussian distribution of work (or particles)

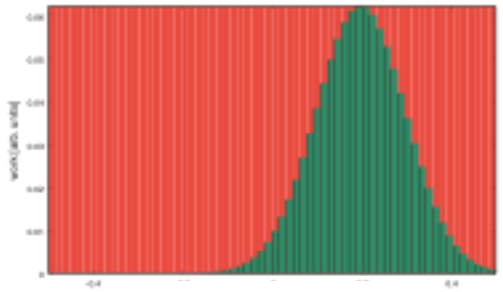


$$P \gg 1: \ \delta W_{i,max}(P) - \frac{1}{2}\,\delta W_{i,max}(2P) \to 0$$

P = 4



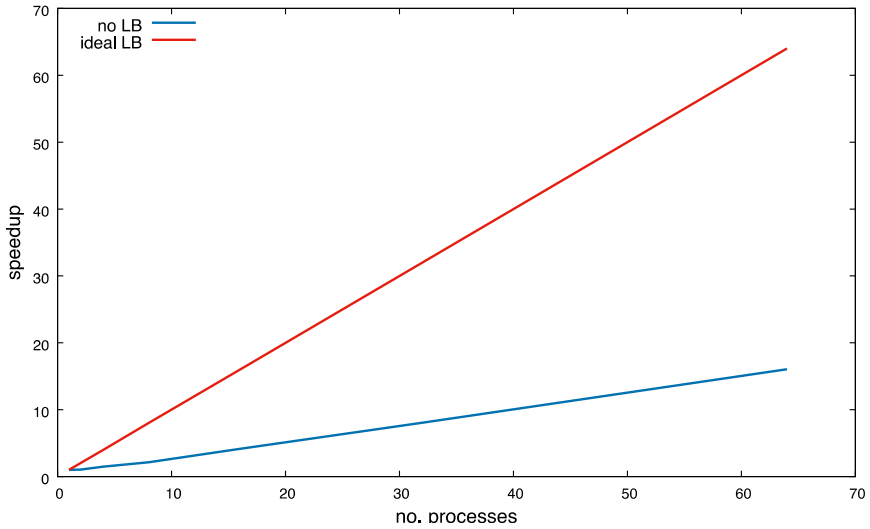
P = 16



P = 64

Uniform domain decomposition

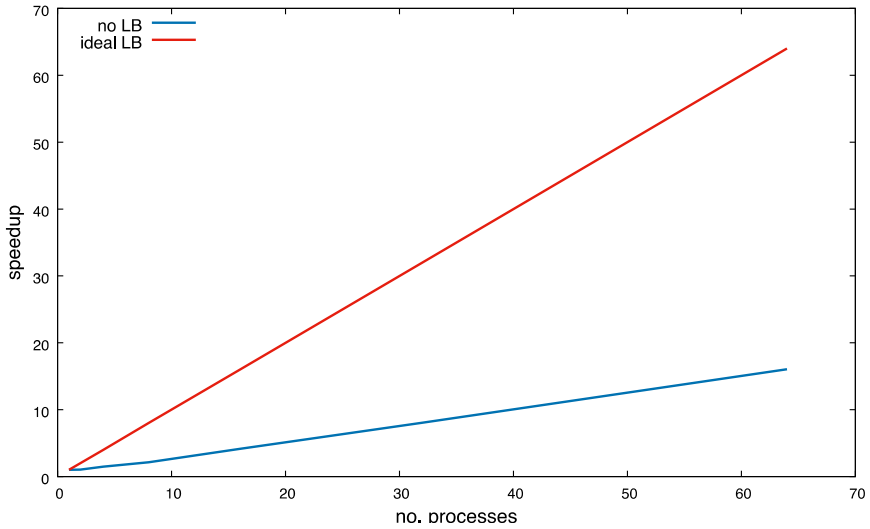
- Consequences for scalability

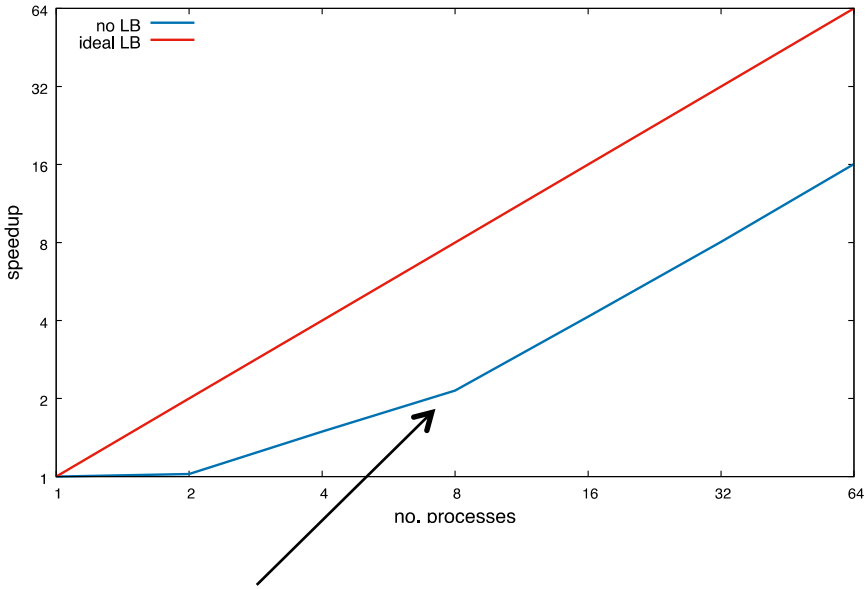


$$\max_P\{\delta W(P)\} \rightarrow \frac{c}{P} \quad (c < 1)$$

Uniform domain decomposition

- Consequences for scalability



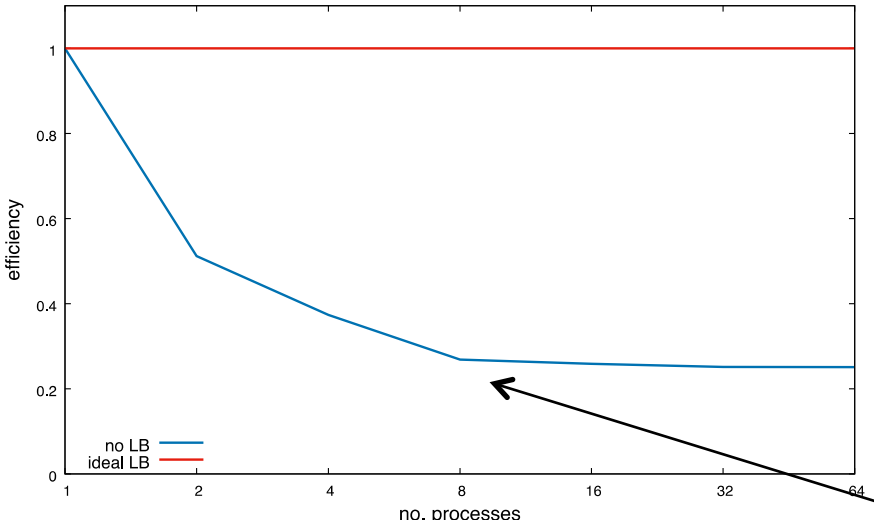


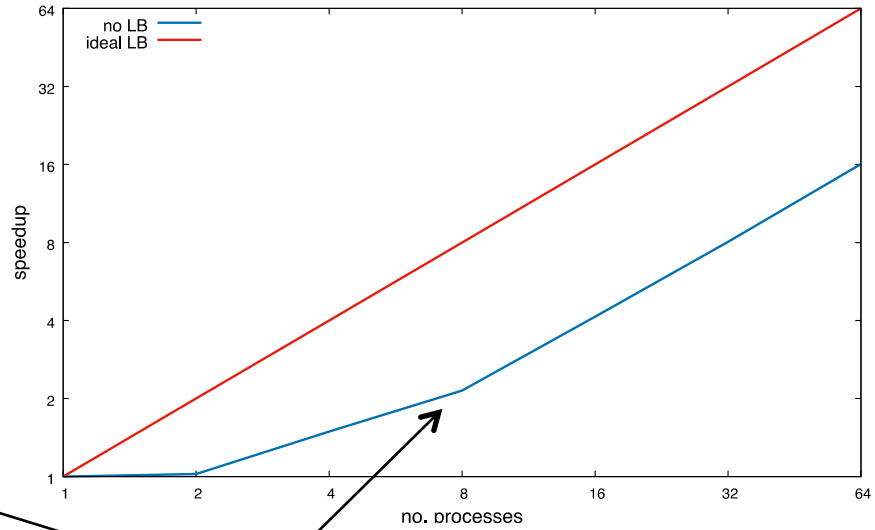
sufficient fine resolution
continue with linear scaling

does not apply for highly dynamic systems

Uniform domain decomposition

- Consequences for efficiency





$$\eta = \frac{P\,T(1)}{\max_P\{T(P)\}}$$

$$T_{i,max}(2P) \approx \frac{1}{2}\,T_{i,max}(P)$$

sufficient fine resolution
continue with linear scaling

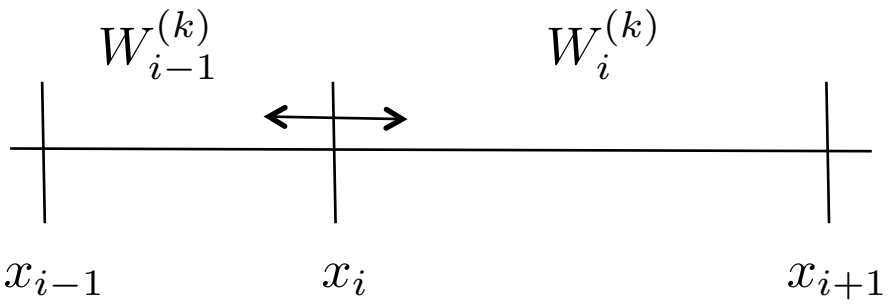
does not apply for highly dynamic systems

Imbalance due to Heterogeneity

- Inclusion of Load Balancing gets highly important for
  - increasing heterogeneity of current hardware architectures
    this becomes an even more important issue for Exascale machines
    - *more involved programming models*
    - *different runtime behaviour of tasks and/or partitions*
  - increasing computational power „invites“ to increase size and/or complexity in simulations, e.g.
    - *multi-physics (different tasks on partitions, different costs of interactions)*
    - *complex geometries*
    - *asynchronous task execution*
- Transition to Exa-Scale needs inclusion of Load Balancing

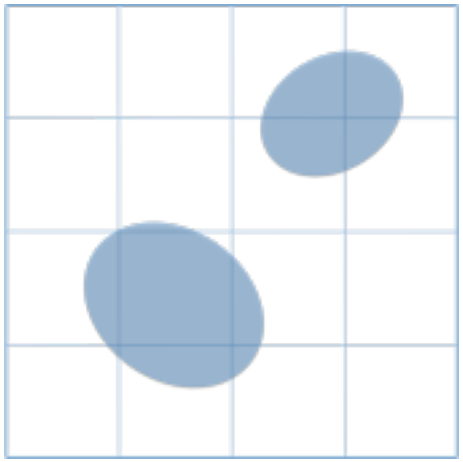
Strategy for load balancing the problem

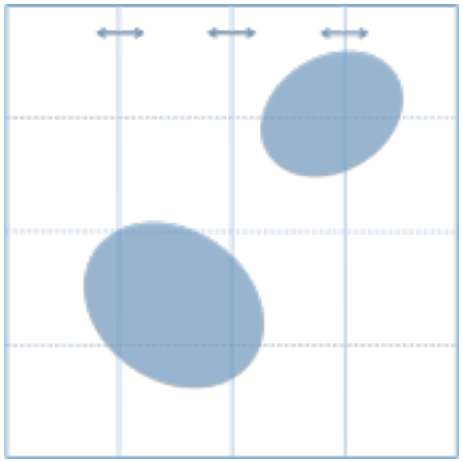
- **Iterative schemes** for best partition size of domains
  - in principle following a master equation approach
  - analogy to PDE solver to find a stationary solution



Load balancing: higher dimensions

- Tensor decomposition





move domain borders independently in each cartesian direction

$$x_i^{\{k+1\}} = \begin{cases} x_0^{\{k\}} & : \quad i = 0 \\ x_i^{\{k\}} + \frac{1}{\gamma} \frac{W_i^{\{k\}} - W_{i+1}^{\{k\}}}{W_i^{\{k\}} + W_{i+1}^{\{k\}}} \left(x_{i-1}^{\{k\}} - x_{i+1}^{\{k\}} \right) & : \quad i \in [1, n-1] \\ x_n^{\{k\}} & : \quad i = n \end{cases}$$

Load balancing: higher dimensions

- Tensor decomposition

$$x_i^{\{k+1\}} = A_x^{\{k\}}\, x^{\{k\}}$$

$$A_x^{\{k\}} = \begin{pmatrix} 1 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ \delta W_1'^{\{k\}} & 1 & -\delta W_1'^{\{k\}} & \cdots & \cdots & \cdots & \vdots \\ 0 & \delta W_2'^{\{k\}} & 1 & -\delta W_2'^{\{k\}} & & & \vdots \\ \cdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \delta W_{n-2}'^{\{k\}} & 1 & -\delta W_{n-2}'^{\{k\}} & 0 \\ \cdots & & & \ddots & \delta W_{n-1}'^{\{k\}} & 1 & -\delta W_{n-1}'^{\{k\}} \\ 0 & \cdots & \cdots & \cdots & 0 & 0 & 1 \end{pmatrix}$$

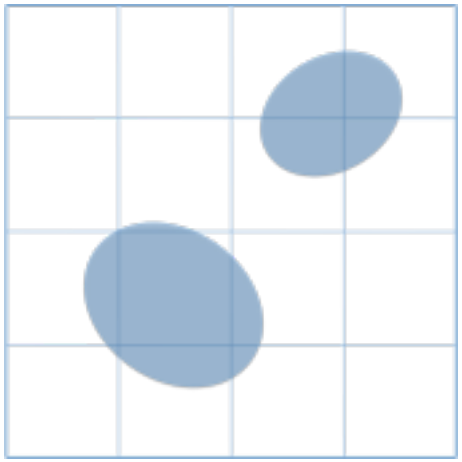
$$\delta W_i'^{\{k\}} = \frac{1}{\gamma} \frac{W_i^{\{k\}} - W_{i+1}^{\{k\}}}{W_i^{\{k\}} + W_{i+1}^{\{k\}}}$$

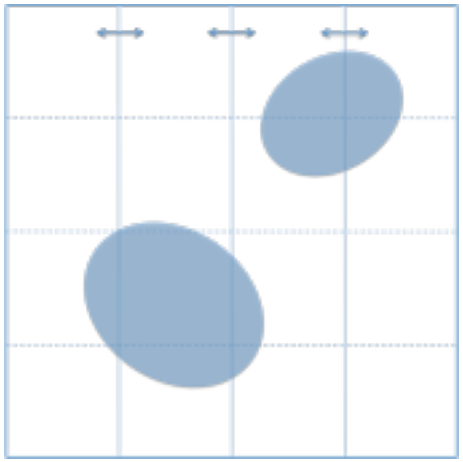
eigenvalue analysis shows that the iteration converges to

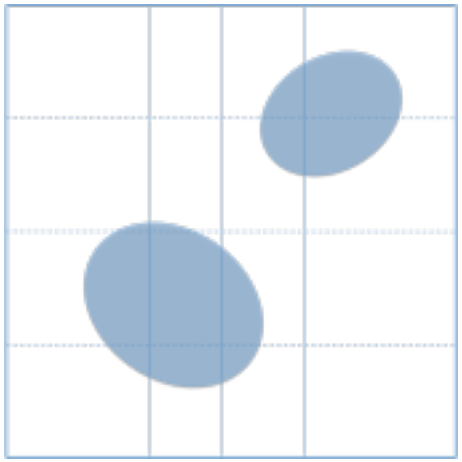
$$\lim_{k\to\infty} W_i^{\{k\}} = \frac{1}{n} \sum_{i=1}^{n} W_i^{\{0\}}$$

Load balancing: higher dimensions

- Tensor decomposition

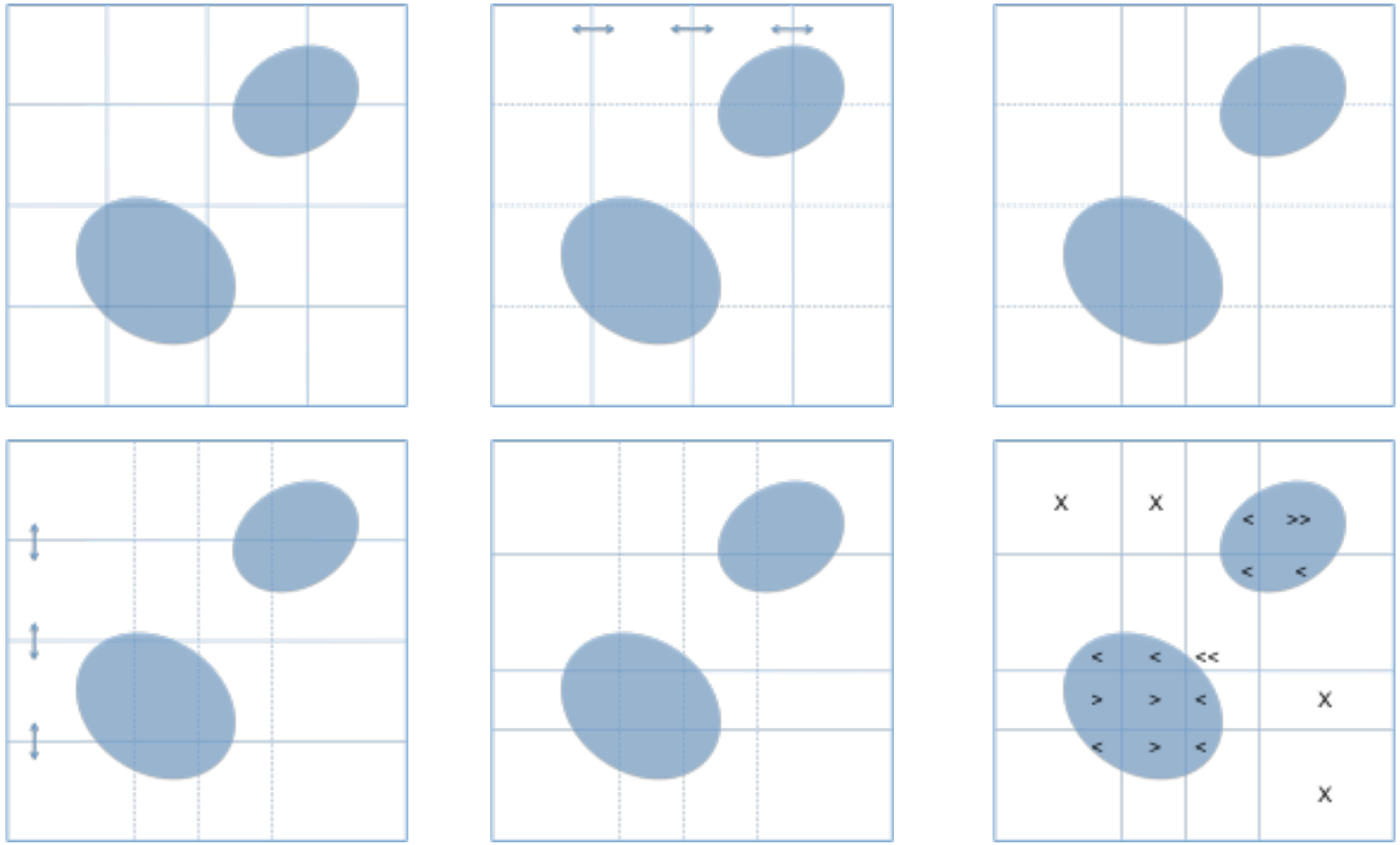






Load balancing: higher dimensions

- Tensor decomposition

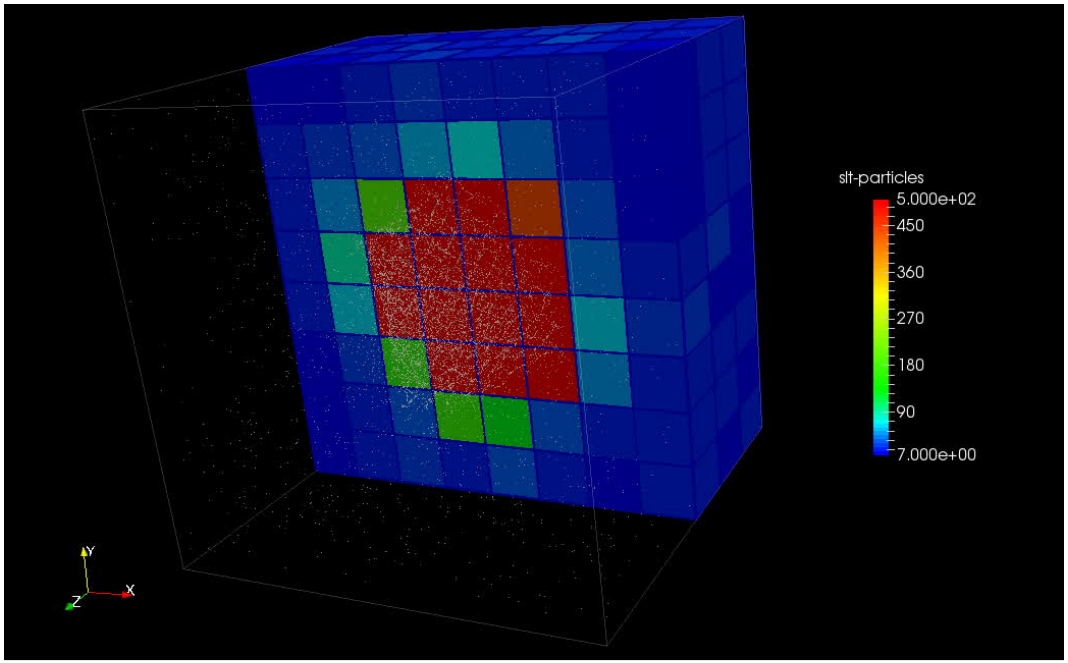


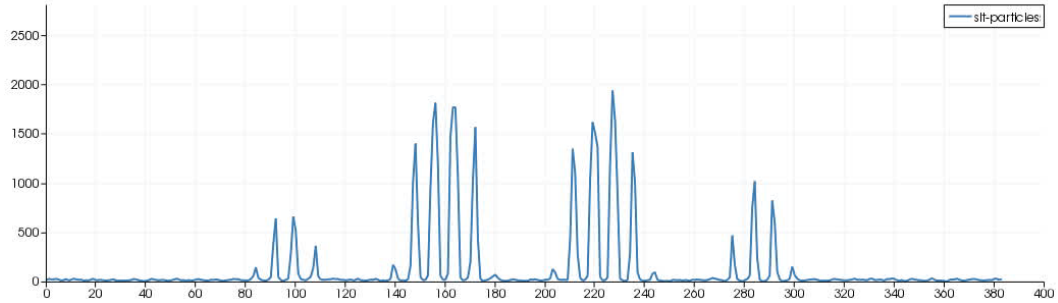
Example: No Load Balancing

- No load balancing

- System:
  - ~50000 MD particles
  - collapsing polymer gel

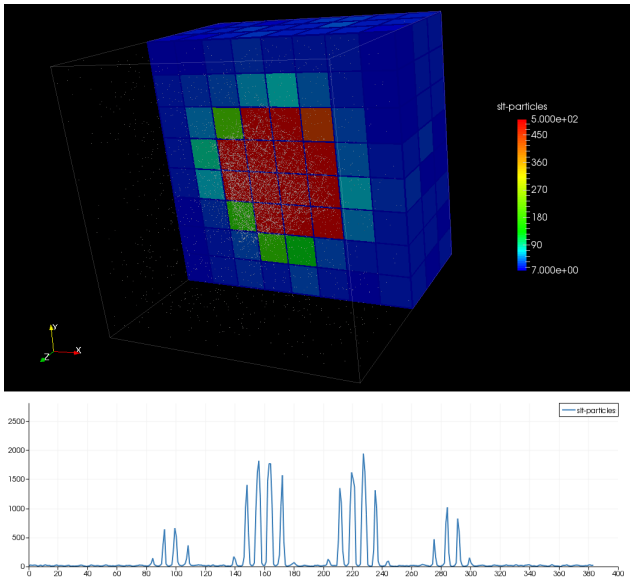
  - strong imbalance due to highly inhomogeneous particles distribution

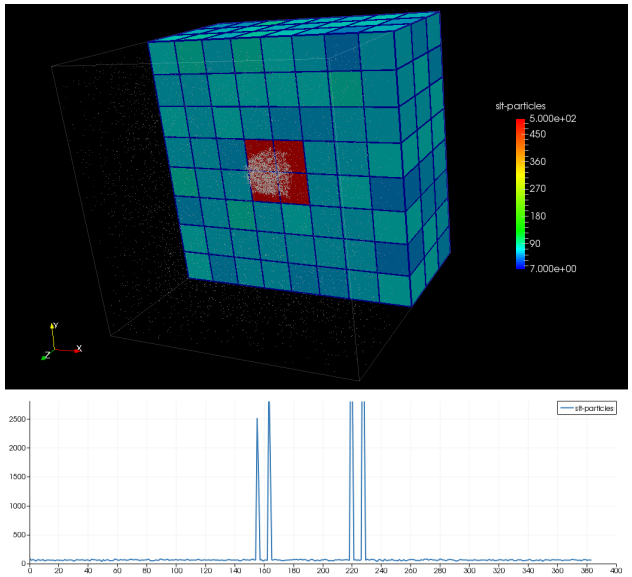




Example: No Load Balancing

- No load balancing
- System:
  - ~50000 MD particles
  - collapsing polymer gel
  - strong imbalance due to highly inhomogeneous particles distribution



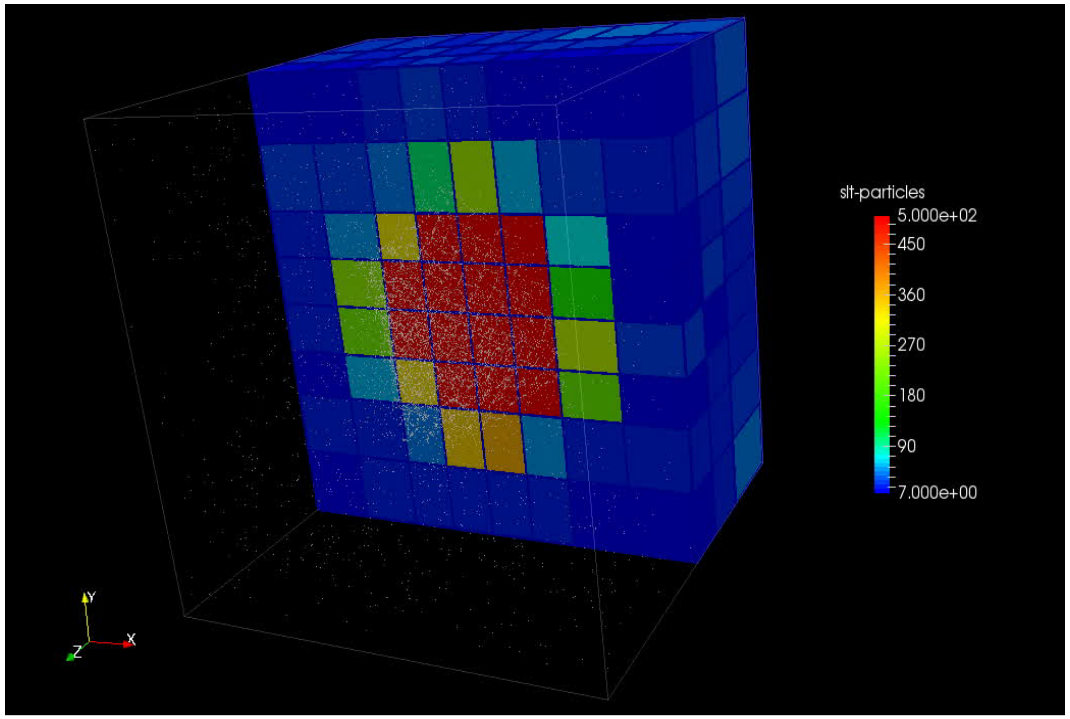


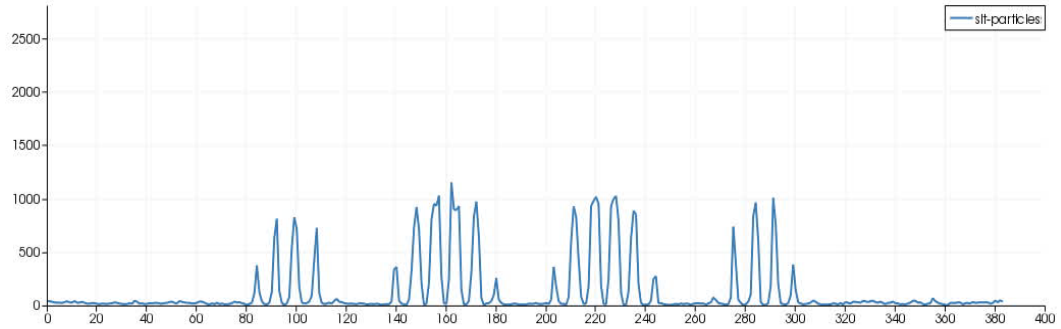
Example: Tensor Product Method

- Load balancing
  - orthogonal decomp.
  - non-staggered

- System:
  - ~50000 MD particles
  - collapsing polymer gel

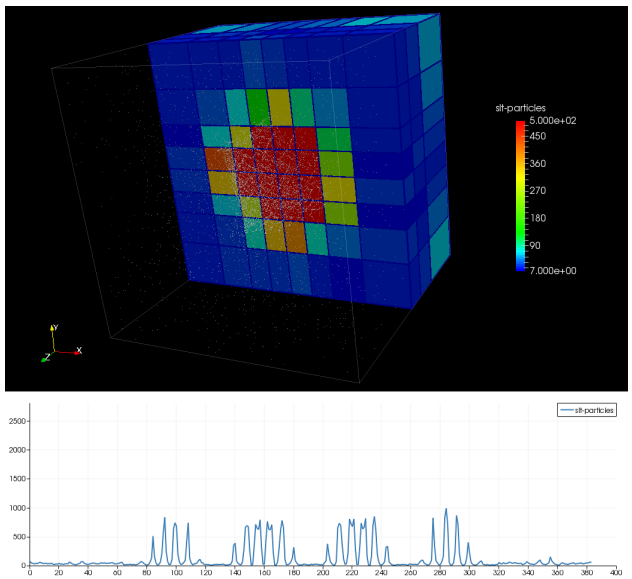
  - strong imbalance due to highly inhomogeneous particles distribution

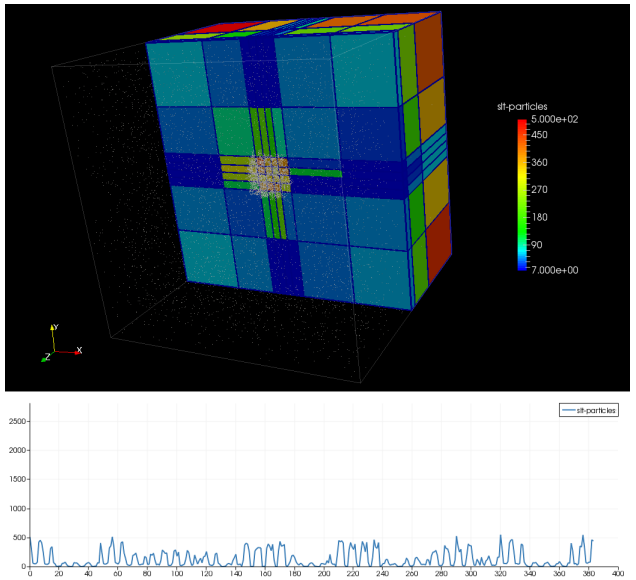




Example: Tensor Product Method

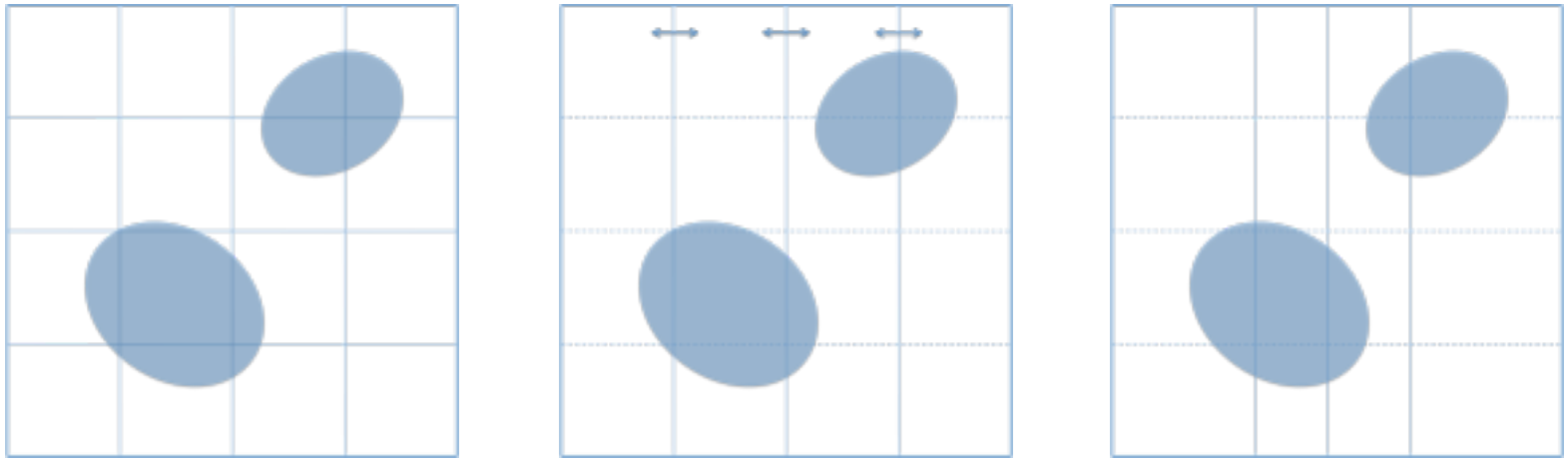
- Load balancing: orthogonal decomposition
- System:
  - ~50000 MD particles
  - collapsing polymer gel
  - strong imbalance due to highly inhomogeneous particles distribution





Improved Load Balancing: staggered mesh

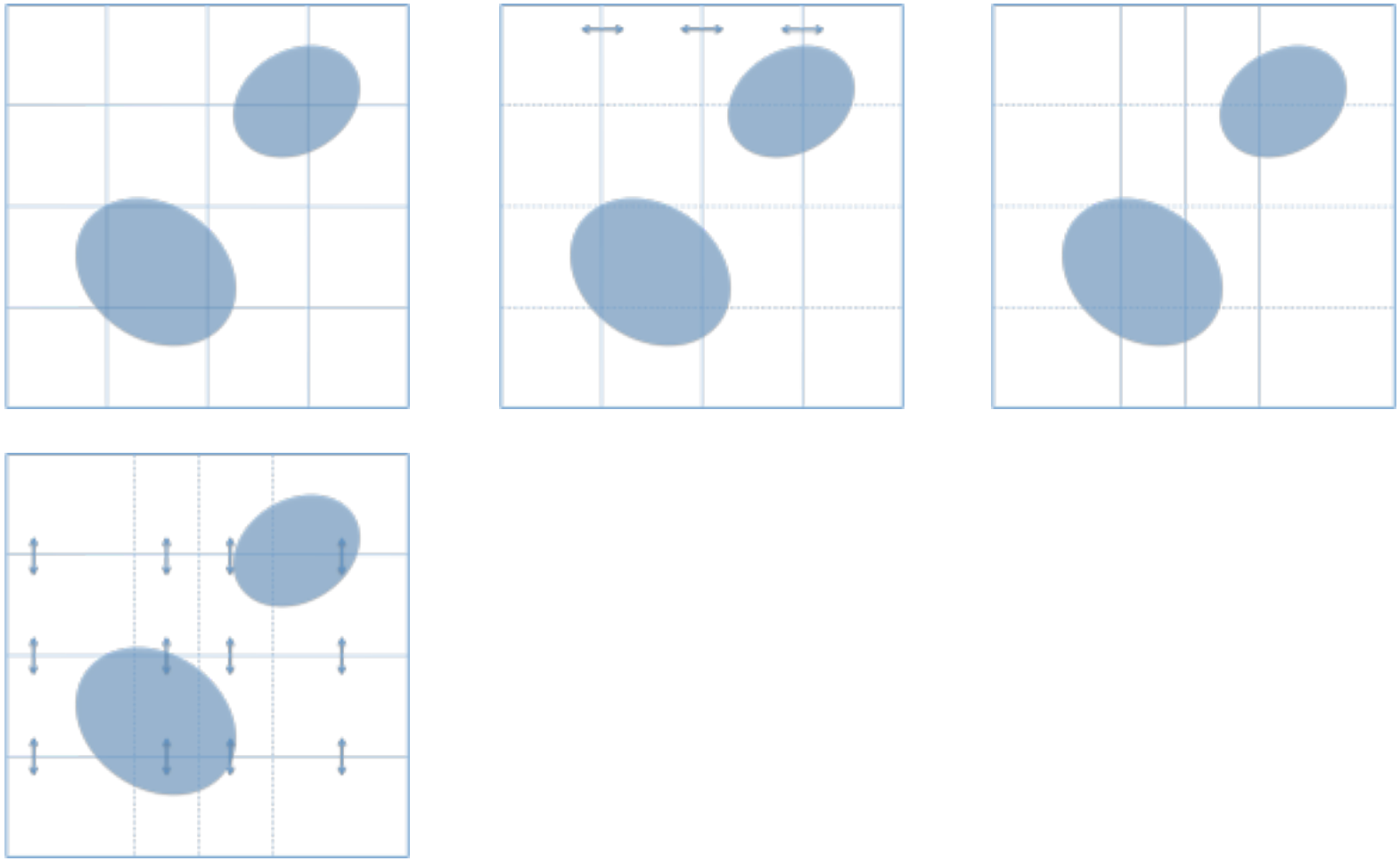
- Improvement should **not** consider dimensions **independently**



Special case of an RCB method

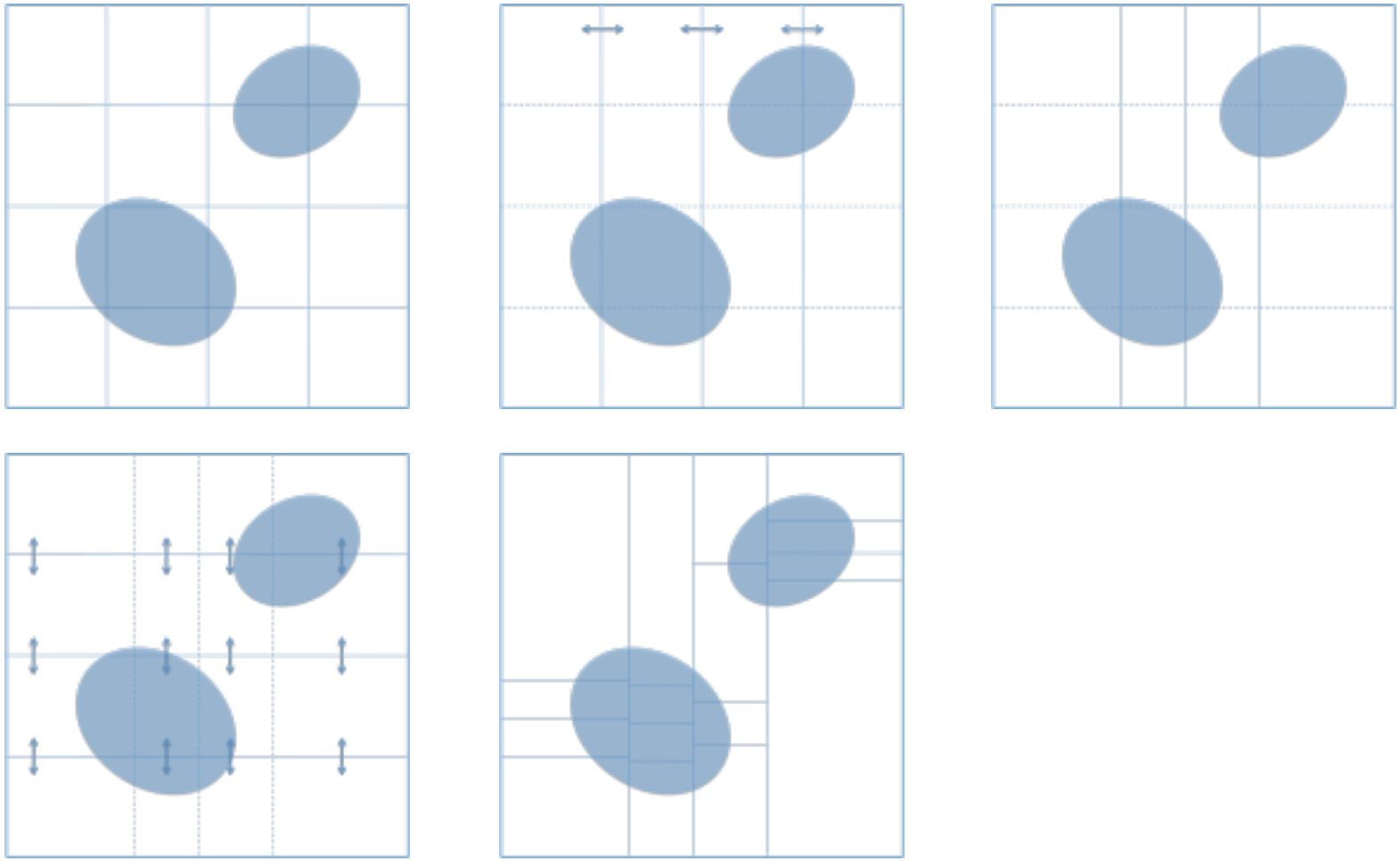
Improved Load Balancing: staggered mesh

- Improvement should not consider dimensions independently



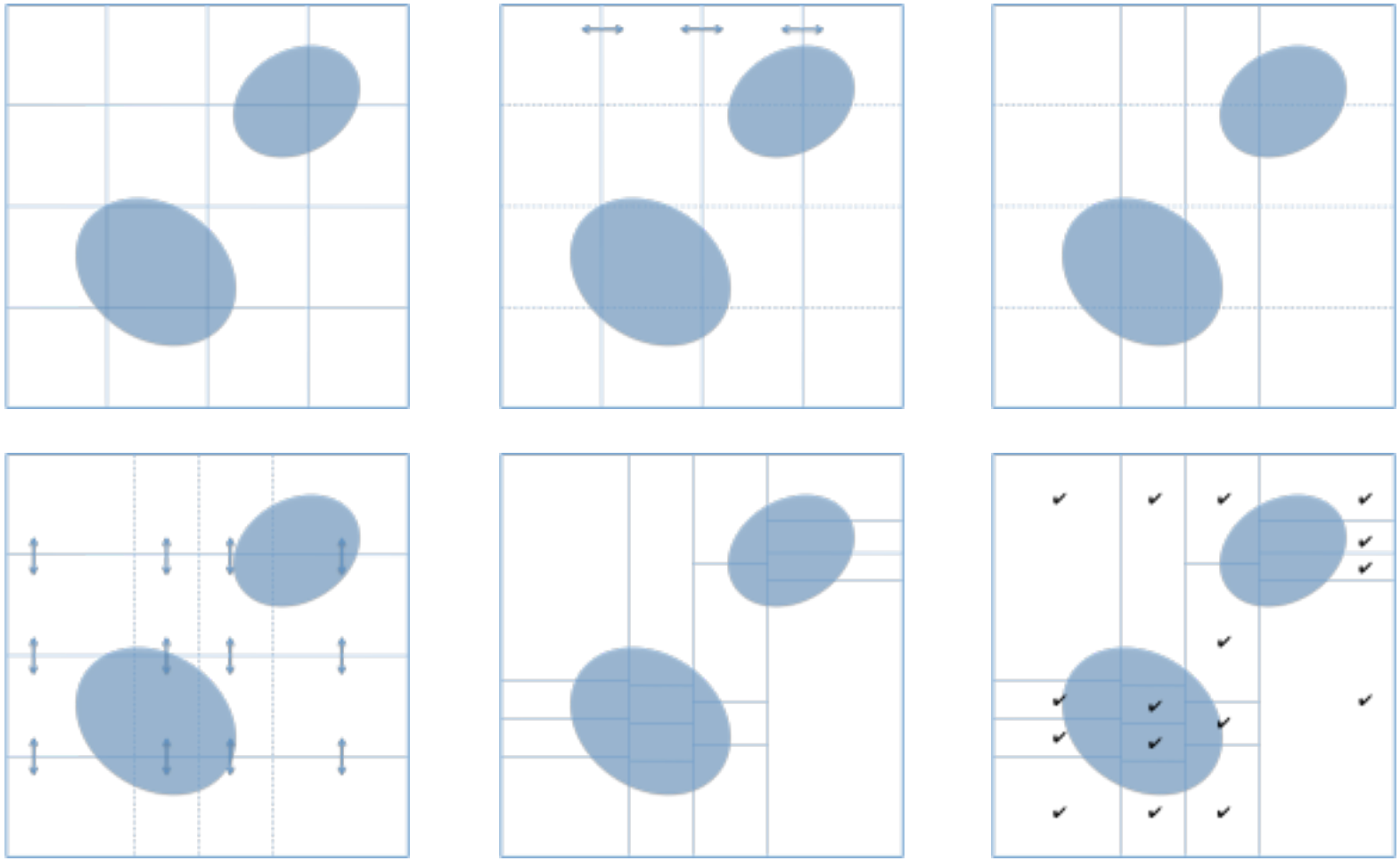
Improved Load Balancing: staggered mesh

- Improvement should not consider dimensions independently



Improved Load Balancing: staggered mesh

- Improvement should not consider dimensions independently

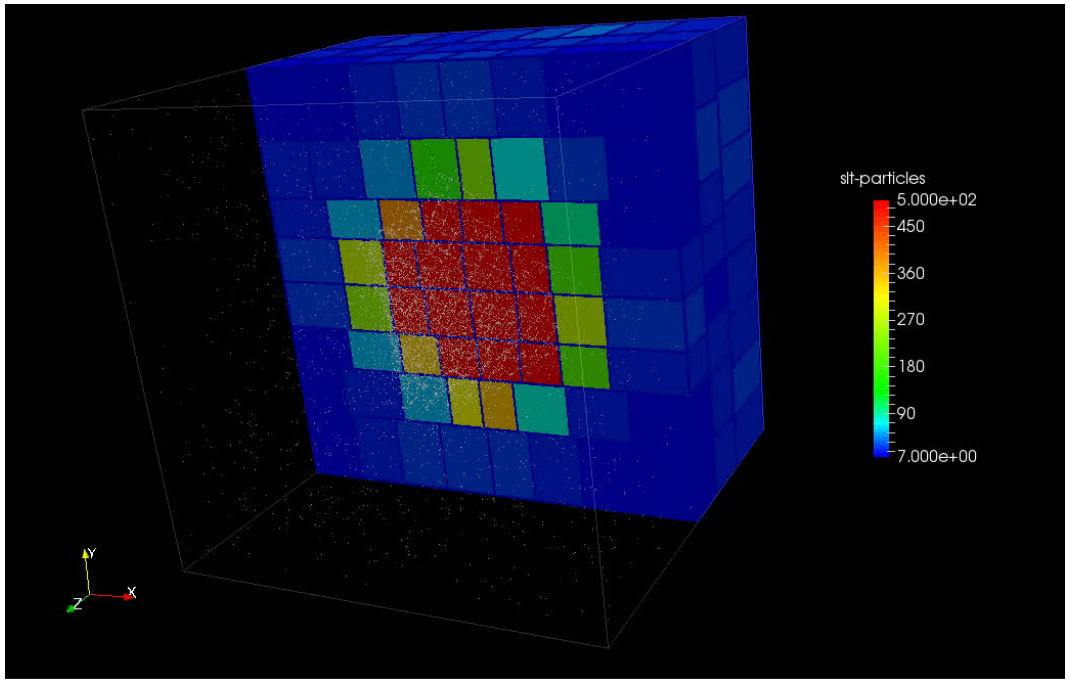


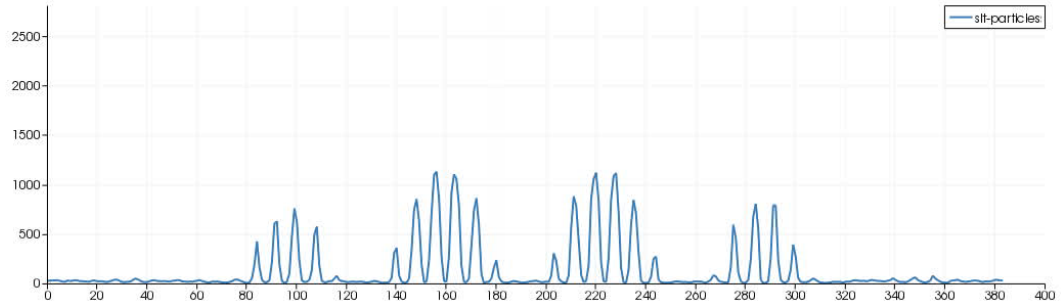
Example: Staggered Mesh Method

- Load balancing
  - orthogonal decomp.
  - staggered

- System:
  - ~50000 MD particles
  - collapsing polymer gel

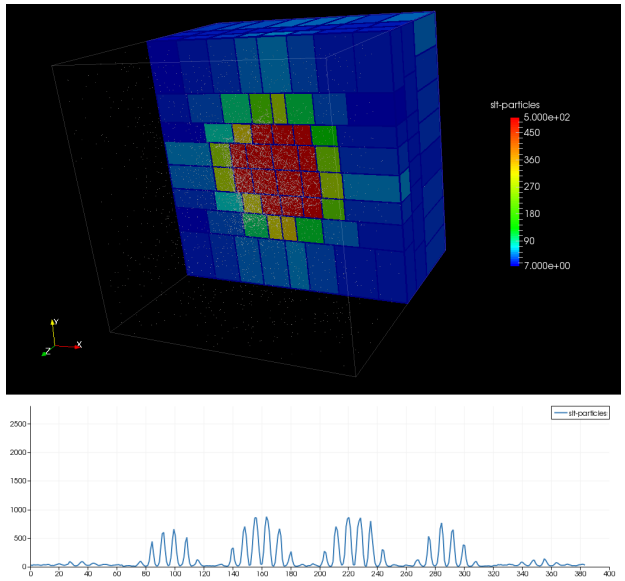
  - strong imbalance due to highly inhomogeneous particles distribution

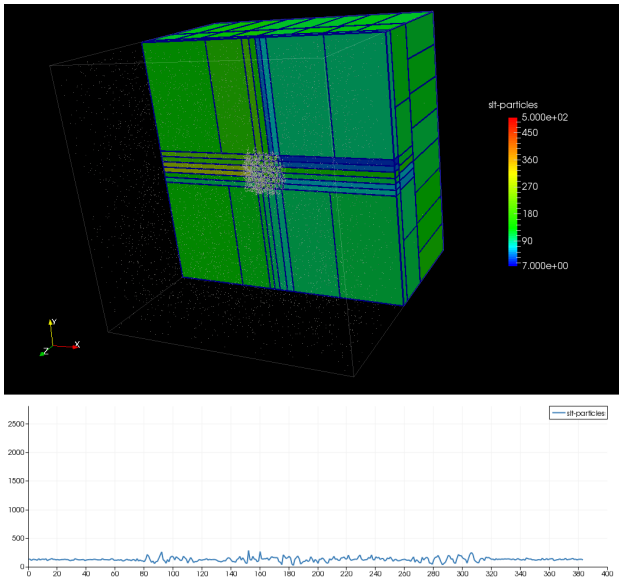




Example: Staggered Mesh Method

- Load balancing: staggered grid decomposition
- System:
  - ~50000 MD particles
  - collapsing polymer gel
  - strong imbalance due to highly inhomogeneous particles distribution





Convergence

- Convergence depending on

$$\delta W_i'^{\{k\}} = \frac{1}{\gamma} \frac{W_i^{\{k\}} - W_{i+1}^{\{k\}}}{W_i^{\{k\}} + W_{i+1}^{\{k\}}}$$

  - relaxation parameter γ
    - ***small*** *γ: large changes in boundaries*
      good for large imbalances <u>but</u> easy to overestimate close to convergence
    - ***large*** *γ: small changes in boundaries*
      slow for large imbalances <u>but</u> smoothly converging

    relaxation should be faster than system dynamics ($\rightarrow$sev. iterat./timestep)

  - number of degrees of freedom to balance
    - *tensor product decomposition*
      $\mathcal{D} = \mathcal{D}_x \otimes \mathcal{D}_y \otimes \mathcal{D}_z \qquad |\mathcal{D}| = (P_x - 1) + (P_y - 1) + (P_z - 1)$
    - *staggered mesh decomposition*

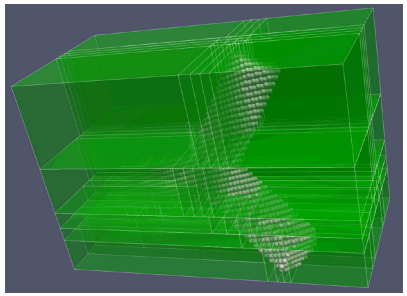
$$\mathcal{D} = \mathcal{D}_x \oplus \mathcal{D}_y[\mathcal{D}_x] \oplus \mathcal{D}_z[\mathcal{D}_y[\mathcal{D}_x]]$$

$$|\mathcal{D}| = P_x - 1 + (P_x - 1)(P_y - 1) + (P_x - 1)(P_y - 1)(P_z - 1)$$

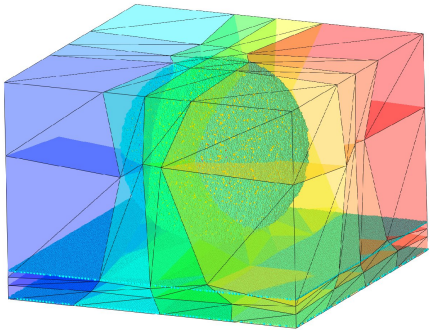
$$= P_x P_y P_z - P_x P_z - P_y P_z + P_x + P_z - 1$$

Library for Load Balancing ALL

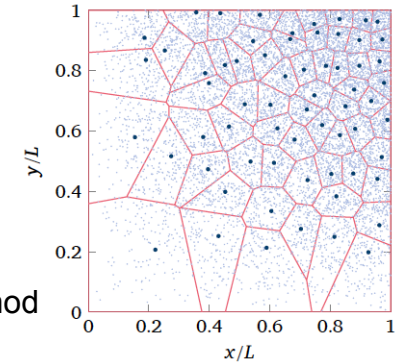
- Methods included in the library
  - orthogonal geometries (LAMMPS)
    - *tensor product decomposition*
    - *staggered mesh*
    - *histogram method*
      (at present for starting configuration)
  - non-orthogonal geometries
    - *topological meshes (vertex based)*
    - *Voronoi cell decomposition*
    - *graph partitioning*



histogram method



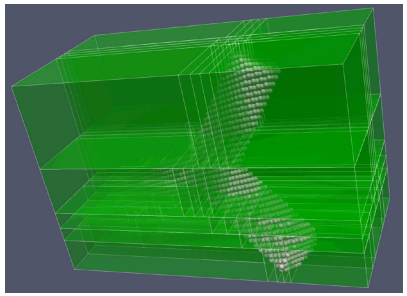
topological mesh method



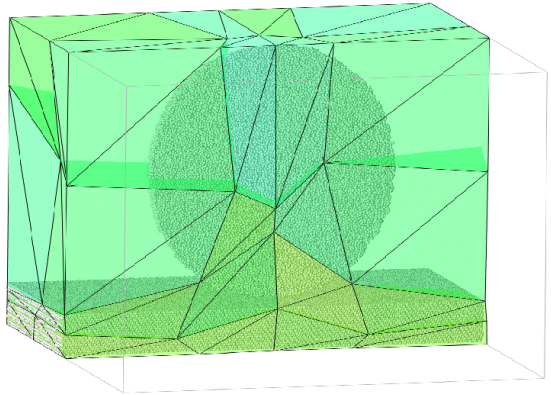
Voronoi method

Library for Load Balancing ALL

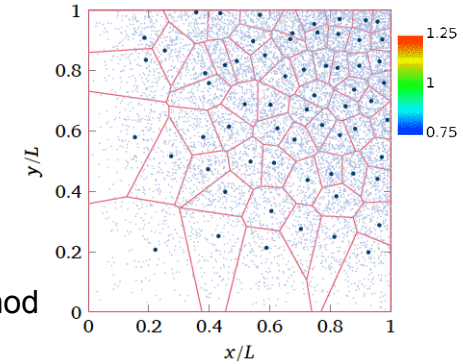
- Methods included in the library
  - orthogonal geometries (LAMMPS)
    - *tensor product decomposition*
    - *staggered mesh*
    - *histogram method*
      (at present for starting configuration)
  - non-orthogonal geometries
    - *topological meshes (vertex based)*
    - *Voronoi cell decomposition*
    - *graph partitioning*



histogram method



topological mesh method



Voronoi method

Load Balancing Library

- Current implementation of the library includes
  - provides a suggestion of an **improved domain decomposition** scheme reducing imbalances
  - based on **iterative scheme**
  - easy to use
    - *low threshold for application developers to apply the library*
    - *depending on the chosen method some adjustments might be needed*
  - provides **selection** of load balancing schemes
  - does not deal with communication of particles / data
    - *done on user level*
    - *does not interfer with administration of simulation code*

Load Balancing Library

- Required input from user:
  - work per process (scalar) – different definitions possible
    - *number of particles, number of interactions, time per particle, ...*
  - current domain decomposition geometry (2x3x8 Bytes)
    - *coordinates of verteces*
  - MPI communicator
    - *Cartesian or provide process grid information*
- Optional input:
  - cell information, e.g. if particle information is cell based (under development)
- Output
  - new domain decomposition
  - (optional): list of cells that entered / left the domain due to new geometry (under development)

Integration (Workshop @JSC in June)

- Codes considered for library
  - DL_Poly
  - DL_Meso
  - ESPResSo
  - ESPResSo++
  - HemeLB
  - IMD
  - MP2C
- First implementations for
  - HemeLB, MP2C, ESPResSo++

Outlook

- Possible integration into LAMMPS (**complementary to existing methods**)
  - Orthogonal methods
    - *requirements: domain boarders flexible*
  - Non-Orthogonal methods
    - *requirements: domain boarders composable by cells*
- Orthogonal iterative methods offer advantage for, e.g., RCB method
  - not each LB step has to be completely reconstructed

DD at core of LAMMPS architecture

Suggestion for cooperation with core developers of LAMMPS