German Research School
for Simulation Sciences

Distributed Octree Mesh Infrastructure for Flow Simulations

Mesh and algorithmic considerations

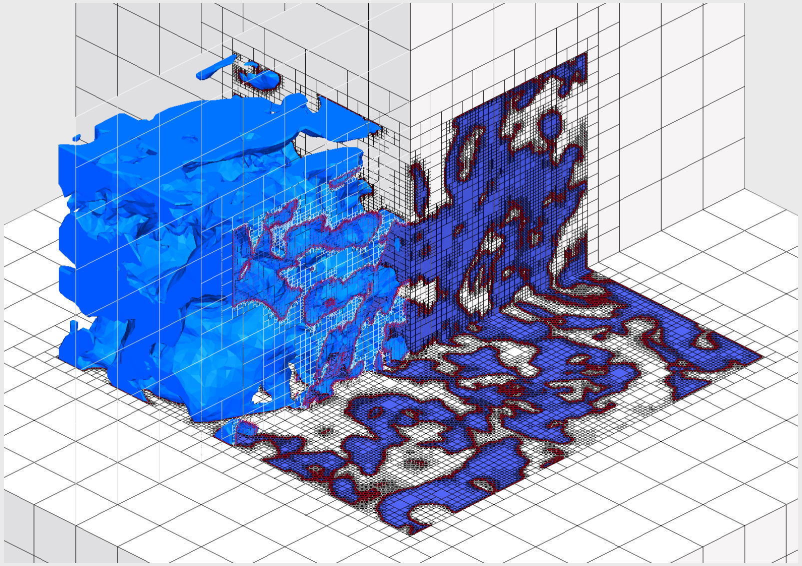
H. Klimach
h.klimach@grs-sim.de
Applied Supercomputing in Engineering

JÜLICH FORSCHUNGSZENTRUM

RWTH AACHEN UNIVERSITY

What we Want to Compute

- PDEs
  - Electrodynamic (Maxwell)
  - Fluiddynamic (Euler, Navier-Stokes)
- Coupled systems
- Arbitrary spatial domains
- Large systems
- Schemes:
  - Lattice Boltzmann
  - Finite Volume with WENO reconstruction
  - Modal Discontinuous Galerkin



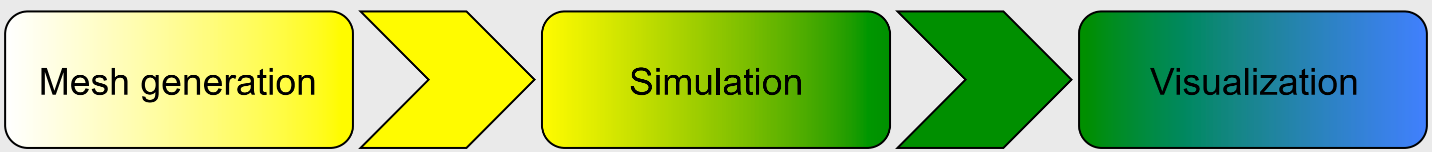
Why we Use an Octree

- Most importantly: compromise between structured cartesian and unstructured irregular meshes
  - Irregular meshes:
    - + arbitrary spatial domains
    - - usually some bottleneck in the parallel processing
    - - complicated for dynamic adaptivity in 3D
  - Structured cartesian meshes:
    - + perfectly distributable
    - + efficient computations, minimal overhead
    - - very rigid, not usable for arbitrary domains
  - Octrees:
    - Have a topology attached, but can represent arbitrary domains
    - Natural path to adaptivity
    - Efficient identification of coupling interfaces

How we Use the Octree

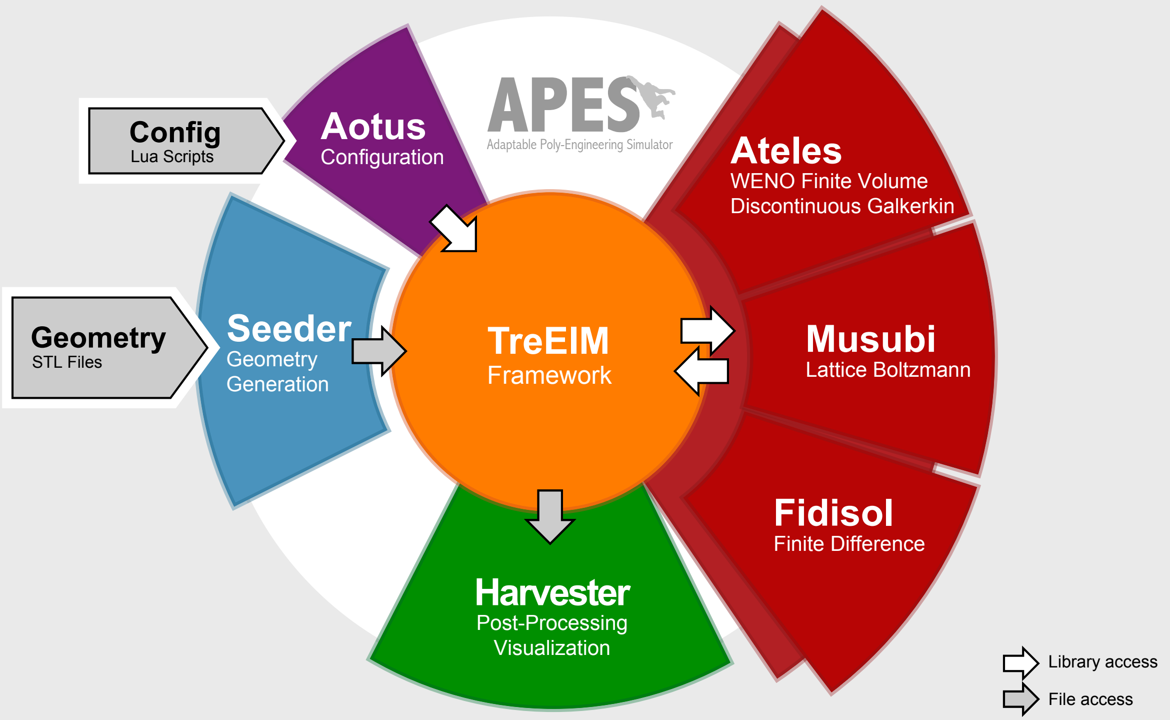
- Use the Octree to directly represent the mesh elements
- Serves as common infrastructure for the complete simulation chain

- The typical three necessary steps are:
  - Mesh generation
  - Simulation on the mesh
  - Visualization/evaluation of resulting data



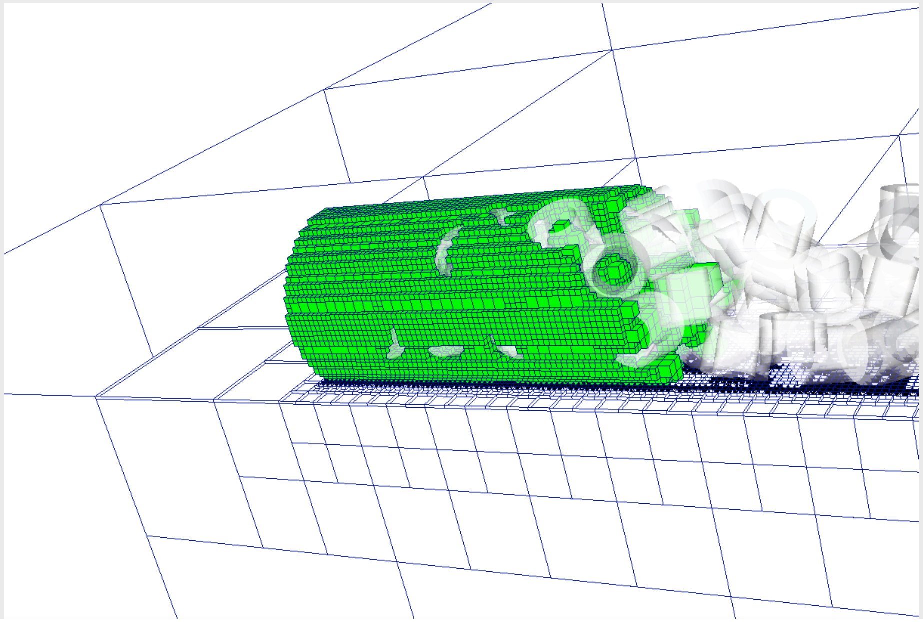
- A bottleneck in any of these steps might prohibit the complete simulation
- A scalable framework has to address all of these steps

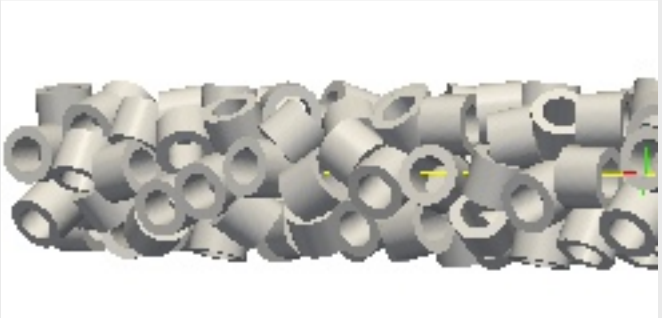
Central Data Structure in Simulation Chain



Sparse Octree Representation

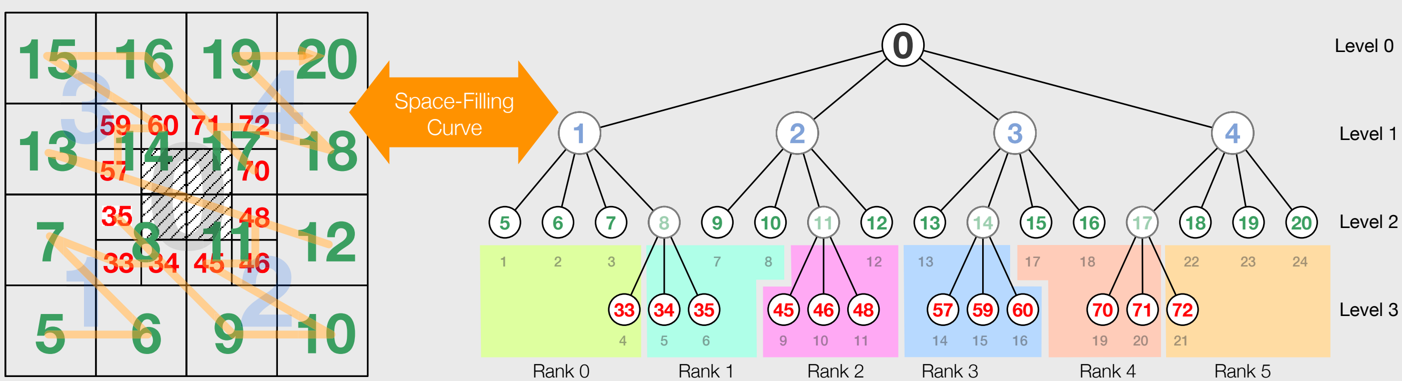
- Only leaf nodes are actually stored, identified by unique treeIDs that describe the spatial position and size of the element.
- Sample mesh with green elements representing the computational domain:





Distribution of the Octree Using SFC

- Octree serialized by space filling curve ordering
- Unique identifcation of elements by breadth-first numbering through the complete tree
- Suitable partitioning by splitting the serialized list of elements
- Very simple format, efficiently read and written in parallel and fully distributed



treeIDlist: Representation on Disk

5	6	7	33	34	35	9	10	45	46	48	12	13	57	59	60	15	16	70	71	72	18	19	20
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

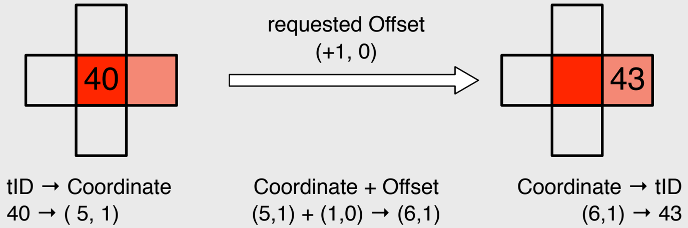
File Format

- 8 byte for treeID and 8 byte for linking additional properties per element
  - 16 bytes for each element + additional data where needed
- Elementwise view on the data allows parallel handling of partitioned mesh
- Additional data distribution can be efficiently found by prefix summation
  - Boundary conditions
  - Mesh deformation
- Redistribution (Load-Balancing) can be achieved by moving splitting positions along the space filling curve
- Meta data stored separately in a Lua script

Connectivity Search

- Done at runtime to account for varying process counts
- Local operation on each process
- Use splitting positions of all processes to decide on remoteness of an element
- Look-up neighbors by their treeID
- Use coordinate offsets to describe required neighbor elements

- Example in 2D, finding the right neighbor of the element with treeID 40 in the quadtree:



- Recursively create halo and ghost elements as needed

Beyond the Mesh

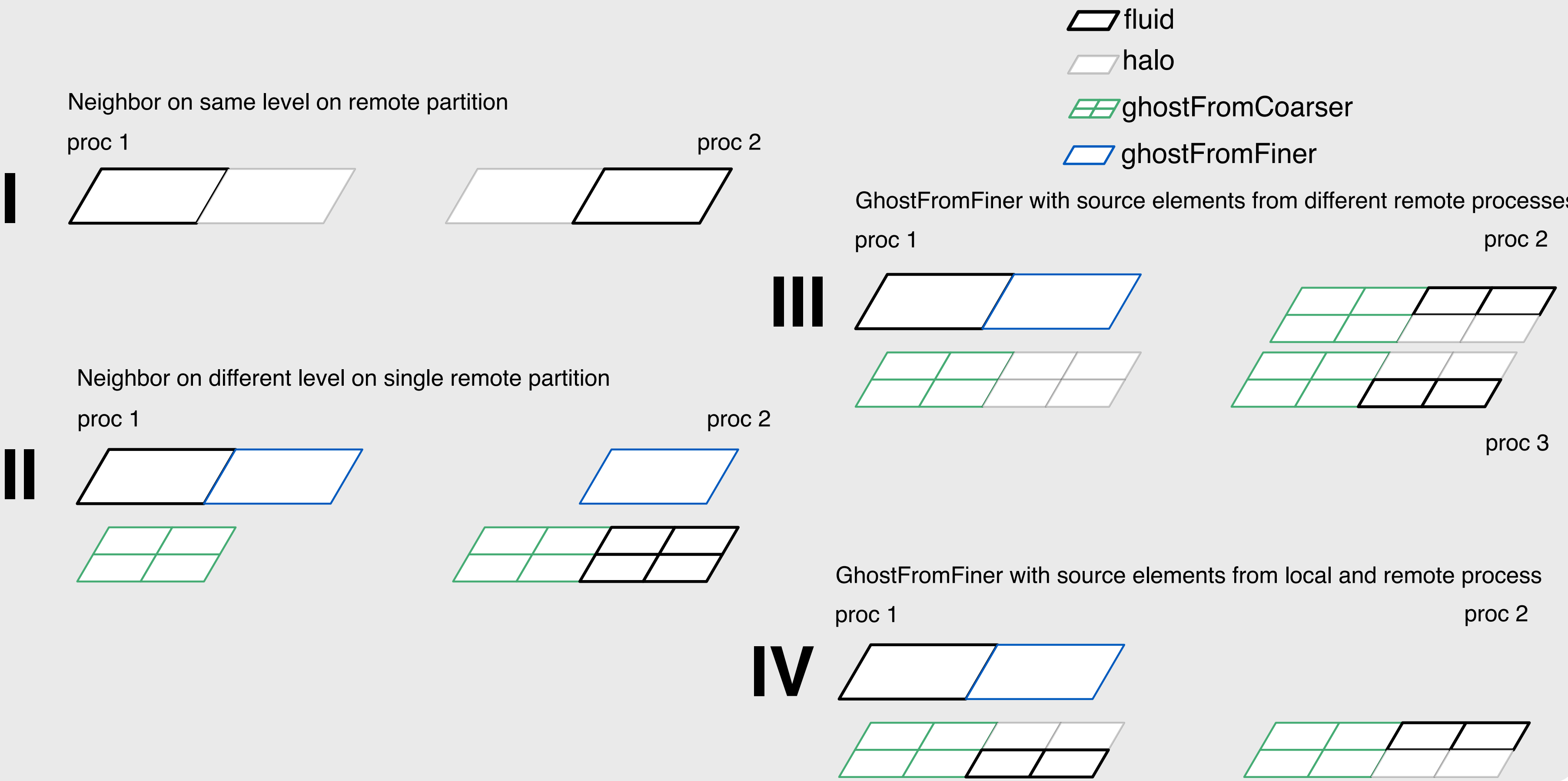
- Flexible configuration with the help of Lua scripts
- Efficient parallel restarting, by using the same elementwise layout on disk for the data as for the mesh
- Arbitrary data extraction by creation of submeshes with communicators attached to them (tracking objects), same formatting as restart
- Visualization by post-processing the restart (or tracking) data
  - Single output format to maintain in solvers
  - Post-processing can be performed separately on dedicated well suited machines

Conclusion

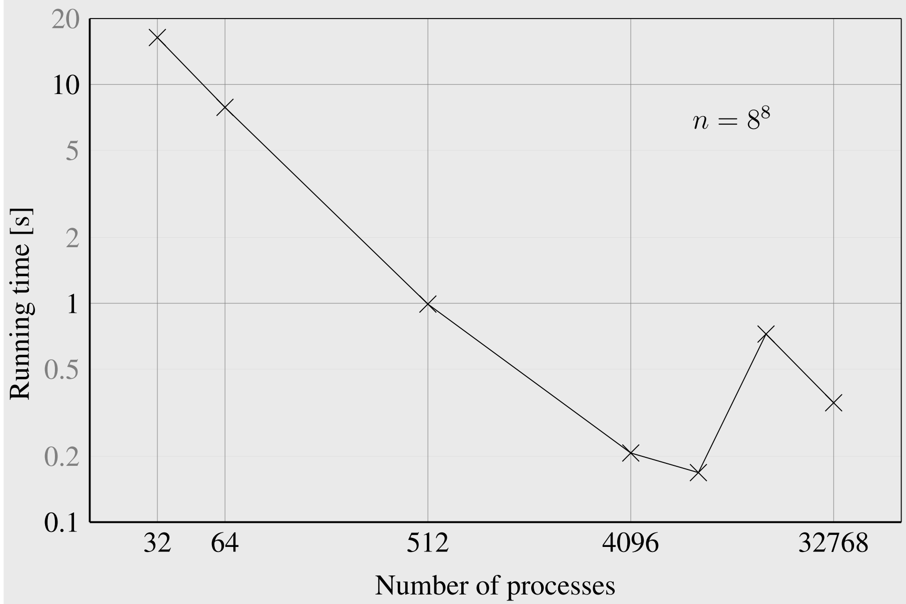
- Advancing simulation techniques for complex large meshes, needs to cope with increased parallelism in computing architectures
- Complete tool chain for the simulation needs to be considered
- Octrees provide a path to represent complex geometries while providing inherent topology information that can be exploited for independent parallel neighbor identification
- Octrees open a natural path to adaptive mesh refinement and coarsening
- With the help of of a space filling curve ordering partitioning can be achieved without further ado.

Thank You for Your Attention!

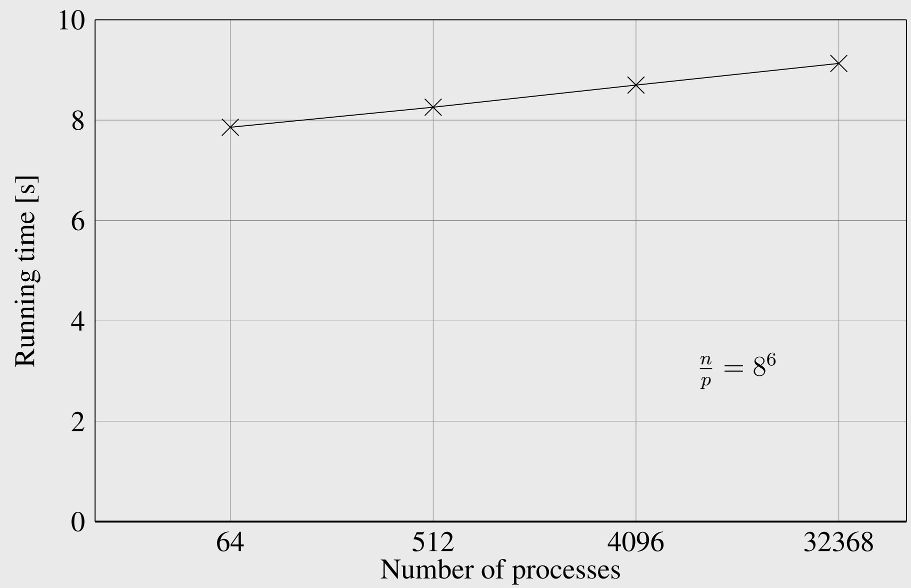
Halo and Ghost Elements



Scaling of Connectivity Search



Strong Scaling on *Hermit*



Weak Scaling on *Hermit*

Hermit: Cray XE6 with AMD Interlagos processors and 32 cores per Node. Scaling with on process per core.

Communication Scheme in the Mesh Initialization

- Distributed parallel implementation using MPI

- Basically required communication steps during initialization:
  - All-gather to collect first and last treeIDs from all partitions
  - All-to-all with a single integer to indicated number of elements to be exchanged between processes
  - Point-to-Point communications for the actual exchange of elemental data, between processes, that need to exchange data