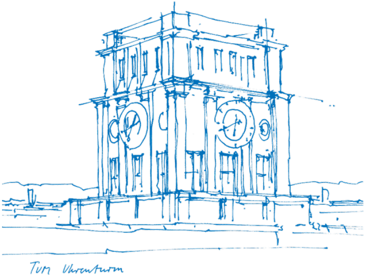
Scalable I/O for the earthquake simulation software SeisSol

Exascale I/O for Unstructured Grids (EIUG)

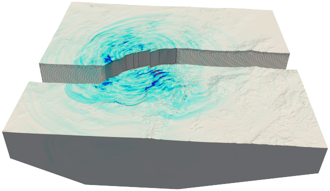
Sebastian Rettenberger
Department of Informatics
Technical University of Munich

25 September 2017



SeisSol

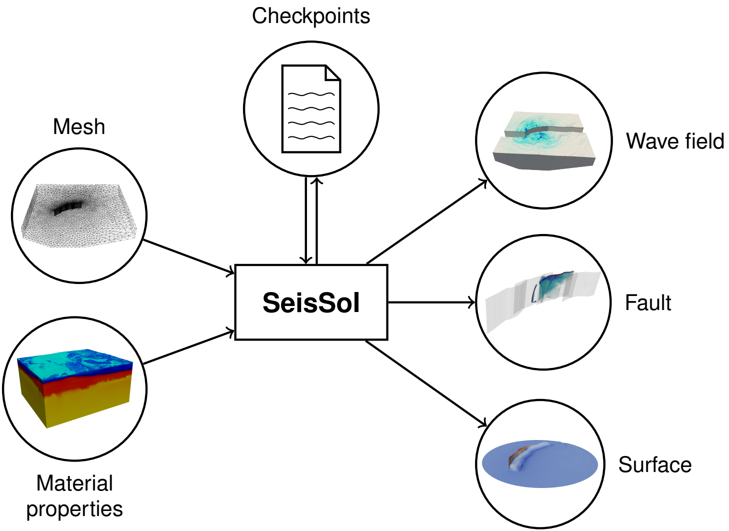
- Seismic wave propagation (incl. attenuation) in 3D and complex heterogeneous media
- Coupled to dynamic rupture simulations
- High order: ADER(time)-DG(space)
- Unstructured tetrahedral meshes
- Gobal and local time-stepping
- MPI/OpenMP hybrid parallelization



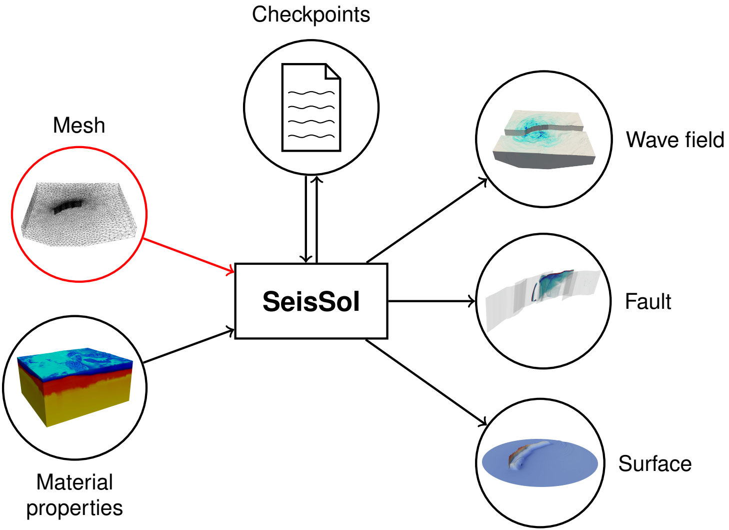
1992 Landers earthquake

- PRACE ISC Award 2014
- Gordon Bell Finalist 2014
- Nominated for the Best Paper Award @ SC17

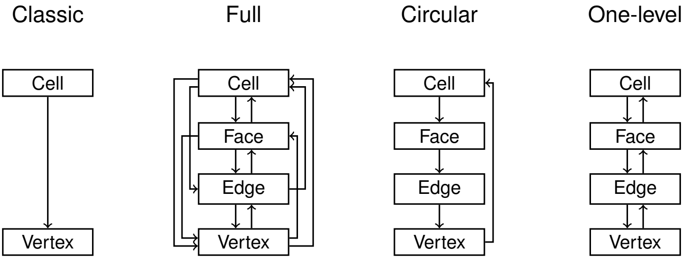
SeisSol: I/O


Checkpoints
Mesh
Wave field
SeisSol
Fault
Material properties
Surface

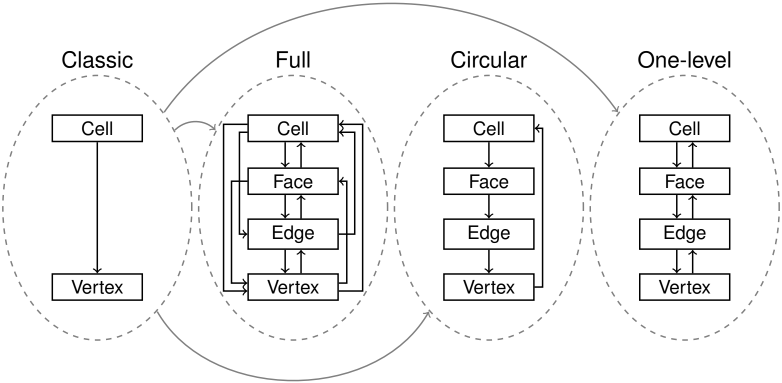
Mesh Initialization



Mesh Representation



Mesh Representation

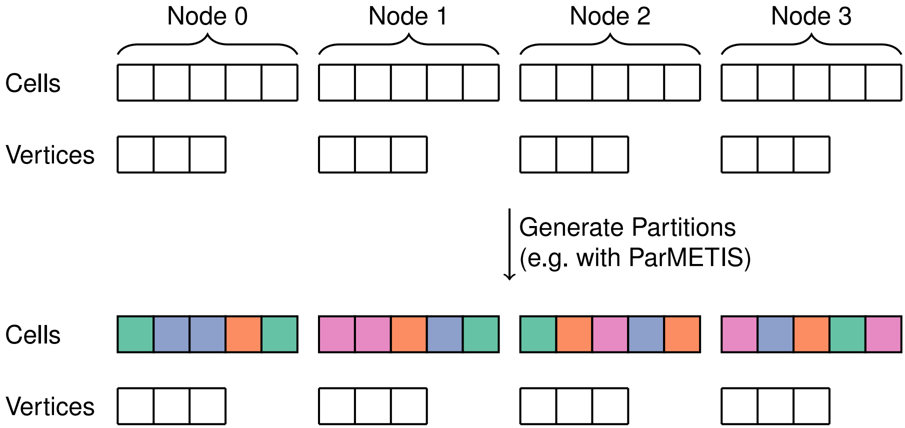


How can we reconstruct (first-level) adjacencies in parallel?

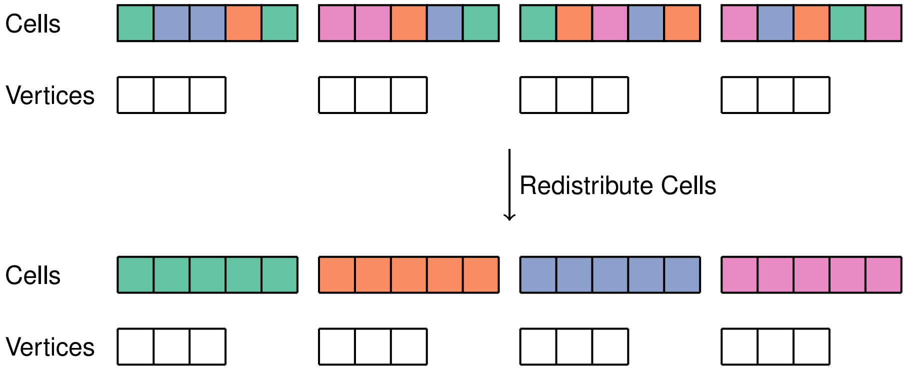
PUML: Parallel Unstructured Mesh Library

- Small header-only C++ library
- Reads **XDMF**/**HDF5** files
- **Partitions** are created at **runtime** (e.g. with ParMETIS) or in a pre-processing step
- The application gets **global IDs**, first-level **adjacencies** and faces/edges/vertices on **partition boundaries**
- Open source: `https://github.com/TUM-I5/PUML2`

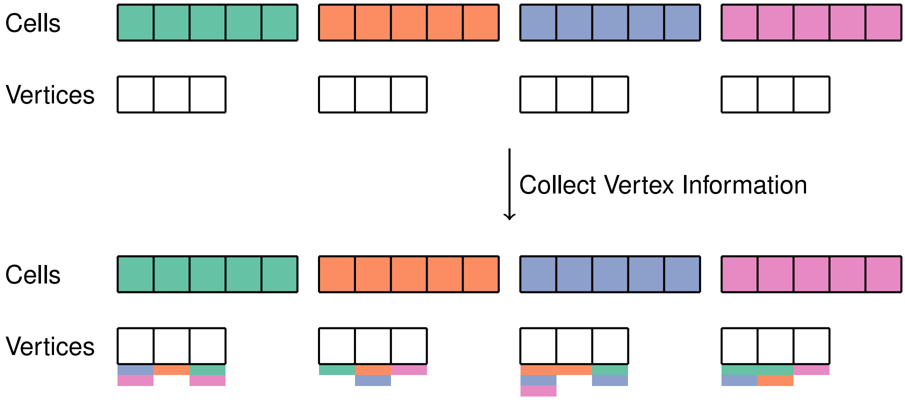
Reconstructing Adjacencies (1)



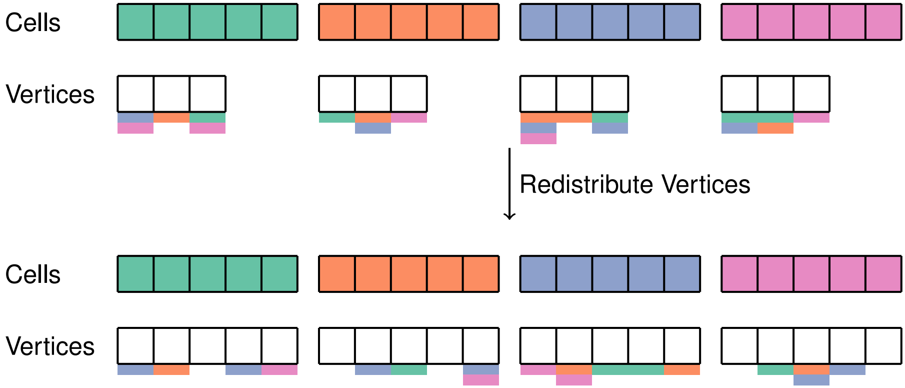
Reconstructing Adjacencies (2)



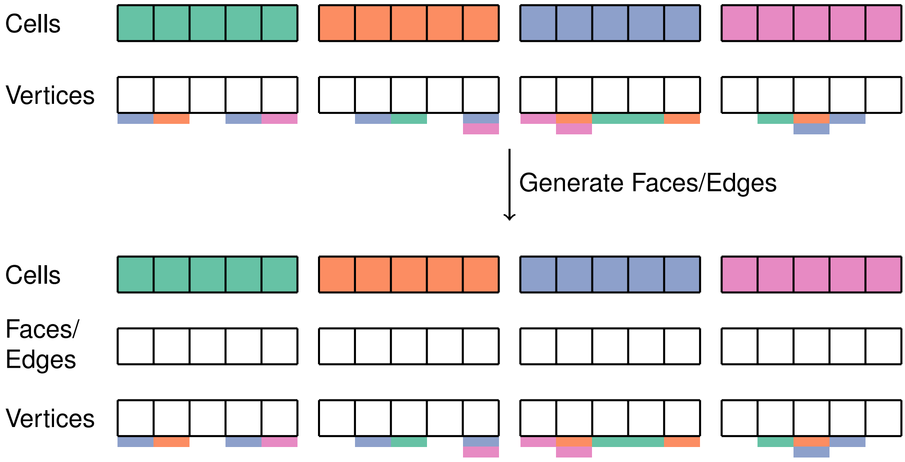
Reconstructing Adjacencies (3)



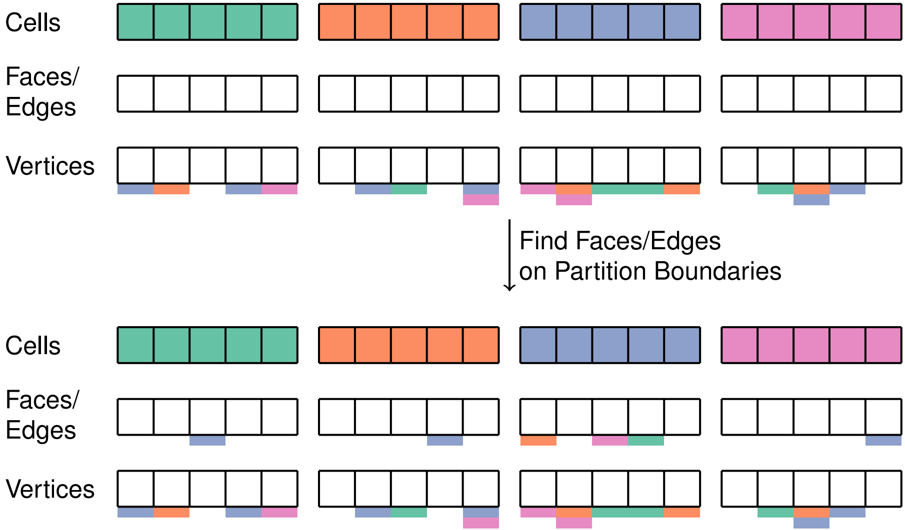
Reconstructing Adjacencies (4)



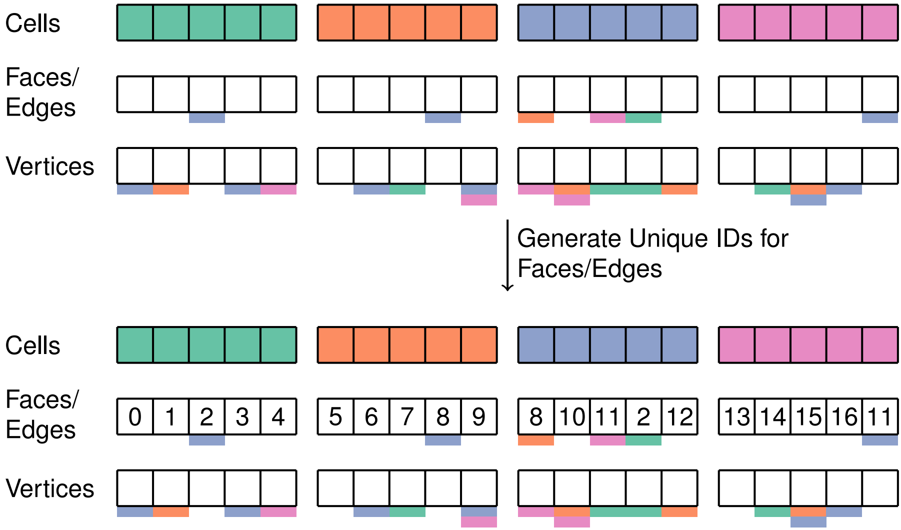
Reconstructing Adjacencies (5)



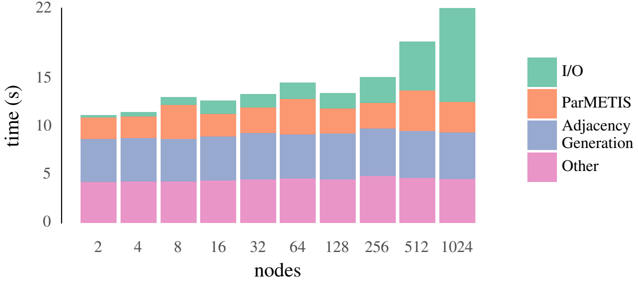
Reconstructing Adjacencies (6)



Reconstructing Adjacencies (7)

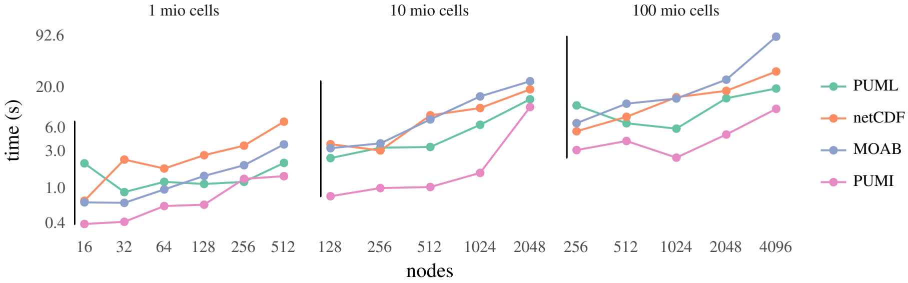


Weak Scaling



- Measured with SeisSol
- SuperMUC Phase 1
- 640,000 cells per node

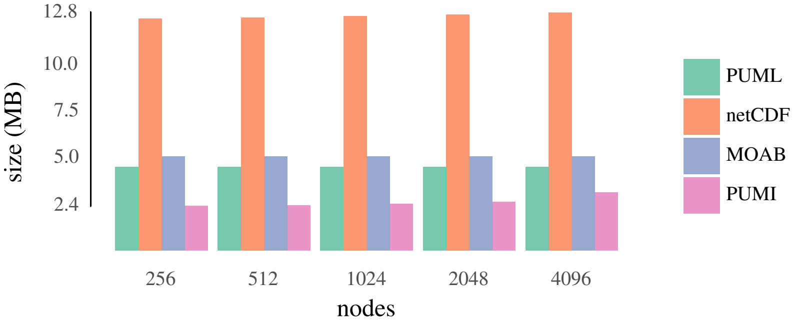
Strong Scaling



- Scenario based on the 1992 Landers earthquake
- MOAB, PUMI measured with proxy applications
- netCDF: SeisSol's *old* mesh format (based on heavy pre-computing)

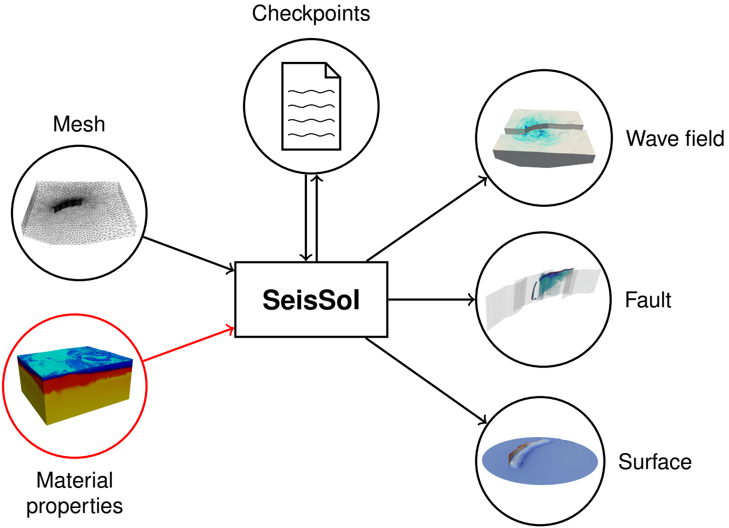
Mesh Size

100 mio cells



Only PUML supports runtime partitioning!

Material Properties

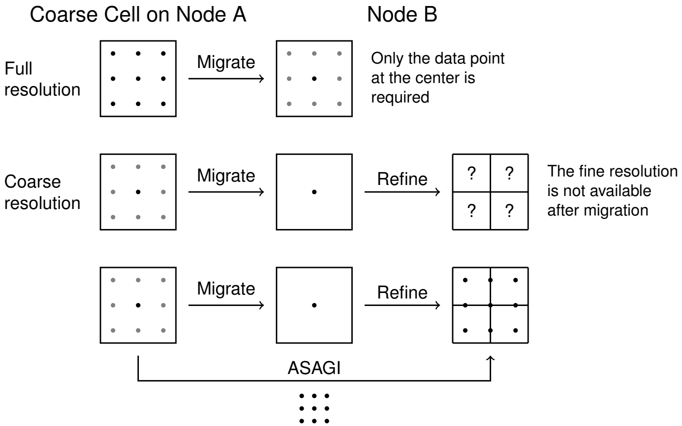


ASAGI: A Parallel Server for Adaptive Geoinformation

- Designed for **dynamic adaptive** simulations
- Reads geoinformation (material properties, bathymetry data, ...) stored in **Cartesian** grids
- **Automatic replication** between nodes using **MPI windows** or an explicit **communication thread**
- Support for **hybrid parallelization** (MPI+X)
- Simple interface
- Open source: `https://github.com/TUM-I5/ASAGI`

Load Balancing in Dynamic Adaptive Simulations

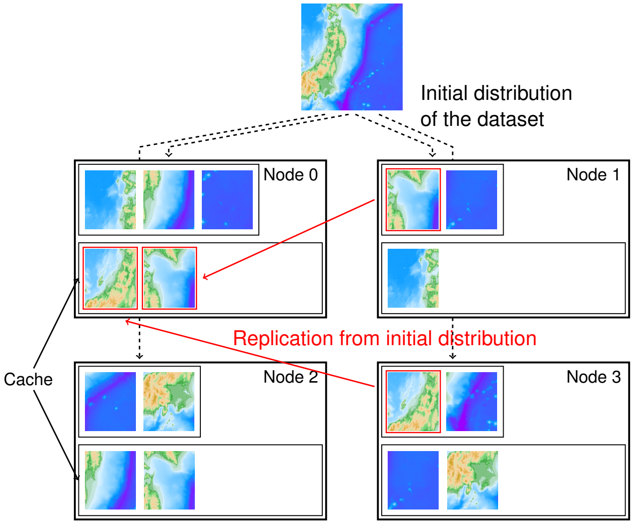
What happens with the geoinformation
if a coarse cell is **migrated between nodes**?



User Interface

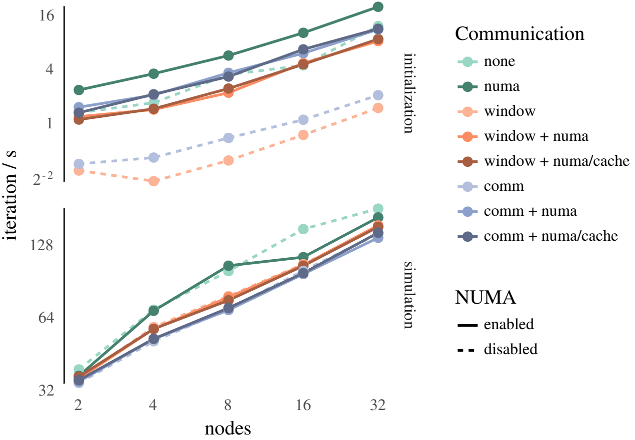
```
// Initialization ...

asagi::Grid* geoData = asagi::Grid::create();
geoData->setComm(MPI_COMM_WORLD);
geoData->setThreads(omp_get_max_threads());
// Set other parameters (optional):

#pragma omp parallel
{ geoData->open("geo_data.nc"); }

// Do the simulation
while (t < end_time) {

  #pragma omp parallel for
  for (int i = 0; i < cells.size(); i++) {
    float par = geoData->getFloat(cells[i].coord);
    // Compute 'cells[i]' with parameter 'par'
  }

}

delete geoData;
// Free other resources
```

ASAGI: A Distributed Server



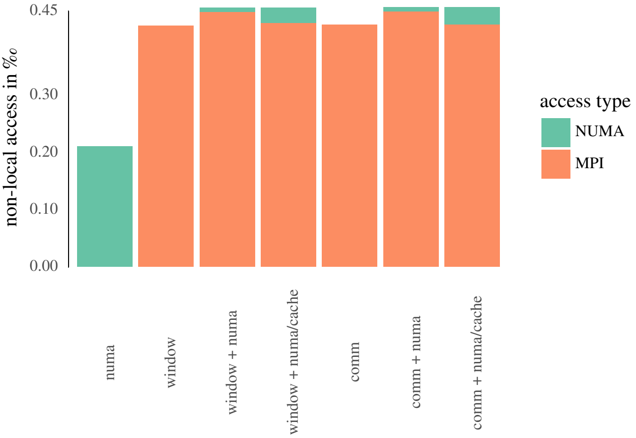
- Dataset is divided into chunks and distributed to all nodes
- First access copies the chunk to the local cache

Sam(oa)^2



- 2011 Tohoku tsunami
- Adaptive mesh refinement
- SuperMUC Phase 1
- Bathymetry: $14,000 \times 8,000$
- 3D displacement (80 time steps): $2,572 \times 3,621$

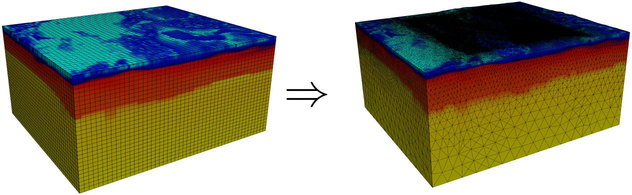
Sam(oa)^2: Locality



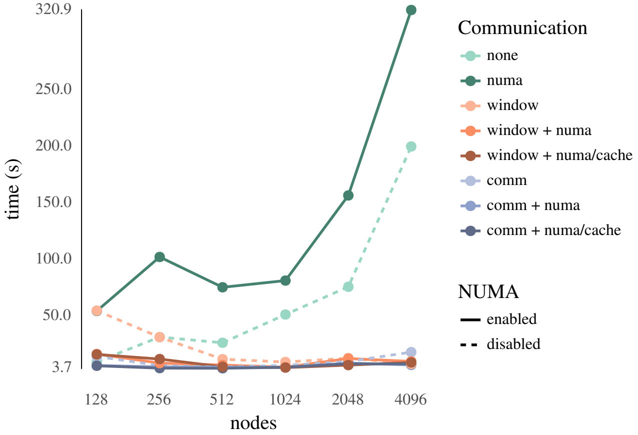
- 2011 Tohoku tsunami
- Adaptive mesh refinement
- 32 nodes
- Bathymetry: $14,000 \times 8,000$
- First 100 time steps

ASAGI in SeisSol

- Only used at initialization
- **Online mapping** of material properties (e.g. density) to cells
- Same input dataset is **reused for different meshes**

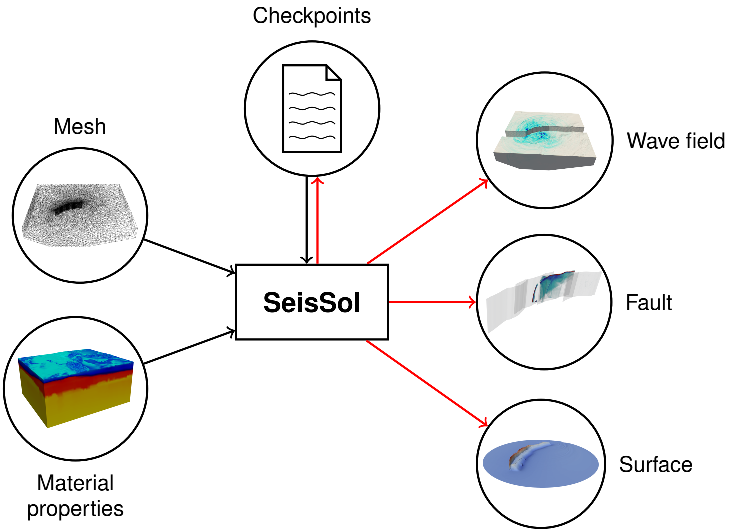


1994 Northridge Earthquake



- SuperMUC Phase 1
- 75 mio cells in SeisSol
- 527 mio data points in the material file
- Trilinear interpolation
- I/O time included

Asynchronous I/O

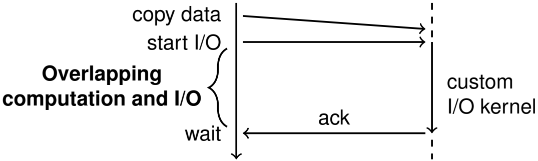


HDF5, netCDF, PnetCDF, SIONlib

- HDF5
  → no asynchronous I/O
- netCDF
  → no asynchronous I/O
- PnetCDF
  → non-blocking API but no asynchronous I/O
- SIONlib
  → no asynchronous I/O
- ADIOS
  → asynchronous I/O through advanced back-ends

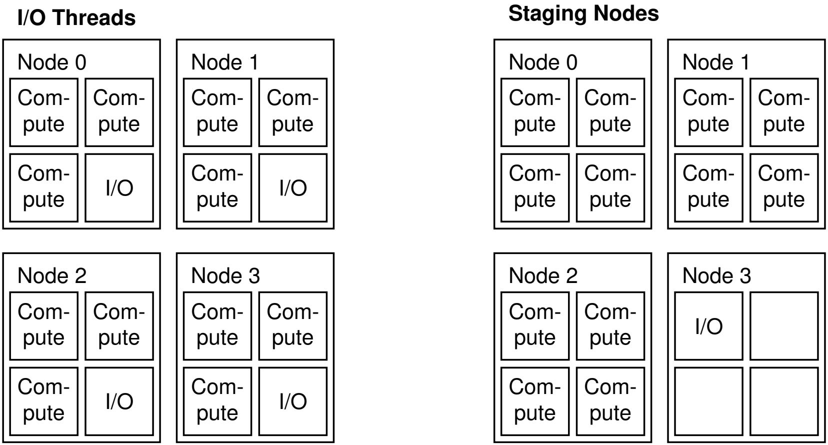
ASYNC: An Asynchronous I/O Library

- Small library for writing data asynchronously
- Handles **data movement** and **synchronization**
- **Custom I/O kernels**



- Can be combined with HDF5, netCDF, . . .
- Useful for **I/O threads** and **staging nodes**
- Open source: `https://github.com/TUM-I5/ASYNC`

I/O Threads vs Staging Nodes



I/O Threads vs Staging Nodes

Advantages and Disadvantages

I/O Threads

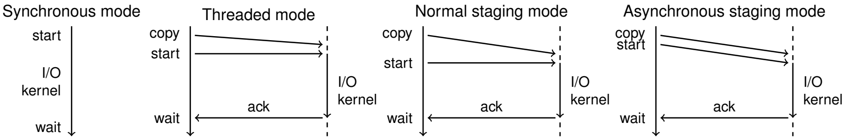
+ **Simple** data movement
+ Useful for CPUs with **many cores per node**
- Requires hybrid parallelization

Staging Nodes

+ **Controllable overhead**
+ Useful for **heterogeneous systems**
+ I/O nodes can be used for post-processing
- **Complex implementation** / explicit staging server
- Requires large amount of memory on staging nodes

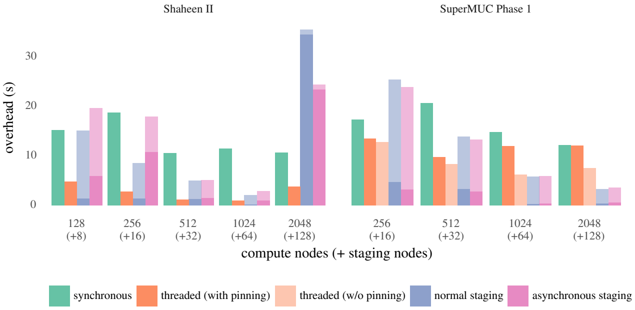
ASNYC: I/O Threads vs Staging Nodes

- ASYNC supports I/O threads and staging nodes
  → Switch between modes by changing a configuration variable



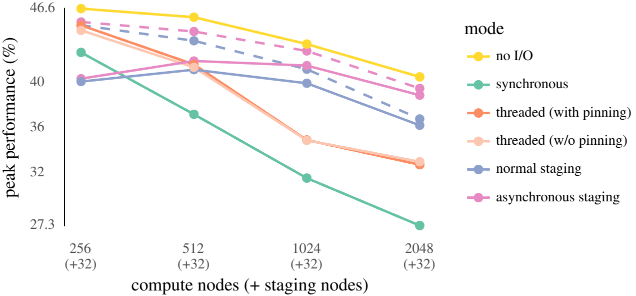
- No staging server required
  → `MPI_COMM_WORLD` has to be replaced

Wave field output



- 1992 Landers earthquake
- 13 GB of data per snapshot (191 million cells)
- Average over 16 snapshots
- XDMF/HDF5 format

Multiple I/O Kernels

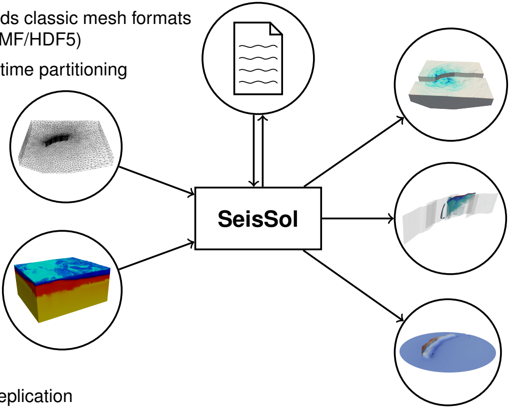


- 1992 Landers earthquake
- Wave field every 100th time step
- Checkpoint every 200th time step (717 GB of data per checkpoint)

Conclusion

PUML:

- Reads classic mesh formats (XDMF/HDF5)
- Runtime partitioning



ASYNC:

- I/O threads or staging nodes
- Custom I/O kernels
- Combined with netCDF, HDF5, . . .

ASAGI:

- Automatic replication
- Chunk cache
- Support for dynamic adaptive applications