OPM Flow on GPU: Matrix Assembly and Linear Solvers

Tobias Meyer Andersen
Jakob Torben, Kjetil Olsen Lye, Atgeirr Flø Rasmussen
SINTEF Digital

OPM Summit 2026 Utrecht

Main topics

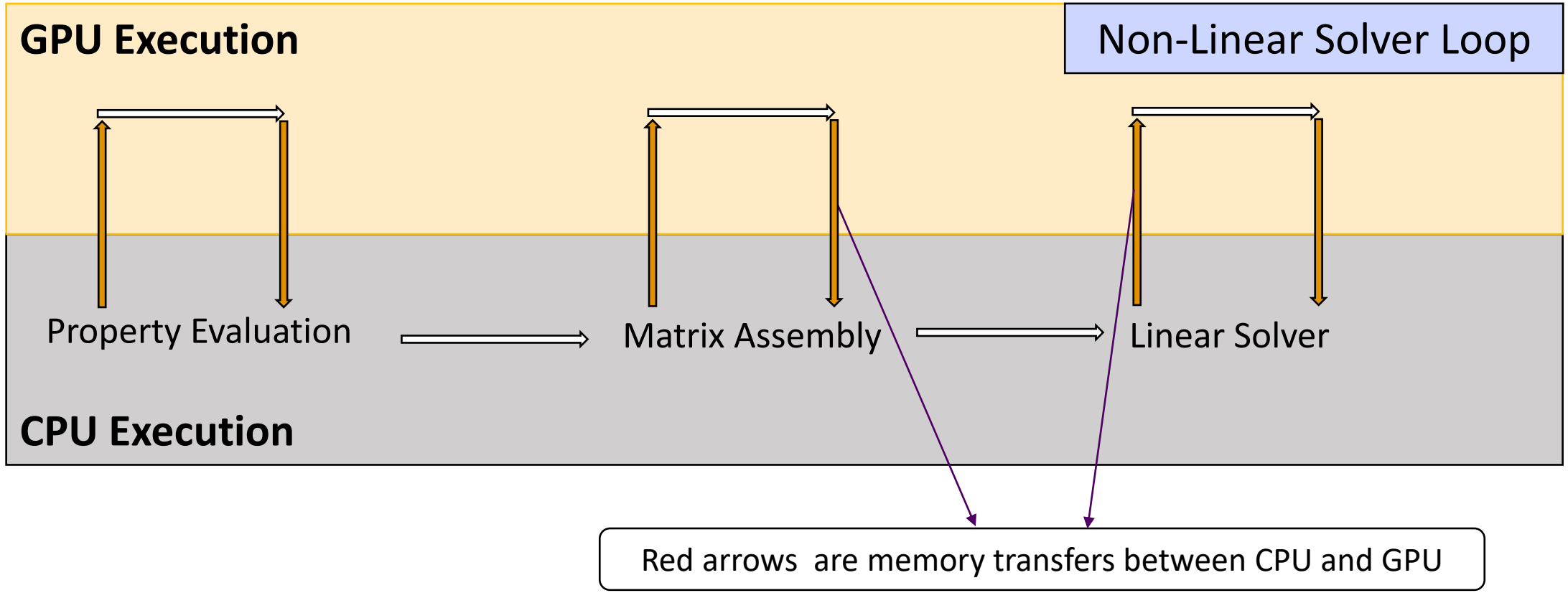
- Progress of GPU porting OPM simulation loop on GPU
  - Linear solvers advances
  - First GPU assembly
- Future work

GpuISTL Progress Since Last OPM Summit:

- CPR-AMG preconditioners:
  - Optimized and merged
  - Both AMGX (Nvidia) and HYPRE (AMD+Nvidia) support
  - MPI support in development
  - Tested on several real-asset cases
  - Still ongoing parameter-studies
  - ECMOR conference paper coming about this
    - *An Open GPU CPR–AMG Implementation and Configuration Study for Field-Relevant Cases, J. Torben et.al*

OPM Flow on GPU Status

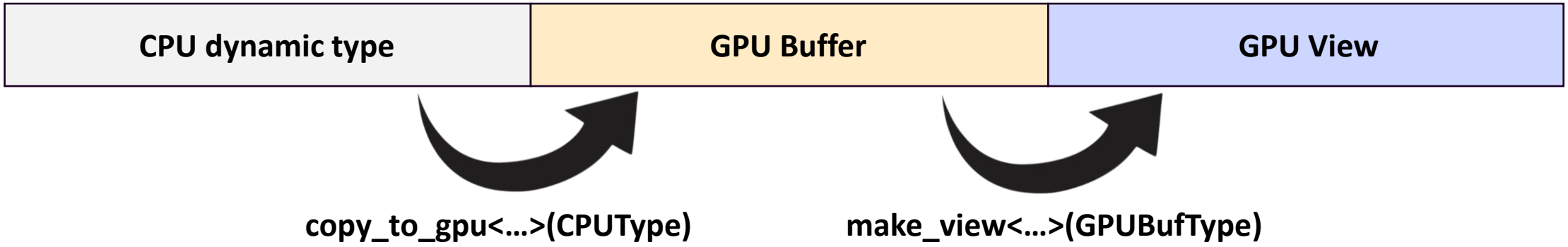
- Linear Solvers are the most mature part, CPR-AMG, MPI, custom smoothers etc…
- Matrix assembly currently tailored to SPE11 & blackoil physics
- Property evaluation is most experimental part, SPE11 variant coming first



Bringing GPU Support to Matrix Assembly

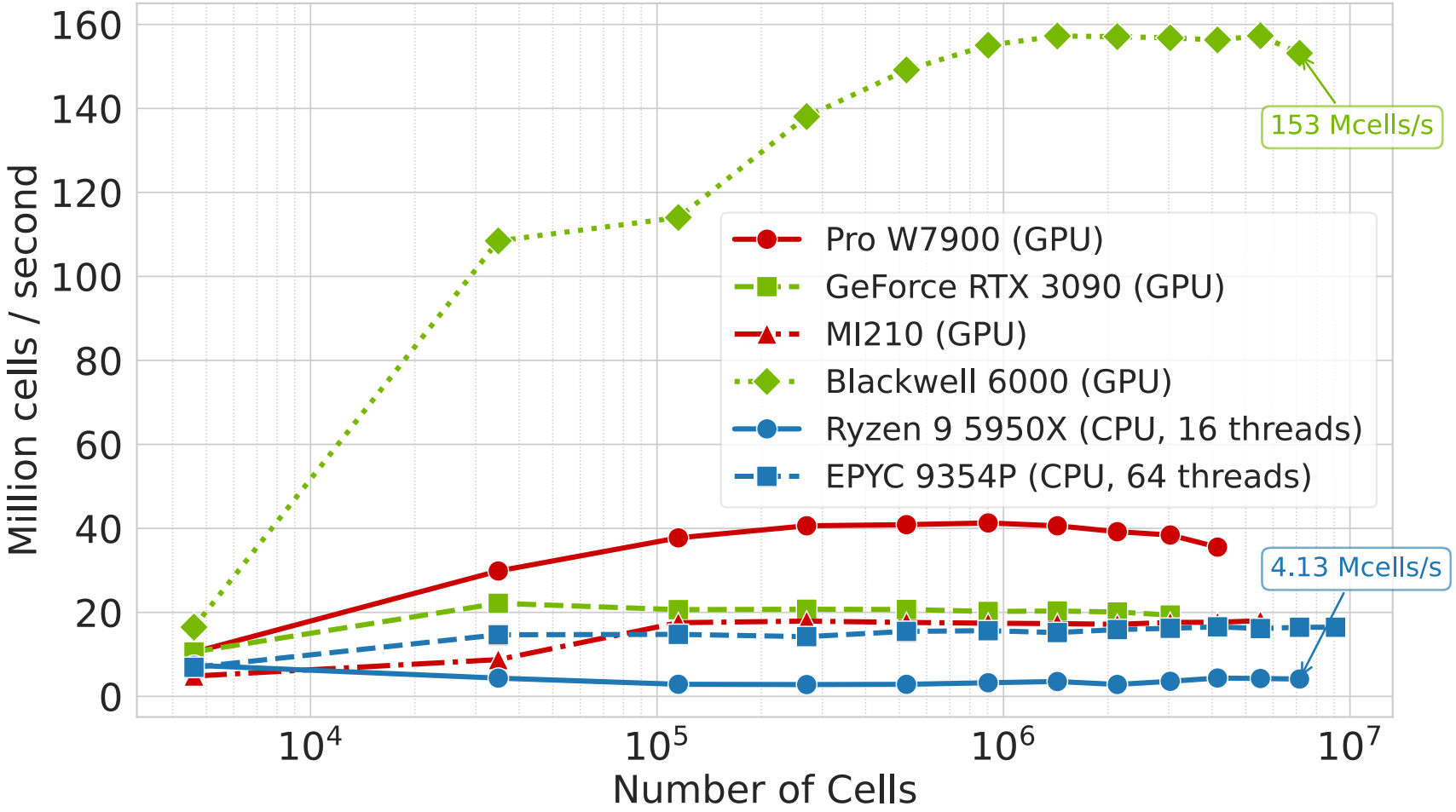
- Each cell can be linearized in parallel, excellent for GPU!
- Make types used in tpfalinearizer GPU friendly
  - Virtually no code duplication!
- Create a parallelization wrapper
  - For loop over cells on CPU, spawn a thread per cell on GPU

GPU assembly supports both AMD and NVIDIA GPUs



GPU Assembly SPE11 Scaling Study

- CSP SPE11 C cases generated with pyopmspe11
- Only time inside computational kernel considered, not preparatory data movement



What is currently limiting the performance?

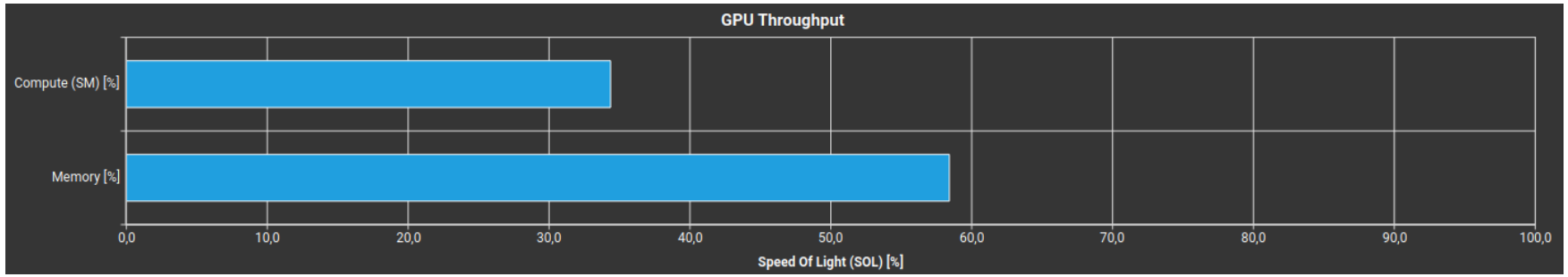
- The way things are computed
  - Computational workload and mapping to GPU hardware
- The place where things are computed
  - Memory layout of the data operated on

The Way Things are Computed

- GPUs run many threads in parallel
- Each thread has a state
  - High register usage limits parallelism because of resource requirement
  - Also causes register spillage
- What precisely is 'complicated'?
  - Extensive use of complex objects like blackoilintensivequantities, FluidState etc…
  - Deep function nesting
  - Many properties being accessed and computed in a single kernel
  - Too many values/registers must be available simultaneously!

The Place where things are Computed

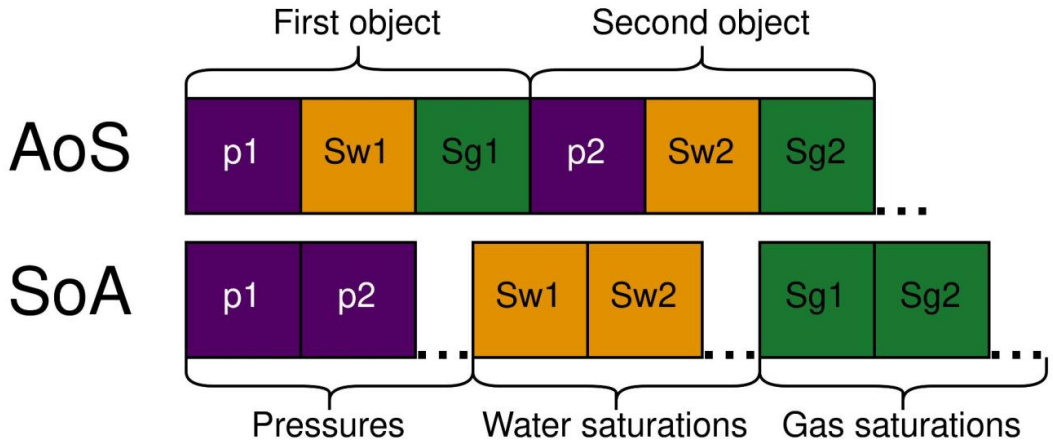
- Profiling reveals we use 60% of memory bandwidth – looks good?



- Profiling reveals inefficient cache usage
  - Highly polluted cache
- Why?

The Place where things are Computed

- Operating on scattered parts of the Jacobian
  - Partially unavoidable
- Array of structures (AoS) instead of structure of arrays (SoA)

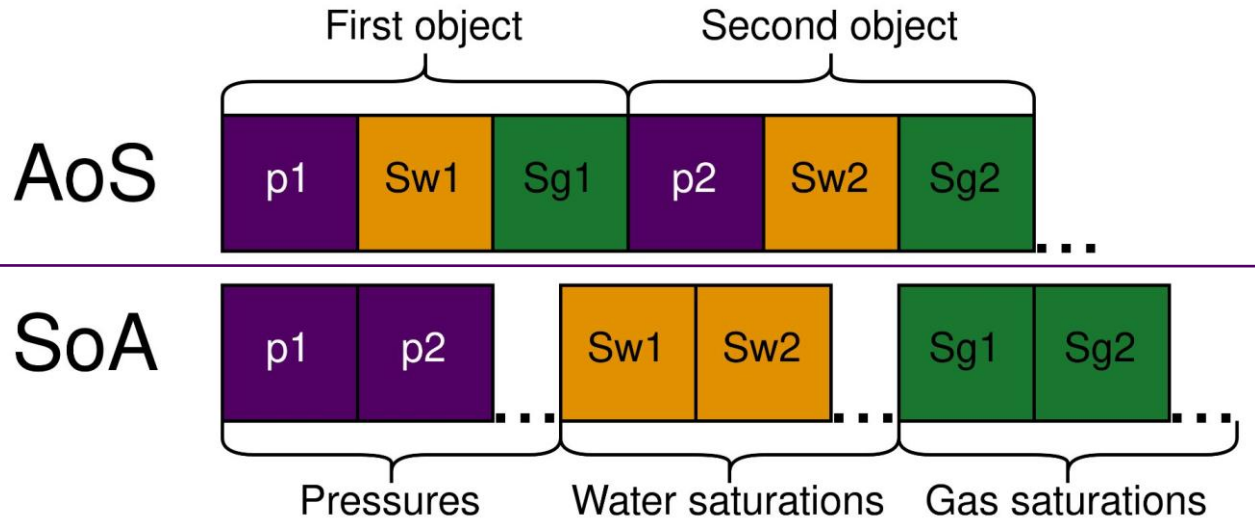


  - GPU cache heavily favors threads operating on contiguous memory

Arrays of Structures vs Structure of Arrays

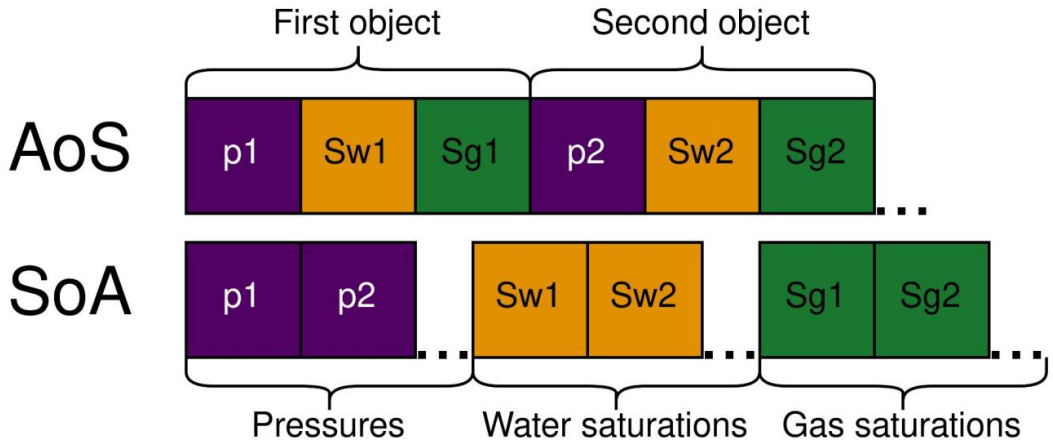
```
struct SimpleFluidState
{
 double pressure
 double Sw;
 double Sg;
};

std::vector<SimpleFluidState> fstates;
```

```
struct SoA
{
 std::vector<double> pressure;
 std::vector<double> Sw;
 std::vector<double> Sg;
};
SoA fstates;
```



The Place where things are Computed

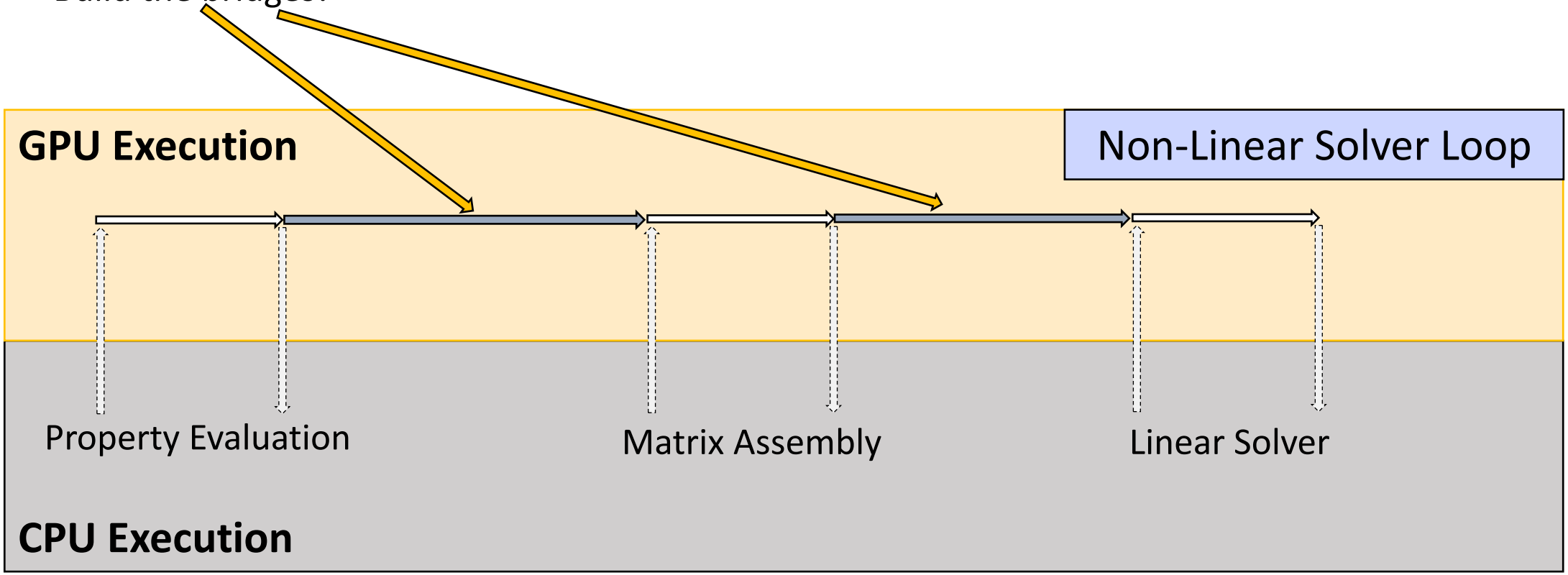
- Operating on scattered parts of the Jacobian
  - Partially unavoidable
- Array of structures (AoS) instead of structure of arrays (SoA)



  - GPU processes multiple cells simultaneously -> SoA is better

Further Work

- Merge the last part of the GPU assembly code
- Improve memory layout and reduce register pressure, use more single precision compute?
- Complete MPI support for CPR-AMG on real asset models
- Extend property PoC to handle more realistic cases
- Build the bridges!



Acknowledgement

We thank Equinor for funding this work
We also thank AMD for technical discussions

Contact: tobiasmeyer.andersen@sintef.no