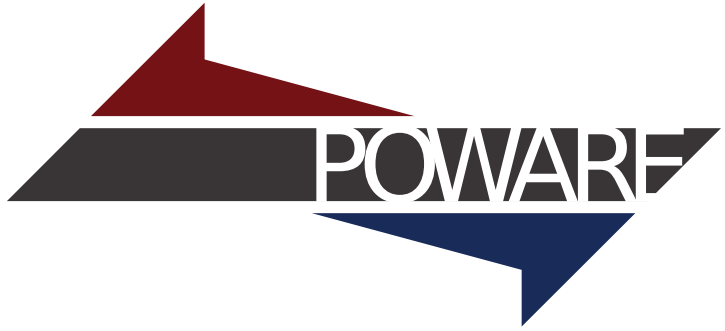
Localized Linearization

Andreas Lauser

March 12, 2015



The Problem

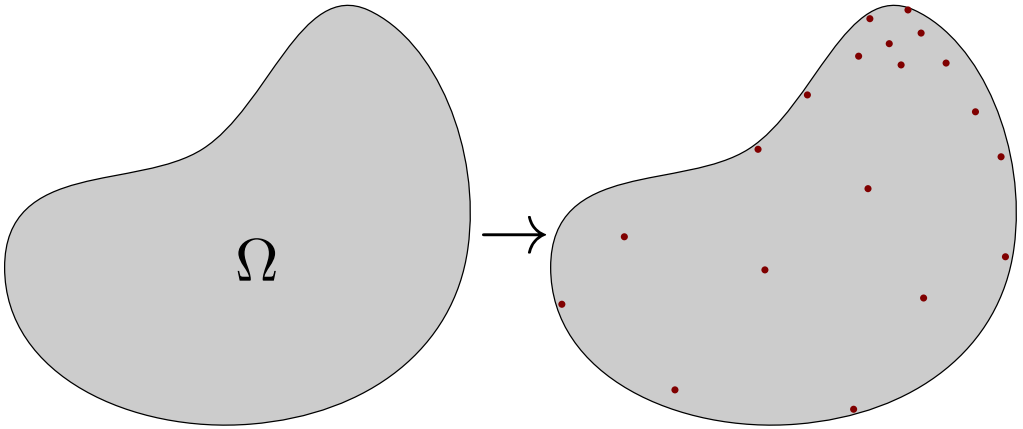
Caches are small,
memory is slow,
let's do it on the GPU!

Or not?!

Overview

Overview

Field Discretization

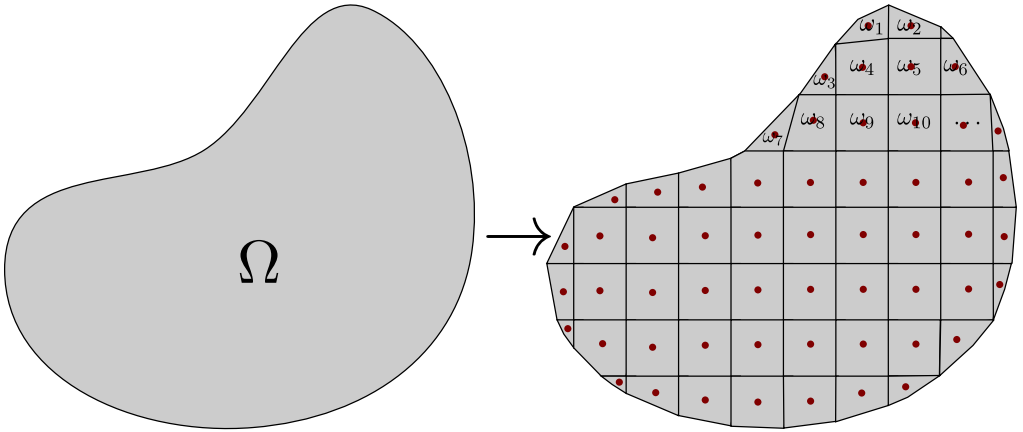


Scalar field: Function $f : \Omega \mapsto \mathcal{R}$ were $\Omega \subseteq \mathcal{R}^n$ for some $n \in \mathcal{N}$

Usual discretization approach:

- Define the value of the field at a finite number of *degrees of freedom* (DOFs)
- Interpolate in-between
  - Approximates field by a vector of scalars

Grids



Partitioning Ω of into smaller sets ω_i:

- Sub-domains ω_i usually exhibit regular shape
  - Simplices, parallelepipeds, etc.
- ω_i usually called *element* or *cell*
- In the following: Cell-centered finite volume method (i.e., one DOF per cell)

Combining Fields

$$h(\mathbf{x}) = (f \star g)(\mathbf{x}) = f(\mathbf{x}) \star g(\mathbf{x}) \text{ where } \mathbf{x} \in \Omega$$

Left hand side:

- Iterate over all DOFs, calculate the values $f(\mathbf{x})$ and $g(\mathbf{x})$ and combine them locally
- (in the following called “element based/localized approach”)

Right hand side:

- Calculate all values of f, then all of g, then combine them
- (in the following called “grid based/global approach”)

Simple Example: $h(\mathbf{x}) = |\mathbf{x}|\sqrt{|\mathbf{x}|}$

Consider the following:

$$f(\mathbf{x}) = |\mathbf{x}|, \qquad g(\mathbf{x}) = \sqrt{|\mathbf{x}|}$$
$$\star \equiv \cdot, \qquad h(\mathbf{x}) = |\mathbf{x}|\sqrt{|\mathbf{x}|}$$

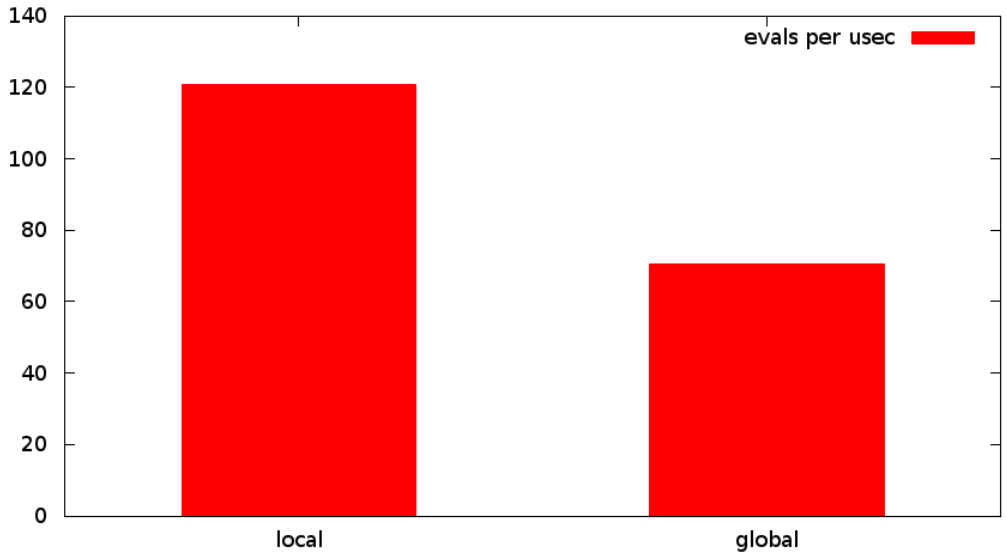
Localized:

```
for i in cells:
  x = abs(pos(i))
  f[i] = x
  g[i] = sqrt(x)
  h[i] = f[i] * g[i]
```

Global:

```
for i in cells:
  f[i] = abs(pos(i))
for i in cells:
  g[i] = sqrt(abs(pos(i)))
for i in cells:
  h[i] = f[i] * g[i]
```

Performance[1]: $h(\mathbf{x}) = |\mathbf{x}| \sqrt{|\mathbf{x}|}$



Machine: Intel Core i7-5930K
Compiler: GCC 4.8.3, '-O3 -march=native -ffast-math'

Ratio Localized/Global: ≈ 1.60

[1] `https://poware.de/adue3aib/global-vs-local.tar.gz`

Simple Example: Notes

- Both approaches also applicable for more complicated functions (e.g., relperms) and operators (e.g., gradients).
- Due to prefetching, performance advantage of localized approach is smaller for more expensive f or g
- Very often, function to be computed is very cheap
  - e.g. $h = 1 - f$
  - (Off topic: Iterative linear solvers particularly affected)

Advantages and Disadvantages

Advantages of the localized approach:

- Better cache locality
- Easier to parallelize
  - Only a single loop

Advantages of the global approach:

- More modular representation of the underlying equations
  - Simplifies/allows implementation of special purpose PDE languages like *DOLFIN*[2] or *Equelle*[3]

[2] `http://fenicsproject.org/`

[3] `http://equelle.org/`

Overview

Preceeding Remarks

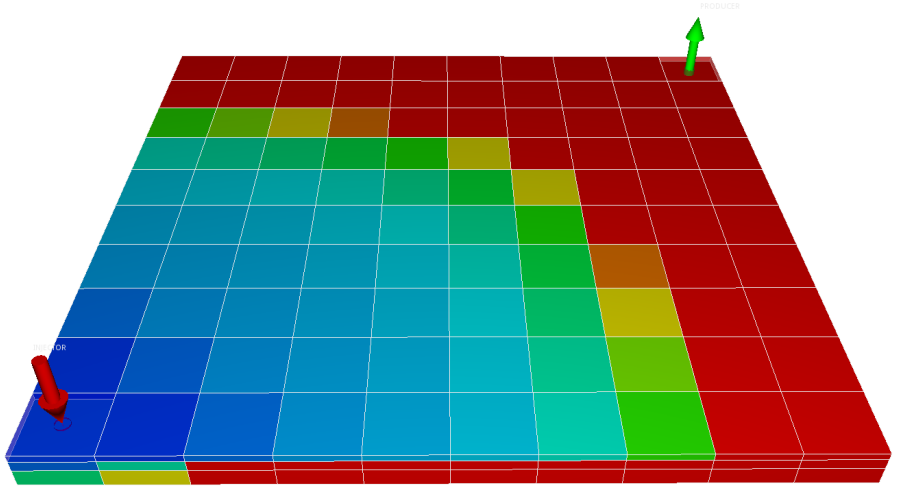
OPM is one of the few projects which where an “apples-to-apples” comparison can be attempted:

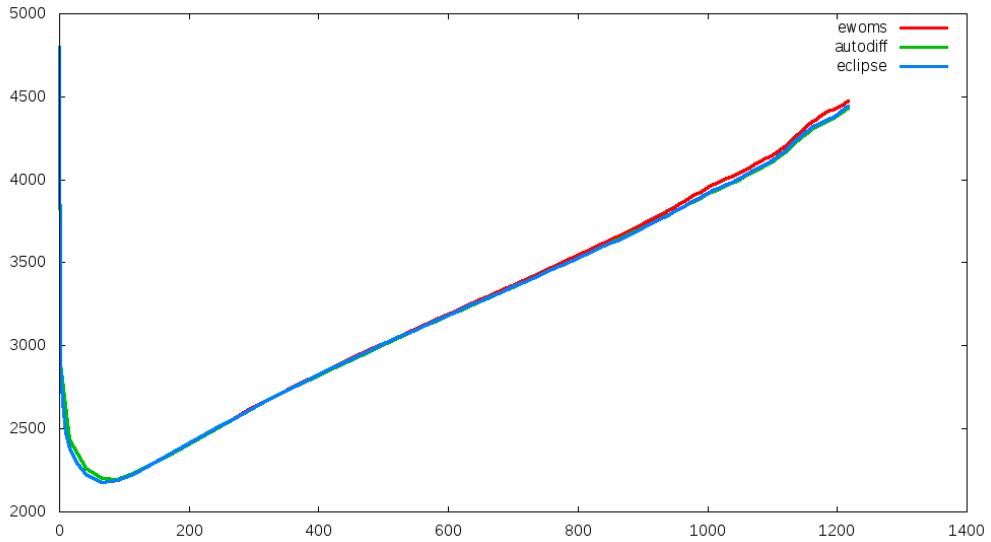
- opm-autodiff uses the global approach to linearization
- eWoms/ebos uses the localized approach
- Grid and deck input parameters identical/very similar

Other differences in implementation are pretty large:

- All tricks like partial relinearization and linearization rescaling are disabled
- Results should be taken with a grain of salt!

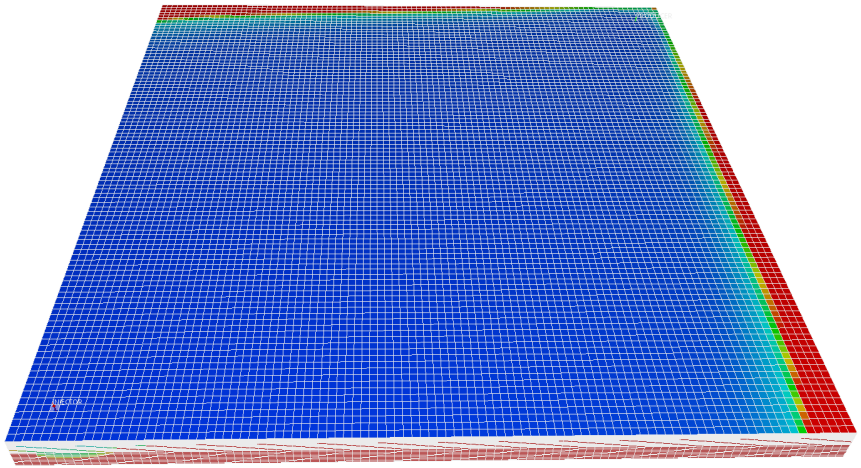
Boring Problem: SPE-1

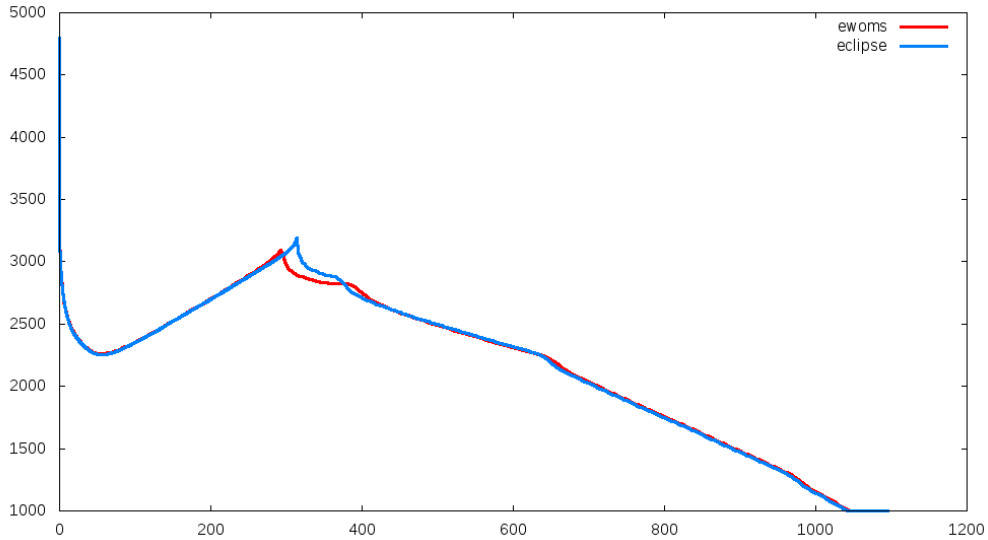




BHP of the “PRODUCER” well vs. time

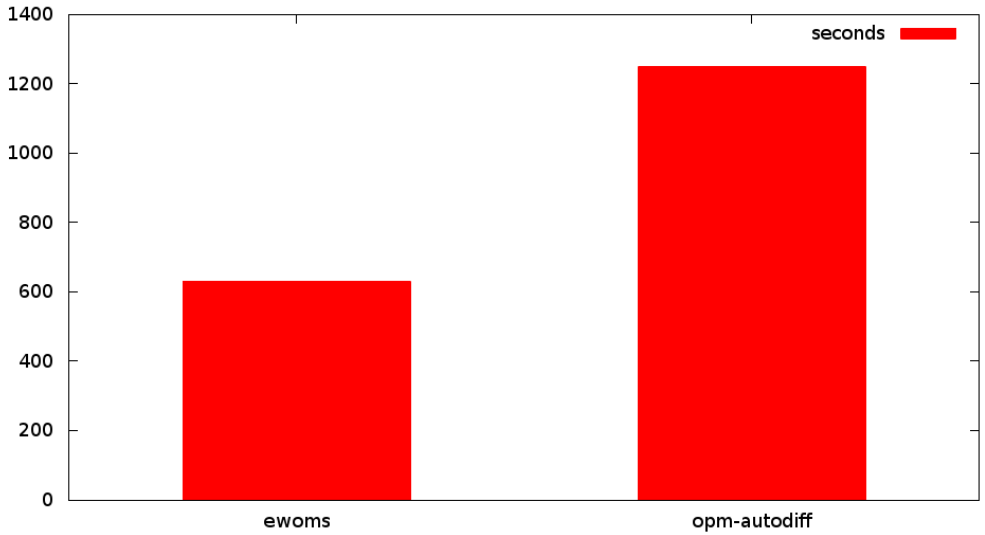
Large Boring Problem: Refined SPE1





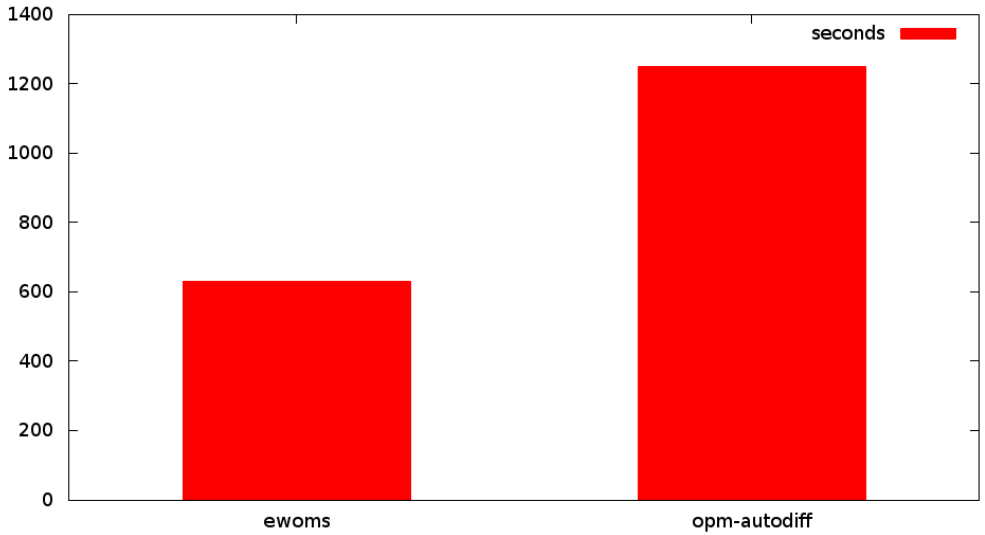
BHP of the “PRODUCER” well vs. time

Performance: Apples and Bananas



Overall time of the refined SPE1 problem. Ratio: 1.98

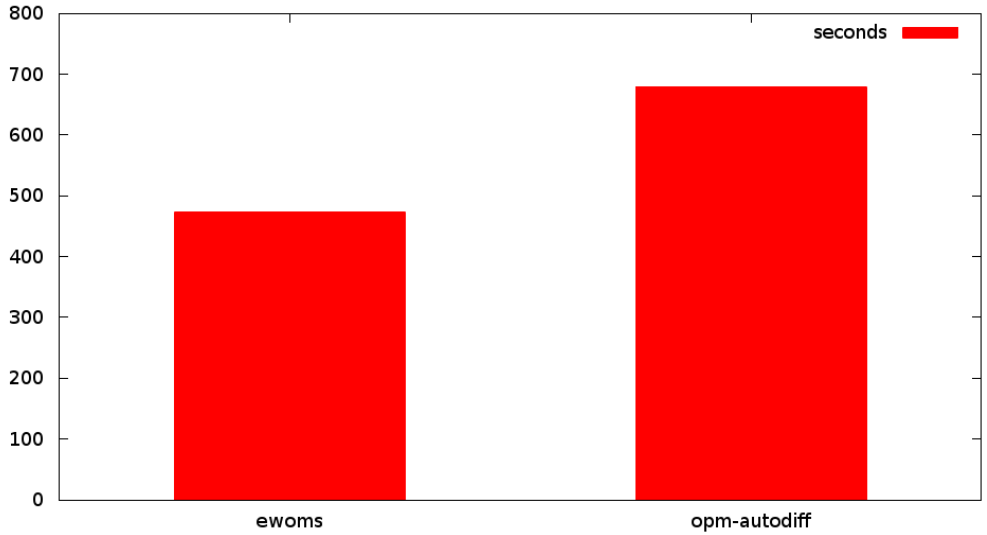
Performance: Apples and Bananas



Overall time of the refined SPE1 problem. Ratio: 1.98

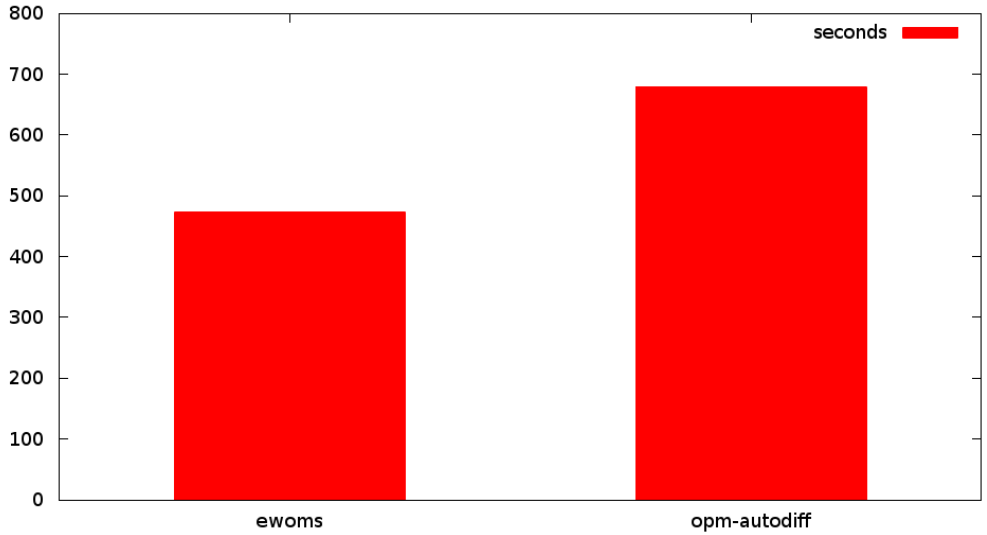
- Linear solver also suffers from “memory bandwith bottleneck”

Performance: Apples and Oranges



Aggregate linearization time of the refined SPE1 problem. Ratio: 1.43

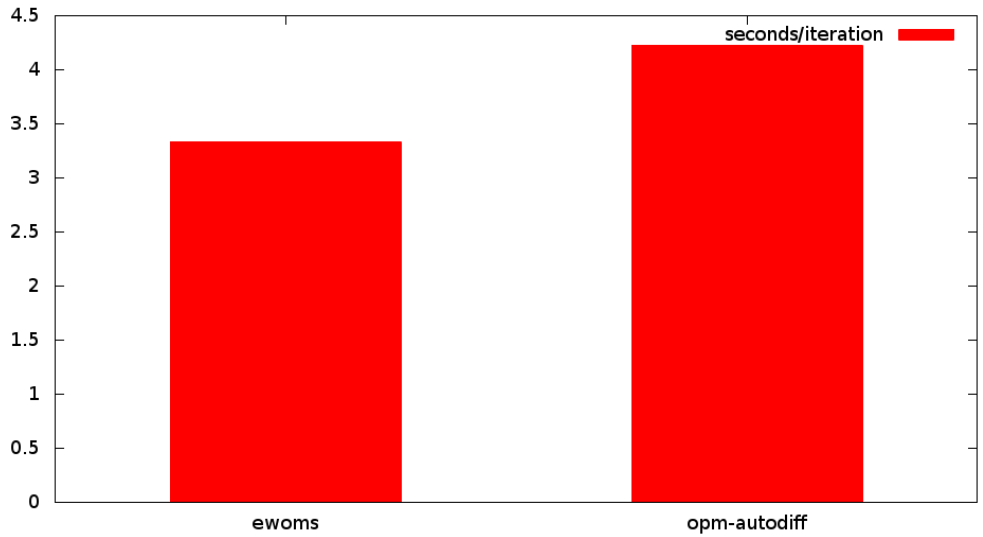
Performance: Apples and Oranges



Aggregate linearization time of the refined SPE1 problem. Ratio: 1.43

- opm-autdiff requires a few iterations more

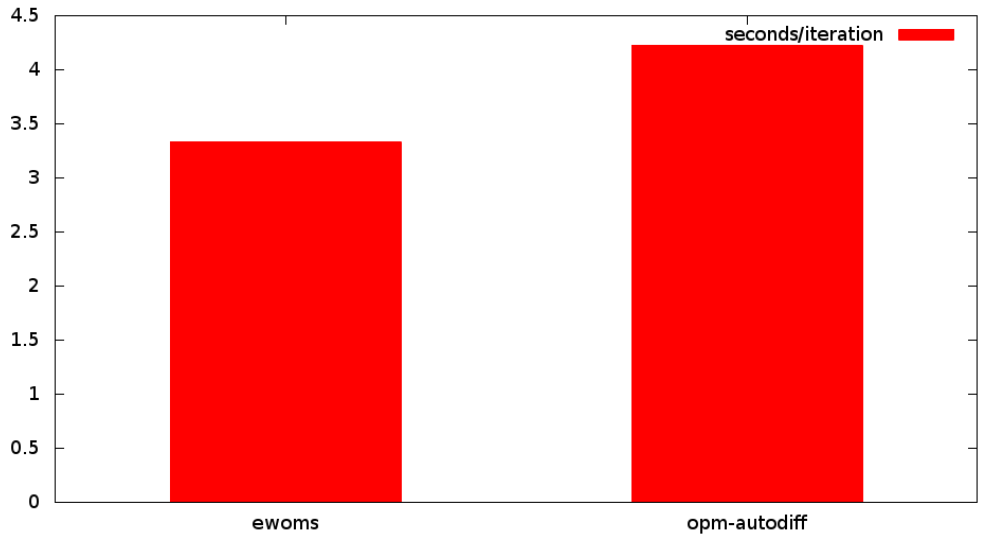
Performance: Green Apples and Red Apples



Iteration adjusted linearization time for the refined SPE1 problem.

Ratio: 1.26

Performance: Green Apples and Red Apples



Iteration adjusted linearization time for the refined SPE1 problem.

Ratio: 1.26

- opm-autodiff uses automatic differentiation, eWoms finite differences

Summary

- Performance comparisons are hard
- Quality of results of all simulators comparable for the unrefined and refined SPE-1 problems.
- Localized linearization approach saw better performance
  - Likely faster in principle because it reduces the “RAM bottleneck” problem

Summary

- Performance comparisons are hard
- Quality of results of all simulators comparable for the unrefined and refined SPE-1 problems.
- Localized linearization approach saw better performance
  - Likely faster in principle because it reduces the “RAM bottleneck” problem

Thank you for your attention.