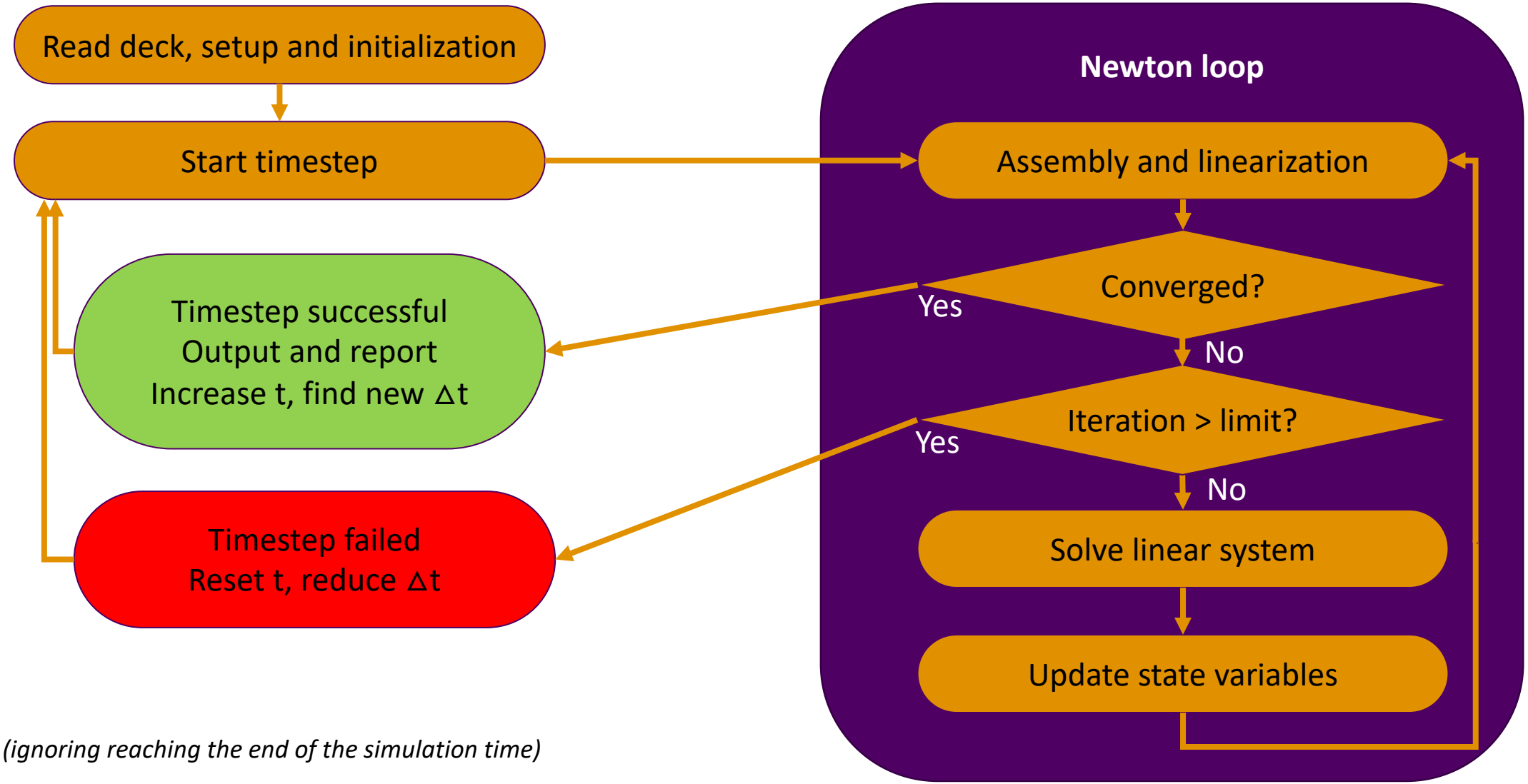
Performance acceleration for CO2 simulations

Atgeirr Flø Rasmussen, Kai Bao
Halvor Møll Nilsen, Bård Skaflestad

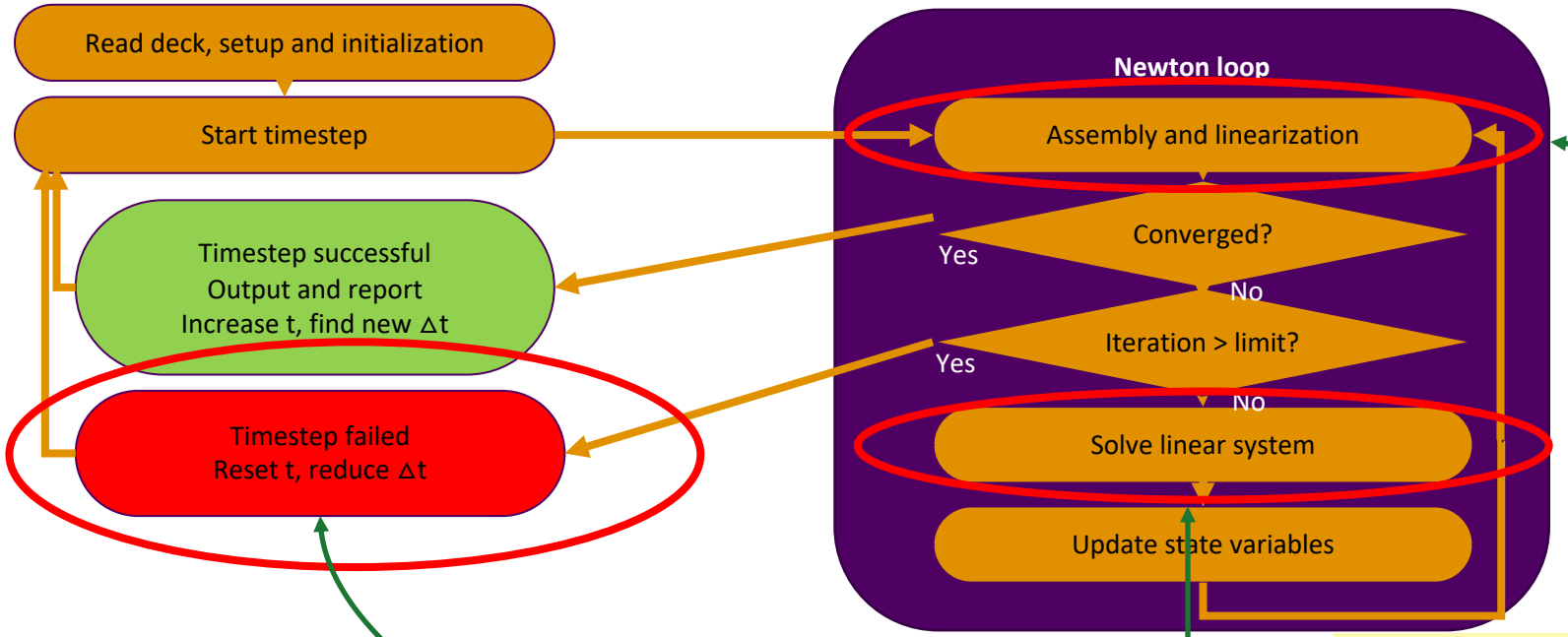
Introduction and example

Understanding OPM performance



(ignoring reaching the end of the simulation time)

What causes performance loss?



What takes the most time?

- Assembly of residual and Jacobian
- Solving linear system

```
================    End of simulation     ===============

Number of MPI processes:          8
Threads per MPI process:          1
Number of timesteps:            271
Total time (seconds):          7611.14
Solver time (seconds):         7610.13
 Assembly time (seconds):       610.28   Failed: 259.6; 42.5%)
   Well assembly (seconds):      23.56   Failed: 10.3; 43.8%)
 Linear solve time (seconds):  6409.74   Failed: 3766.3; 58.8%)
   Linear setup (seconds):      280.46   Failed: 133.7; 47.7%)
 Update time (seconds):         472.65   Failed: 209.4; 44.3%)
 Pre/post step (seconds):       106.13   Failed: 10.0; 9.4%)
 Output write time (seconds):     1.93
Overall Linearizations:         1579     Failed: 671; 42.5%)
Overall Newton Iterations:      1410     Failed: 671; 47.6%)
Overall Linear Iterations:     43554     Failed: 25942; 59.6%)
```

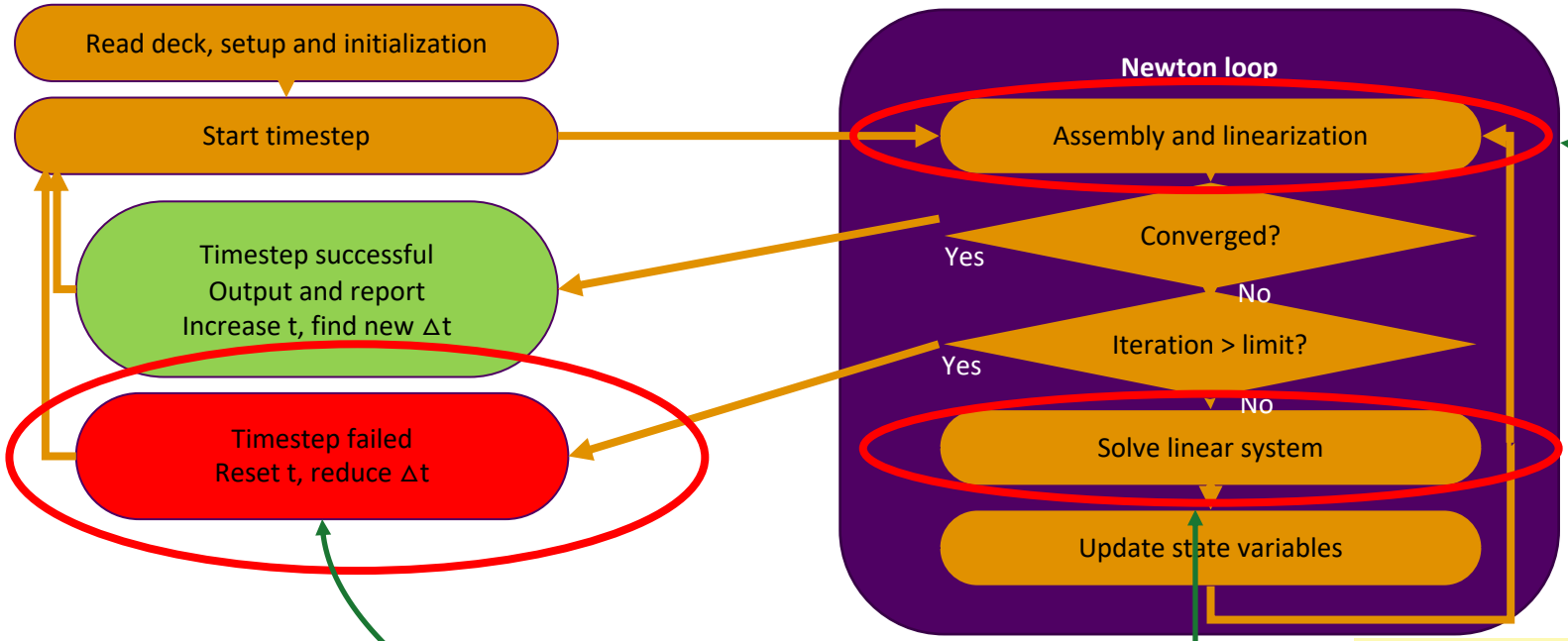
What causes wasted time (”Failed”)?

- Failed Newton solves
- Also failed linear solves (not in flow chart)

Leads to timestep cut!

→ Effort already done on timestep is wasted

What can help? Obscure options?



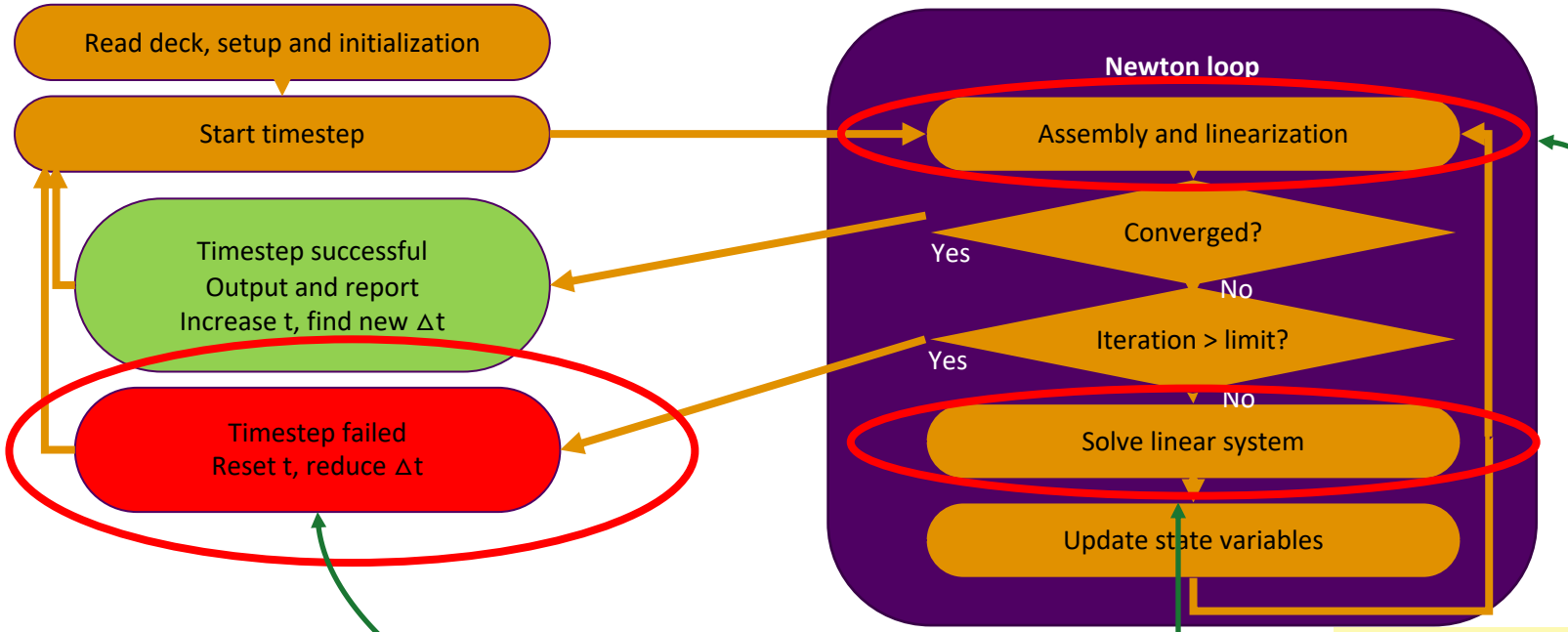

```
================    End of simulation     ===============

Number of MPI processes:          8
Threads per MPI process:          1
Number of timesteps:             78
Total time (seconds):          5636.26
Solver time (seconds):         5634.33
 Assembly time (seconds):       302.18  Failed: 42.4; 14.0%)
   Well assembly (seconds):      11.69  Failed: 1.9; 16.0%)
 Linear solve time (seconds):  5047.34  Failed: 525.0; 10.4%)
   Linear setup (seconds):      140.05  Failed: 21.3; 15.2%)
 Update time (seconds):         259.25  Failed: 49.8; 19.2%)
 Pre/post step (seconds):        22.55  Failed: 0.2; 1.0%)
 Output write time (seconds):     2.57
Overall Linearizations:           768  Failed: 105; 13.7%)
Overall Newton Iterations:        695  Failed: 105; 15.1%)
Overall Linear Iterations:      35058  Failed: 3557; 10.1%)
```

In this particular case, the option

`--linear-solver-ignore-convergence-failure=true`

What can help? Change linear solver?




```
================    End of simulation     ===============

Number of MPI processes:          8
Threads per MPI process:          1
Number of timesteps:             21
Total time (seconds):        308.78
Solver time (seconds):       307.47
 Assembly time (seconds):     64.97 (Failed: 0.0; 0.0%)
   Well assembly (seconds):    3.11 (Failed: 0.0; 0.0%)
 Linear solve time (seconds): 179.12 (Failed: 0.0; 0.0%)
   Linear setup (seconds):    109.47 (Failed: 0.0; 0.0%)
 Update time (seconds):        54.49 (Failed: 0.0; 0.0%)
 Pre/post step (seconds):       6.55 (Failed: 0.0; 0.0%)
 Output write time (seconds):   2.07
Overall Linearizations:         159     (Failed:    0; 0.0%)
Overall Newton Iterations:      138     (Failed:    0; 0.0%)
Overall Linear Iterations:       74     (Failed:    0; 0.0%)
```

Using the CPR linear solver

`--linear-solver=cpr`

Concrete tips

Tip 1: Run OPM Flow in parallel!

Parallel runs are NOT achieved up by modifying the deck! Instead, use mpirun:

```
> flow MYCASE.DATA
> mpirun -np 8 flow MYCASE.DATA
```

Serial run
Parallel run with 8 processes

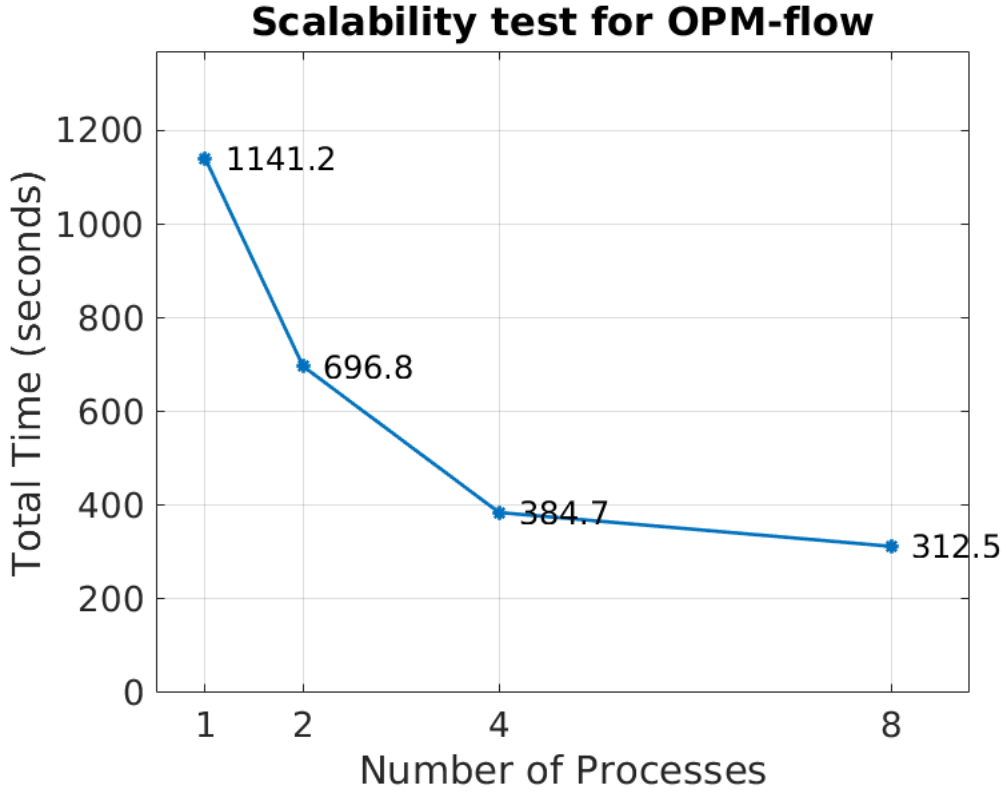
Are there pitfalls?

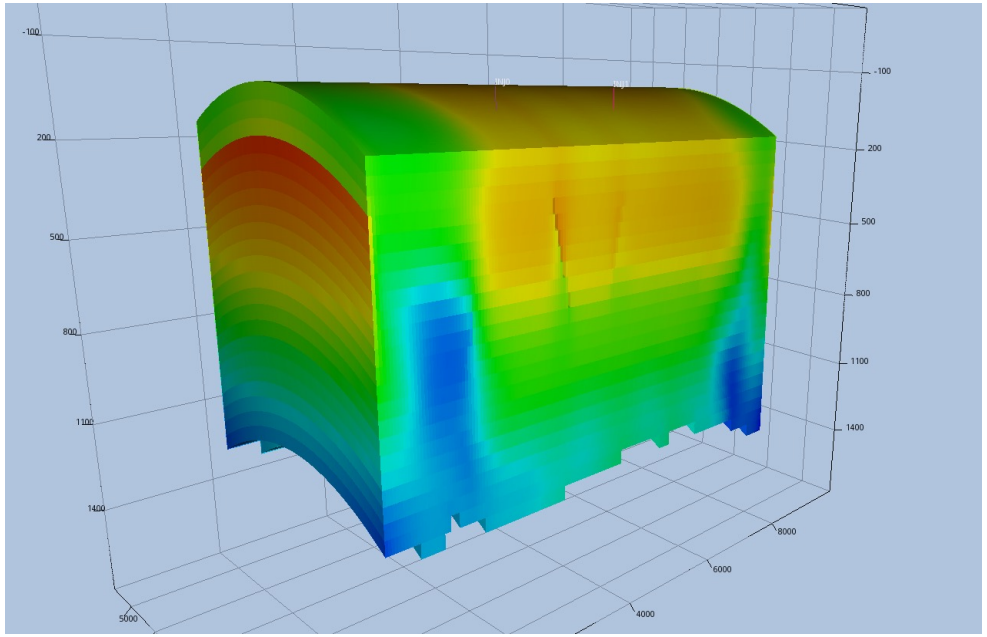
- Can get different number of iterations
  - … which leads to different timestepping
  - … which might lead to different behaviors (esp. for prediction)
- But: sensitivity to timestep sizes is not unique to parallel vs. serial
- Most developers run mostly in parallel

Parallel scalability: desktop

SPE11 Case C

376700 active cells, 1000 years simulation time, two threads per process, Intel i9-7940X CPU @ 3.10GHz

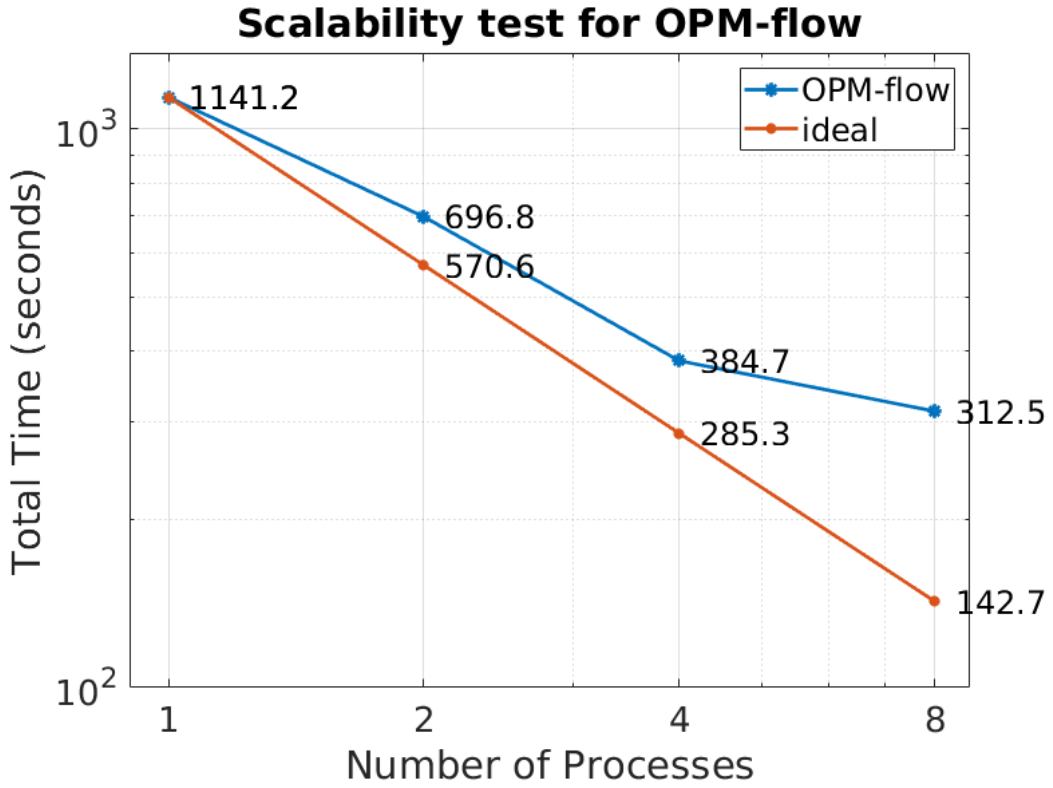


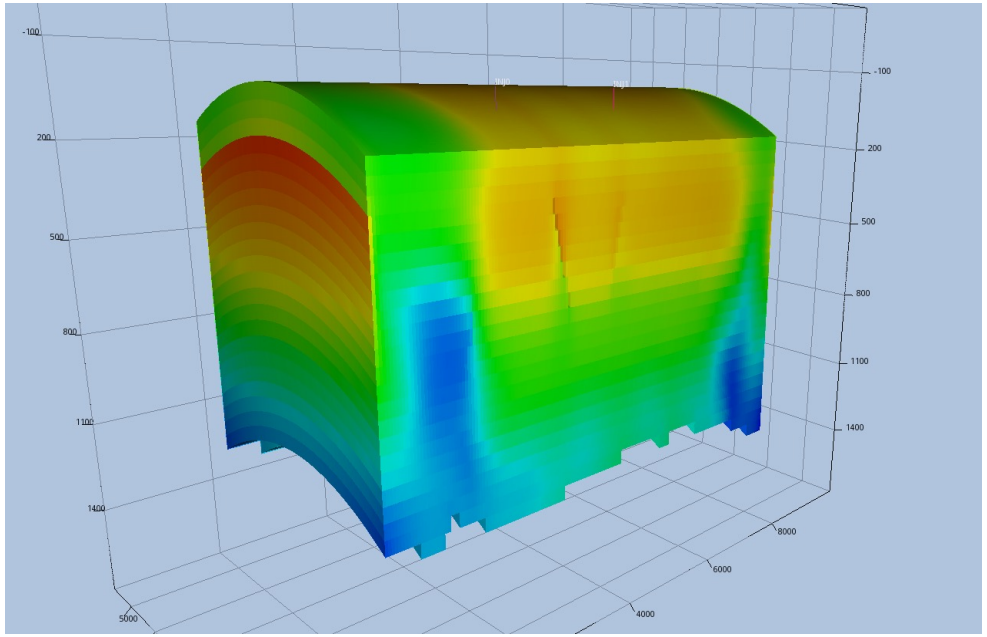


Parallel scalability: desktop

SPE11 Case C

376700 active cells, 1000 years simulation time, two threads per process, Intel i9-7940X CPU @ 3.10GHz

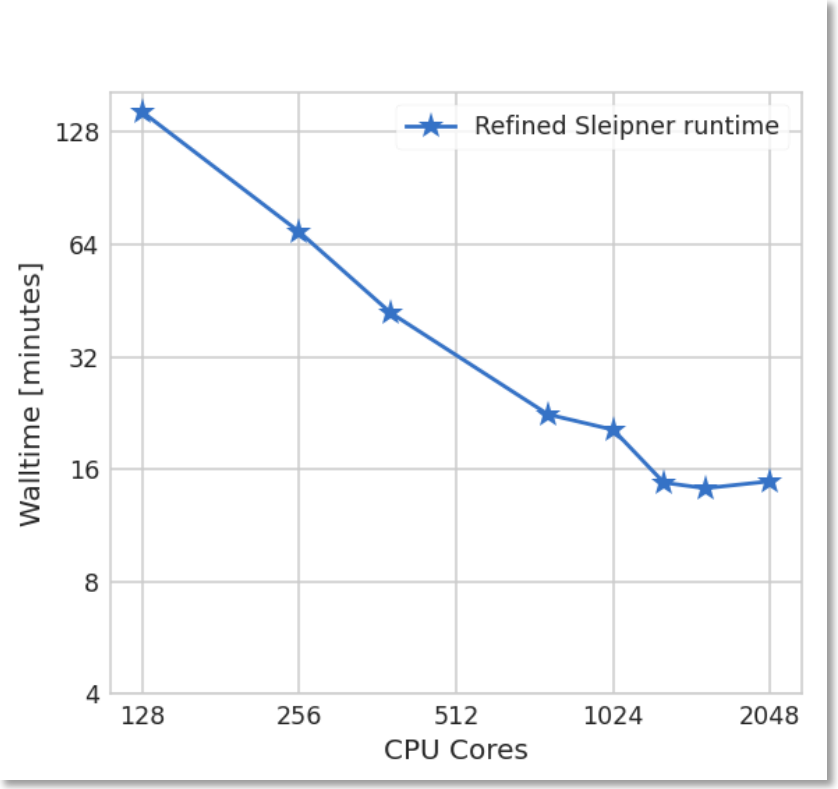




Parallel scalability: HPC

Refined Sleipner-derived case

18M active cells, 20 years simulation time, two threads per process, Karolina cluster (CZ), 128 cores/node

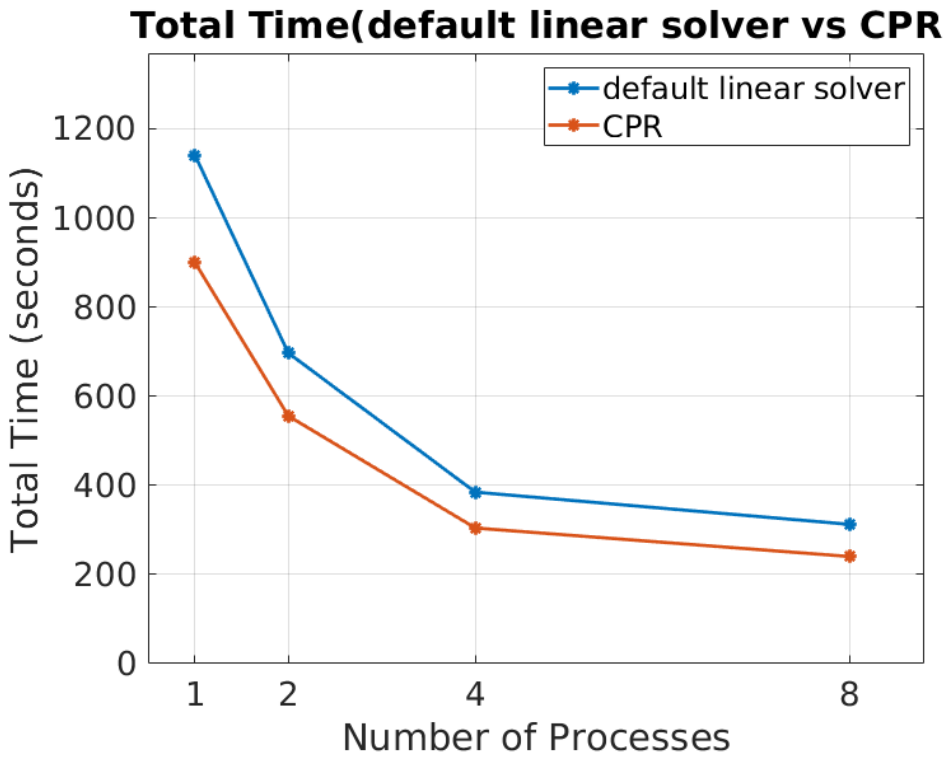


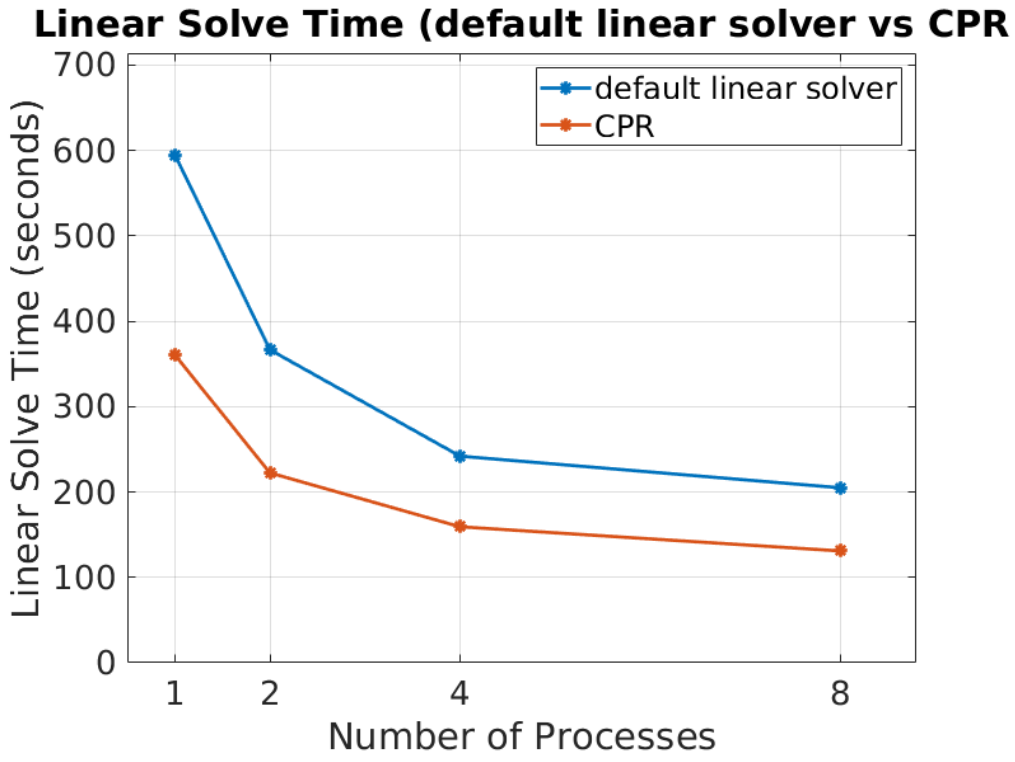
Conclusion:
Scaling can vary significantly with hardware and simulation case, but it is usually worth it to run in parallel!

Tip 2: Use faster linear solvers!

CPR is activated with ”CPR” deck keyword, or on command line:

`--linear-solver=cpr`





Linear solver advanced tips


```
{
    "maxiter": "20",
    "tol": "0.0050000000000000001",
    "verbosity": "0",
    "solver": "bicgstab",
    "preconditioner": {
        "type": "cprw",
        "use_well_weights": "false",
        "add_wells": "true",
        "weight_type": "trueimpes",
        "finesmoother": {
            "type": "ParOverILU0",
            "relaxation": "1"
        },
        "verbosity": "0",
        "coarsesolver": {
            "maxiter": "1",
            "tol": "0.10000000000000001",
            "solver": "loopsolver",
            "verbosity": "0",
            "preconditioner": {
                "type": "amg",
                "alpha": "0.33333333333300003",
                "relaxation": "1",
                "iterations": "1",
                "coarsenTarget": "1200",
                "pre_smooth": "1",
                "post_smooth": "1",
                "beta": "0",
                "smoother": "ILU0",
                "verbosity": "0",
                "maxlevel": "15",
                "skip_isolated": "0",
                "accumulate": "1",
                "prolongationdamping": "1",
                "maxdistance": "2",
                "maxconnectivity": "15",
                "maxaggsize": "6",
                "minaggsize": "4"
            }
        }
    }
}
```


Full description of linear solver in *.DBG output

- JSON Format
- Save to “mylinearsolversetup.json”, and you can modify tons of parameters!
  - (Must end with .json for Flow to accept it)
- Run with command line:

`--linear-solver=mylinearsolversetup.json`

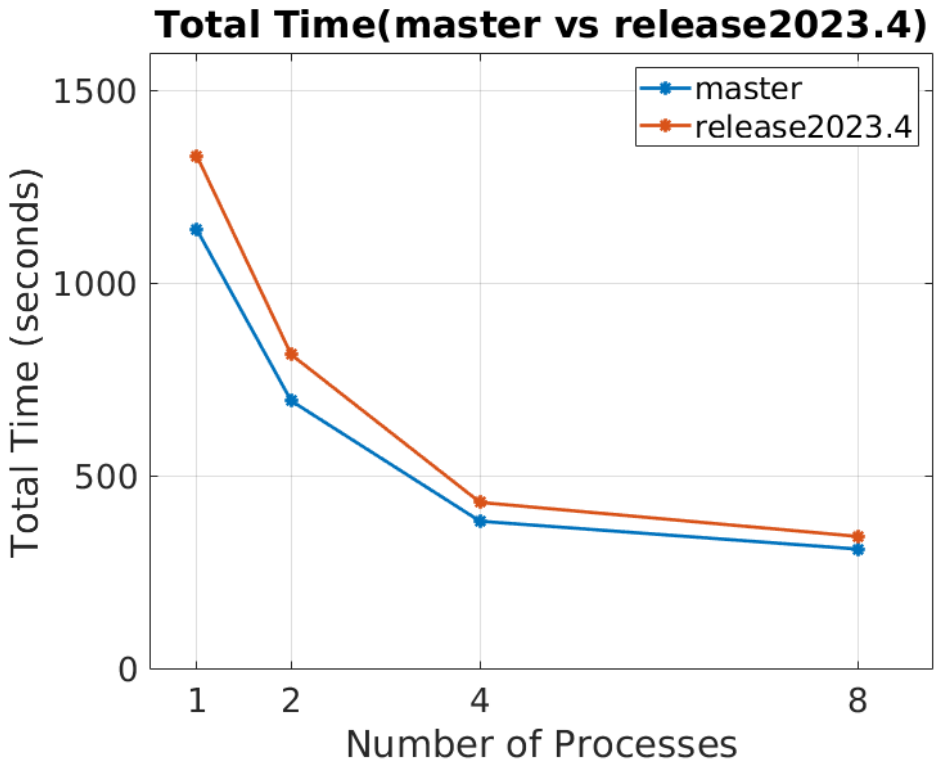
Note: “CPR” option is actually the “CPRW” method recently published.

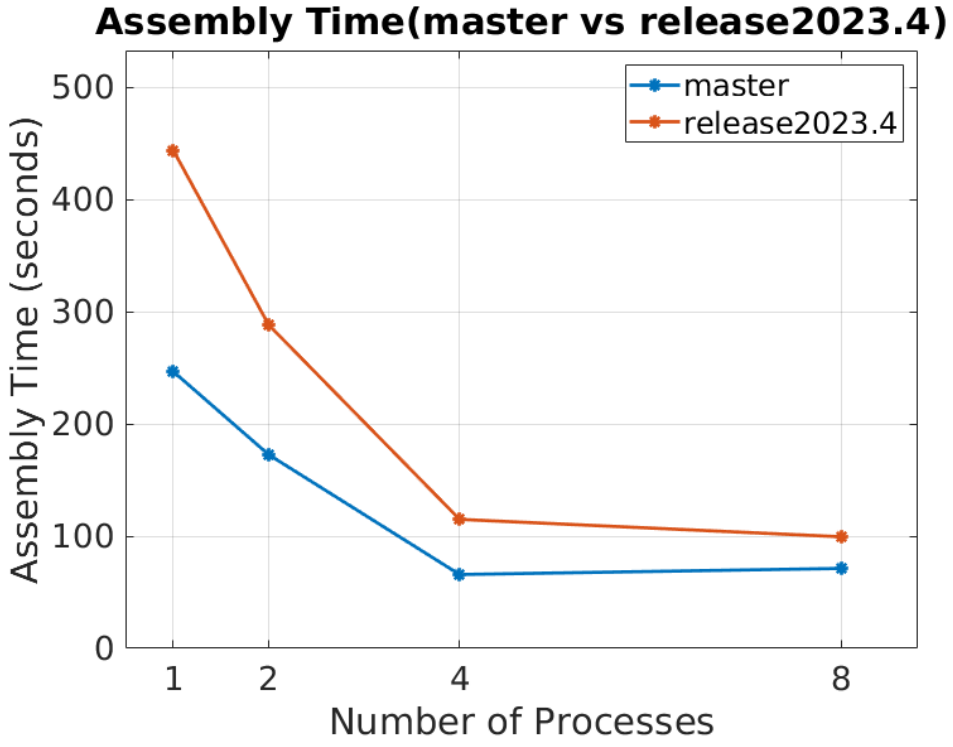
Tip 3: Use fast linearization and assembly!

Improvement for blackoil/CO2STORE in 2022.10, improvement for THERMAL coming in 2023.10.

(So, tip is more precisely: Use a recent version of OPM Flow!)

Comparison on thermal case variant:





Tip 4: Use tuning options!

Flow by default does NOT respect the TUNING keyword

- By using `--enable-tuning=true` you make Flow use it (first record only)

Nonlinear convergence options can be changed on the command line: `tolerance-cnv`, `tolerance-mb`, `tolerance-cnv-relaxed`, `relaxed-max-pv-fraction`, etc.

- See OPM Flow manual for documentation
- Beware! Weakening tolerances may give wrong solution!

For more information about your run: `--output-extra-convergence-info=steps,iterations`

- Will output *.INFOITER and *.INFOSTEP files with iterations, timing etc.

Obscure tuning options

When you know that linear solver problems are frequent:
`--linear-solver-ignore-convergence-failure=true`

- Will try to continue Newton iterations even when linear solver cannot converge fully.

When you think that “this is not complicated, why is it slow”:
`--ecl-enable-drift-compensation=false`

- Beware, this can kill or rescue your runtime!

Secret options not for you…

NDEBUG

- By default, NDEBUG is not set for OPM Flow, so assert()s are left in
- Turn on by setting option WITH_NDEBUG in cmake when compiling Flow
- You may get up to 10% - 15% speed-up (but less security net)

Use `--help-all` to see hidden options (including obsolete ones)

Compile experimental versions of OPM Flow

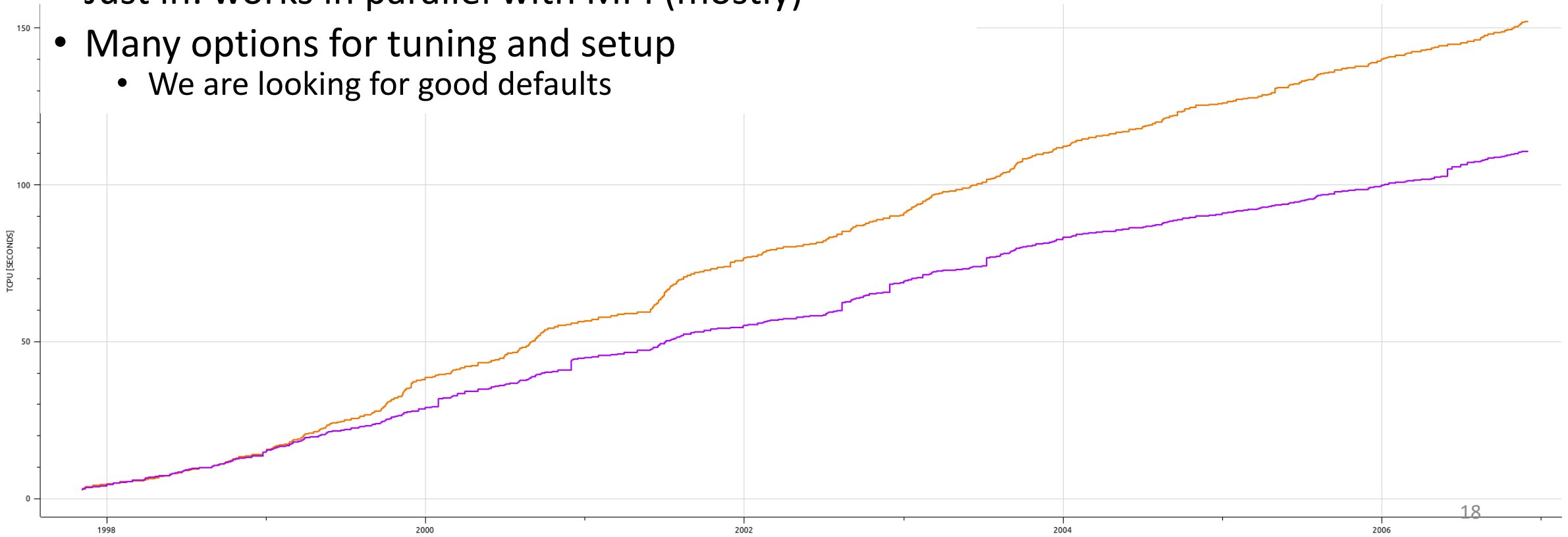
- See for example https://github.com/hnil/opm-flowexperimental

“Tip 4.999”: new nonlinear solvers!

Nonlinear domain decomposition method

- Activate using `--nonlinear-solver=nldd`
- **Not quite mature yet**: may crash!
- Just in: works in parallel with MPI (mostly)
- Many options for tuning and setup
  - We are looking for good defaults

Timing below:
Norne with 6 MPI ranks
using Newton (orange)
or NLDD (purple)



Future improvements

- Faster property evaluation!
  - Shows up as part of “update” time in end-of-run summary
- Improved timestepping logic and algorithms

Acknowledgement

Work done with financial support from

equinor

CLIMIT

EuroHPC Joint Undertaking

Thanks for listening!