Performance acceleration for CO2 simulations

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

SINTEF
Introduction and example
Understanding OPM performance

1. Read deck, setup and initialization
2. Start timestep
3. Newton loop
   - Assembly and linearization
   - Converged?
     - Yes
     - No
       - Iteration > limit?
         - Yes
         - No
1. Solve linear system
2. Update state variables
3. Timestep successful
   - Output and report
   - Increase t, find new $\Delta t$
4. Timestep failed
   - Reset t, reduce $\Delta t$

(ignoring reaching the end of the simulation time)
What causes performance loss?

- Newton loop
  - Assembly and linearization
  - Converged?
    - Yes
    - No
    - Iteration > limit?
      - Yes
      - No
      - Solve linear system
      - Update state variables
  - Solve linear system
  - Update state variables

What takes the most time?
- Assembly of residual and Jacobian
- Solving linear system

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 takes the most time?

<table>
<thead>
<tr>
<th>Description</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of MPI processes</td>
<td>8</td>
</tr>
<tr>
<td>Threads per MPI process</td>
<td>1</td>
</tr>
<tr>
<td>Number of timesteps</td>
<td>271</td>
</tr>
<tr>
<td>Total time (seconds)</td>
<td>7611.14</td>
</tr>
<tr>
<td>Solver time (seconds)</td>
<td>7610.13</td>
</tr>
<tr>
<td>Assembly time (seconds)</td>
<td>110.23</td>
</tr>
<tr>
<td>Well assembly (seconds)</td>
<td>23.56</td>
</tr>
<tr>
<td>Linear solve time (seconds)</td>
<td>4909.71</td>
</tr>
<tr>
<td>Linear setup (seconds)</td>
<td>280.46</td>
</tr>
<tr>
<td>Update time (seconds)</td>
<td>472.65</td>
</tr>
<tr>
<td>Pre/post step (seconds)</td>
<td>106.13</td>
</tr>
<tr>
<td>Output write time (seconds)</td>
<td>1.93</td>
</tr>
<tr>
<td>Overall Linearizations</td>
<td>1579</td>
</tr>
<tr>
<td>Overall Newton Iterations</td>
<td>1410</td>
</tr>
<tr>
<td>Overall Linear Iterations</td>
<td>43554</td>
</tr>
</tbody>
</table>

Failed: 259.6; 42.5% | Failed: 209.4; 44.3% | Failed: 671; 42.5% | Failed: 671; 47.6% | Failed: 25942; 59.6%
What can help? Obscure options?

In this particular case, the option
--linear-solver-ignore-convergence-failure=true
What can help? Change linear solver?

Using the CPR linear solver

---

Using the CPR linear solver

---

**End of simulation**

---

- **Number of MPI processes**: 8
- **Threads per MPI process**: 1
- **Number of timesteps**: 21
- **Total run time (seconds)**: 308.78
- **Solver time (seconds)**: 102.47
- **Assembly time (seconds)**: 64.97
- **Well assembly (seconds)**: 3.11
- **Linear solve time (seconds)**: 178.12
- **Linear setup (seconds)**: 109.47
- **Update time (seconds)**: 54.49
- **Pre/post step (seconds)**: 6.55
- **Output write time (seconds)**: 2.07
- **Overall Linearizations**: 159 (Failed: 0.0 %)
- **Overall Newton Iterations**: 138 (Failed: 0.0 %)
- **Overall Linear Iterations**: 74 (Failed: 0.0 %)
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

Scalability test for OPM-flow

- Number of Processes: 1, Total Time: 1141.2 seconds
- Number of Processes: 2, Total Time: 696.8 seconds
- Number of Processes: 4, Total Time: 384.7 seconds
- Number of Processes: 8, Total Time: 312.5 seconds
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

[Graphs showing Total Time and Linear Solve Time comparisons]
Linear solver advanced tips

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:

![Graphs showing performance comparison between master and release2023.4]
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)
on 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

Thanks for listening!