# Well Solver on GPUs

**Vinícius O. Martins***[1], Alf Birger Rustad[2], Alberto Saa[1], Razvan Nane[3]

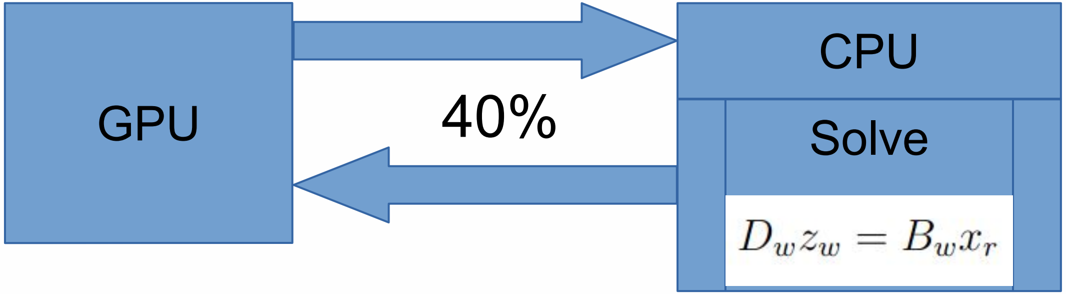
[1]Universidade Estadual de Campinas, Brazil
[2]Equinor, Norway
[3]BigDataAccelerate, Netherlands

* *vmartins@ime.unicamp.br*

# Motivation

- OPM is seeking a “full” simulator on GPU
- The multisegment well model is missing on GPU
- As well contributions can be applied during the reservoir solver, it can lead to unnecessary data transfer if it is not implemented on GPU

Norne 1A



$$D_w z_w = B_w x_r$$

# GPUs

- Originally created to process data for computer displays
- GPGPUs are used to accelerate scientific computing
- Much more computer units than CPUs
- Highly parallel computations

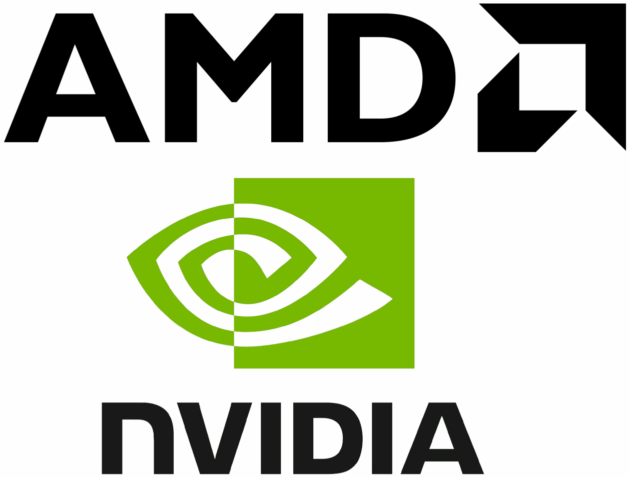


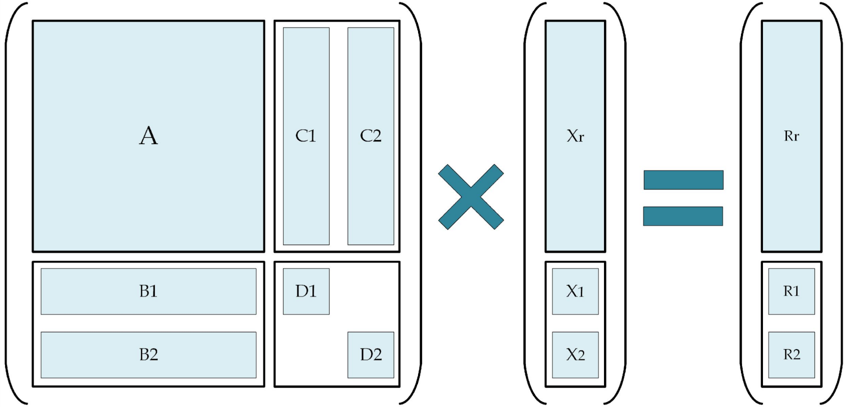
Figure 2: AMD and NVIDIA logos

# Flow on GPUs

- Solve the simulator linear system
- Standard Well Model apply method
- Supports GPUs from different vendors; AMD, NVIDIA, and Intel
- Command line options to run on GPU:

  --accelerator-mode = [none, cusparse, opencl, amgcl, rocalution, rocsparse] (release/2024.04)

# Schur Complement



- We do not calculate D^{-1} for the multisegment wells
- BiCGStab method with ILU0 preconditioner

$$\left(A - \sum_w C_w D_w^{-1} B_w\right) x_r = R_r - \sum_w C_w D_w^{-1} R_w$$

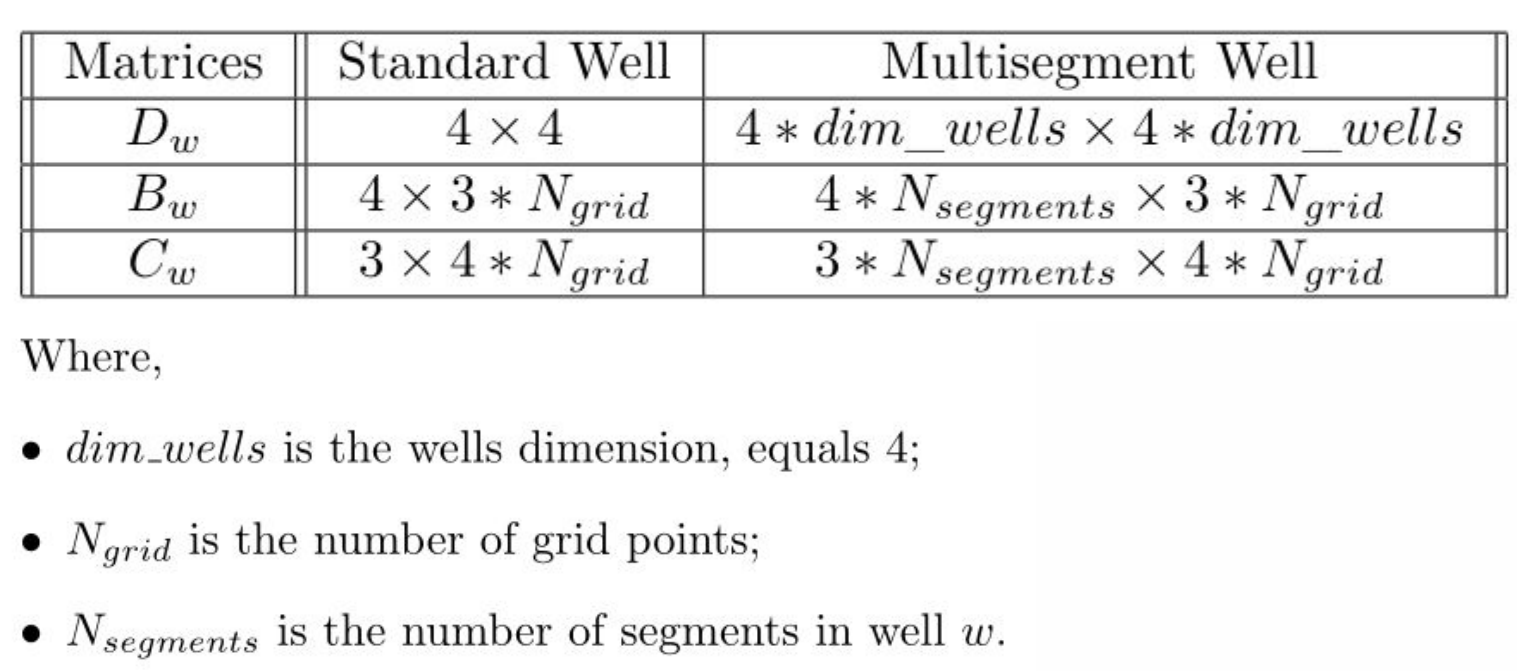
$$x_w = D_w^{-1}(R_w - B_w x_r)$$

# Well Contributions

/ --matrix-add-well-contributions=true $$A^* = A - \sum_w C_w D_w^{-1} B_w$$

/ --matrix-add-well-contributions=false $$Ax_r - \sum_w C_w D_w^{-1} B_w x_r$$

# Multisegment Wells



| Matrices | Standard Well | Multisegment Well |
|---|---|---|
| $D_w$ | $4 \times 4$ | $4 * dim\_wells \times 4 * dim\_wells$ |
| $B_w$ | $4 \times 3 * N_{grid}$ | $4 * N_{segments} \times 3 * N_{grid}$ |
| $C_w$ | $3 \times 4 * N_{grid}$ | $3 * N_{segments} \times 4 * N_{grid}$ |

Where,

- $dim\_wells$ is the wells dimension, equals 4;
- $N_{grid}$ is the number of grid points;
- $N_{segments}$ is the number of segments in well $w$.

# Multisegment Wells

$$\begin{cases} D_w z_w = B_w x_r \\ D_w s_w = R_w \\ D_w x_w = R_w - B_w x_r \end{cases} \Rightarrow \begin{cases} D_w z_w = B_w x_r & (1) \\ D_w s_w = R_w & (2) \\ x_w = s_w - z_w & (3) \end{cases}$$

/ Two additional linear systems for each well for each linear iteration

# Moving between CPU and GPU

/ Multisegment well flow solved in CPU

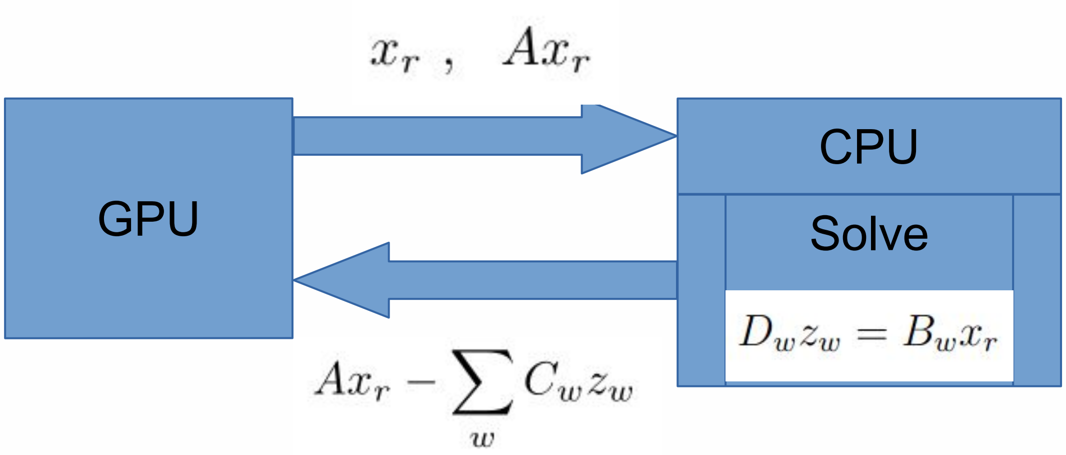
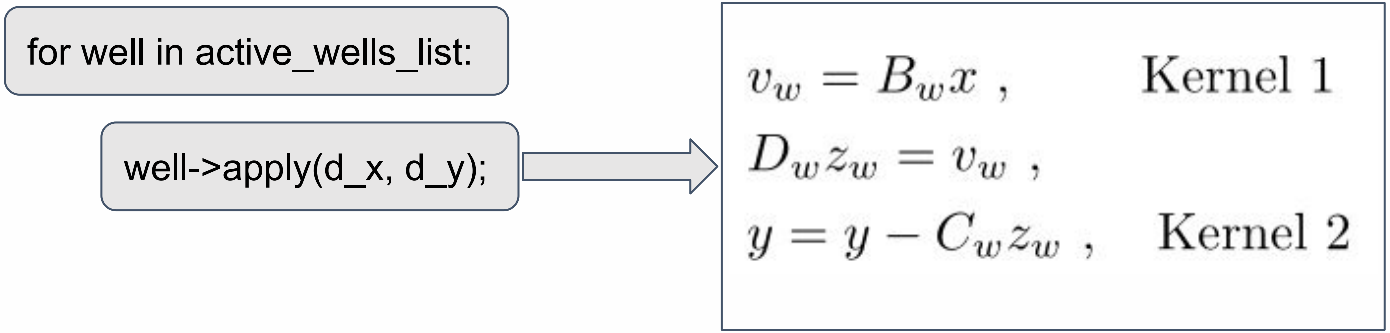


Figure 4: Data transfer diagram

# apply(...) Method

- bdabridge
- apply(w,x,y)

- Factorization is done during object MultisegmentWellContribution construction
- Called twice for each active well (each linear iteration)



```
for well in active_wells_list:
    well->apply(d_x, d_y);
```

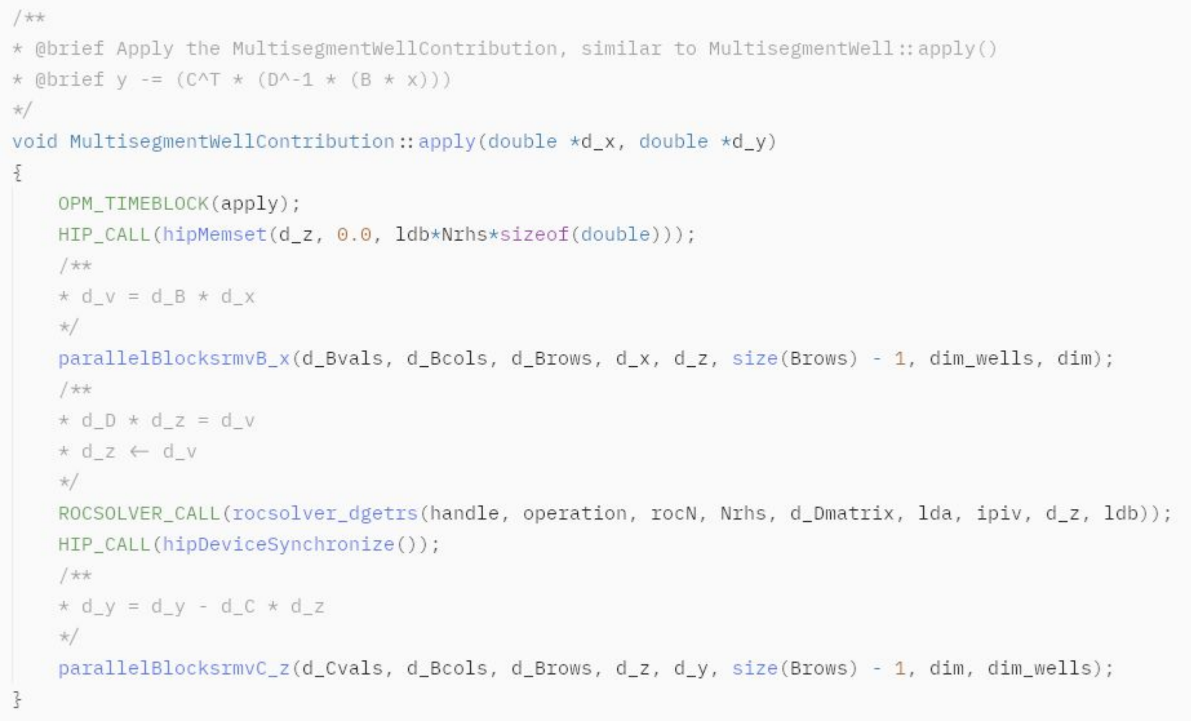
$$v_w = B_w x \; , \qquad \text{Kernel 1}$$

$$D_w z_w = v_w \; ,$$

$$y = y - C_w z_w \; , \qquad \text{Kernel 2}$$

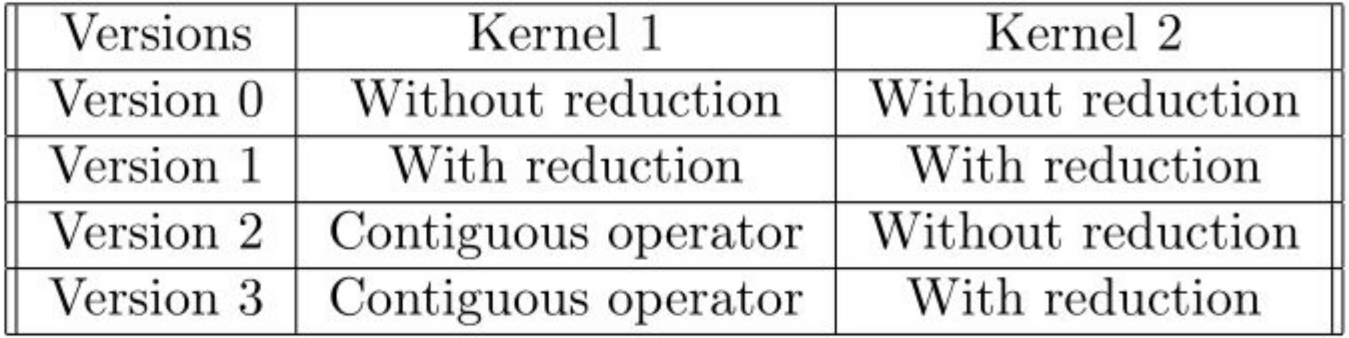
# apply(...) Method

- SPMV operations done with Hip kernels
- Dense LU factorization using partial pivoting with row interchanges
- Triangular solver

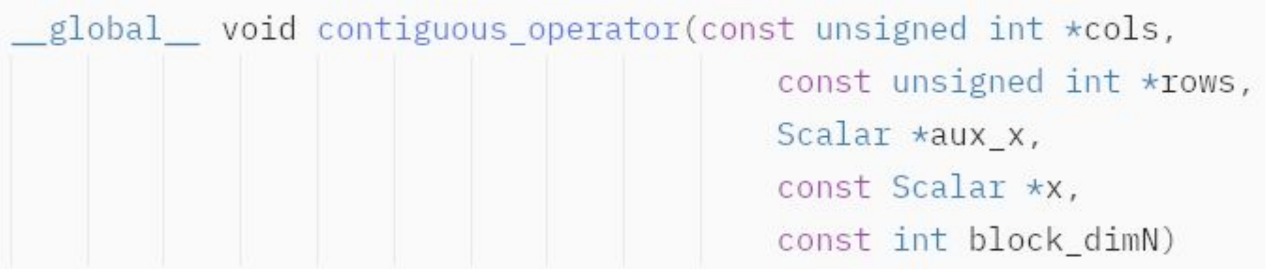


```
/**
* @brief Apply the MultisegmentWellContribution, similar to MultisegmentWell::apply()
* @brief y -= (C^T * (D^-1 * (B * x)))
*/
void MultisegmentWellContribution::apply(double *d_x, double *d_y)
{
    OPM_TIMEBLOCK(apply);
    HIP_CALL(hipMemset(d_z, 0.0, ldb*Nrhs*sizeof(double)));
    /**
    * d_v = d_B * d_x
    */
    parallelBlocksrmvB_x(d_Bvals, d_Bcols, d_Brows, d_x, d_z, size(Brows) - 1, dim_wells, dim);
    /**
    * d_D * d_z = d_v
    * d_z ← d_v
    */
    ROCSOLVER_CALL(rocsolver_dgetrs(handle, operation, rocN, Nrhs, d_Dmatrix, lda, ipiv, d_z, ldb));
    HIP_CALL(hipDeviceSynchronize());
    /**
    * d_y = d_y - d_C * d_z
    */
    parallelBlocksrmvC_z(d_Cvals, d_Bcols, d_Brows, d_z, d_y, size(Brows) - 1, dim, dim_wells);
}
```

# apply(...) Method



| Versions | Kernel 1 | Kernel 2 |
|---|---|---|
| Version 0 | Without reduction | Without reduction |
| Version 1 | With reduction | With reduction |
| Version 2 | Contiguous operator | Without reduction |
| Version 3 | Contiguous operator | With reduction |



```
__global__ void contiguous_operator(const unsigned int *cols,
                                        const unsigned int *rows,
                                        Scalar *aux_x,
                                        const Scalar *x,
                                        const int block_dimN)
```

# Partial Results

/ Norne case with multisegment wells (opm-tests  repository)

Machine

- CPU: AMD Ryzen 9 7900
- GPU: AMD Radeon PRO W7900

Command line options

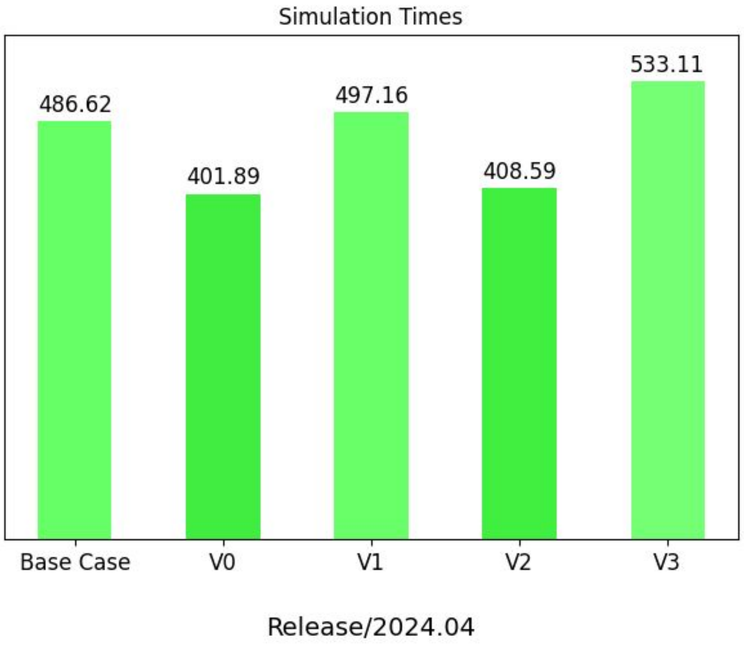
/ --threads-per-process=1

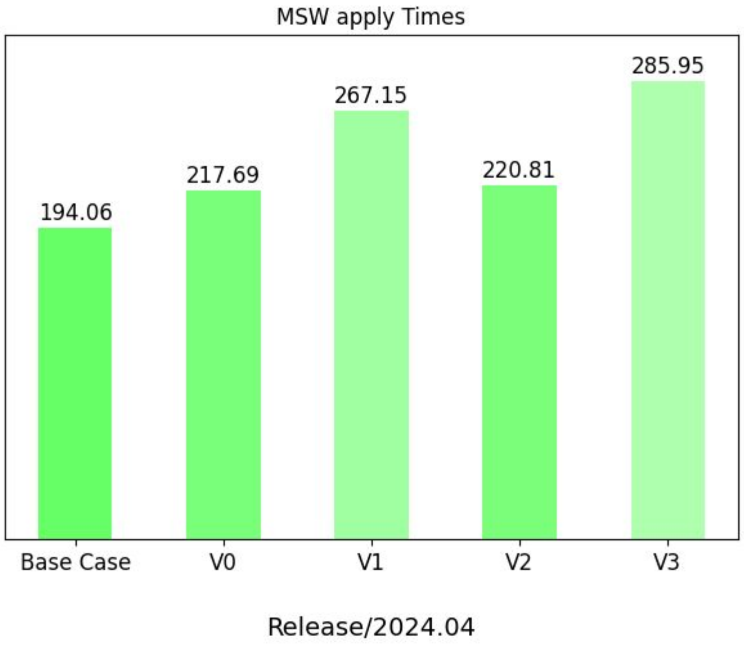
/ --accelerator-mode=rocsparse

/ --matrix-add-well-contributions=false

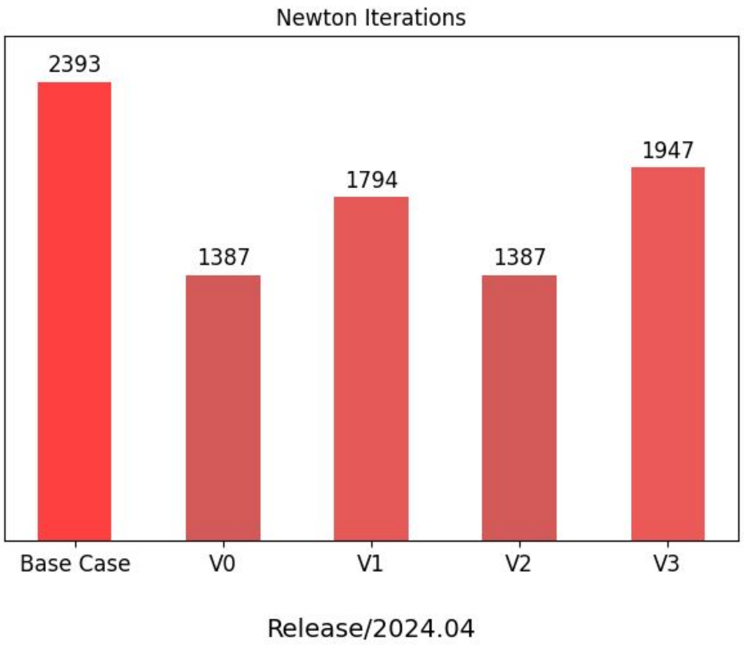
/ --linear-solver=ilu0

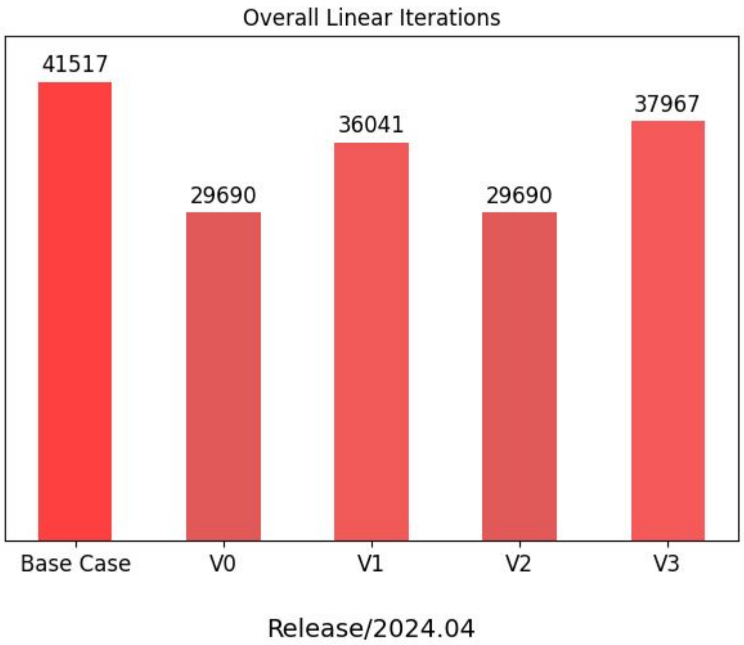
# Partial Results (Norne 1A)





# Partial Results (Norne 1A)





# Next Steps

- / Apply multisegment wells in parallel
- / Solve linear system with sparse LU
- / Measure the impact on bigger models. More wells and more segments.
- / Test strategy without Schur Complement

# Acknowledgements

epic
Research & Education

UNICAMP

equinor

IMECC

CEPETRO
Centro de Estudos de Energia e Petróleo

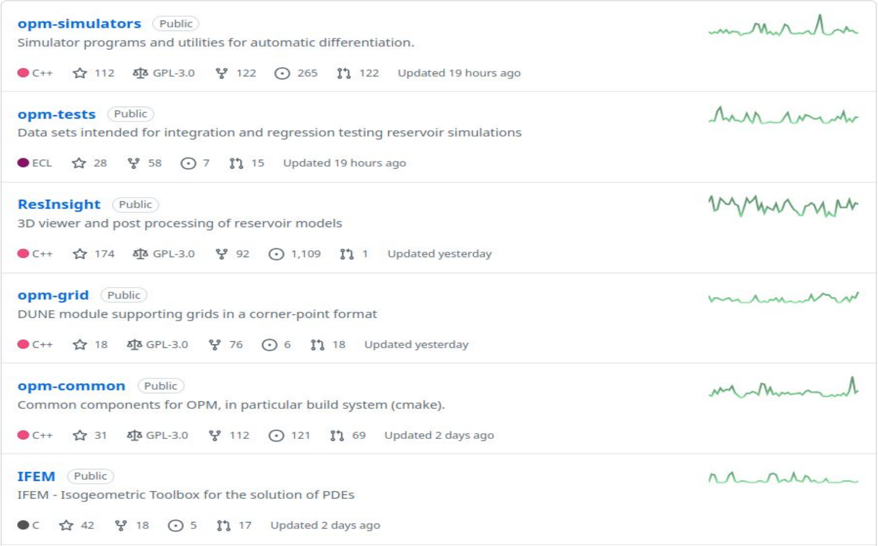
anp
Agência Nacional
do Petróleo,
Gás Natural e Biocombustiveis

FAPESP

# OPM Flow

- Open Source reservoir simulator
- Three-phase black-oil fully implicit
- CO2 storage, thermal simulation, and EOR fluids
- Currently developed by several institutions

OPM
OPEN POROUS MEDIA



**opm-simulators** Public
Simulator programs and utilities for automatic differentiation.
● C++ ☆ 112 ⚖ GPL-3.0 ⑂ 122 ⊙ 265 ⇅ 122 Updated 19 hours ago

**opm-tests** Public
Data sets intended for integration and regression testing reservoir simulations
● ECL ☆ 28 ⑂ 58 ⊙ 7 ⇅ 15 Updated 19 hours ago

**ResInsight** Public
3D viewer and post processing of reservoir models
● C++ ☆ 174 ⚖ GPL-3.0 ⑂ 92 ⊙ 1,109 ⇅ 1 Updated yesterday

**opm-grid** Public
DUNE module supporting grids in a corner-point format
● C++ ☆ 18 ⚖ GPL-3.0 ⑂ 76 ⊙ 6 ⇅ 18 Updated yesterday

**opm-common** Public
Common components for OPM, in particular build system (cmake).
● C++ ☆ 31 ⚖ GPL-3.0 ⑂ 112 ⊙ 121 ⇅ 69 Updated 2 days ago

**IFEM** Public
IFEM - Isogeometric Toolbox for the solution of PDEs
● C ☆ 42 ⑂ 18 ⊙ 5 ⇅ 17 Updated 2 days ago

Figure 1: OPM Github repositories.

# Multisegment Wells

$$R_{\alpha,w,s} = \frac{A_{\alpha,w,s} - A^0_{\alpha,w,s}}{\Delta t} + Q_{\alpha,s} - \sum_{j \in C(w,s)} q_{\alpha,j} - \sum_{k \in I(w,s)} Q_{\alpha,k} = 0$$

$$R_{p,s} = p_s - p_o - H_h - H_f - H_a$$

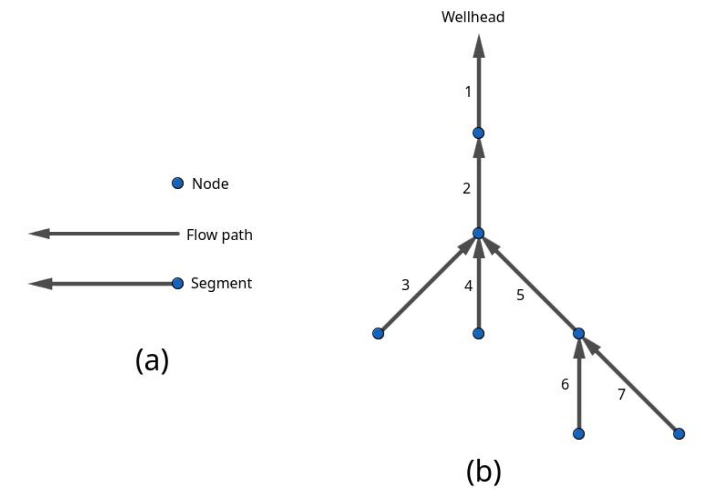


Figure 3: (a) segment elements; (b) multisegment well.

For the top segment:

$$R_{c,w} = p_{bhp,w} - p^{target}_{bhp,w} = 0$$

$$R_{c,w} = Q_\alpha - Q^{target}_\alpha = 0$$

# Partial Results

/ Norne cases with multisegment wells (opm-tests  repository)

Machine 1

- CPU: AMD Ryzen 9 7900
- GPU: AMD Radeon PRO W7900

Machine 2

- CPU: AMD Epyc 7763
- GPU: AMD Instinct Mi210

Command line options

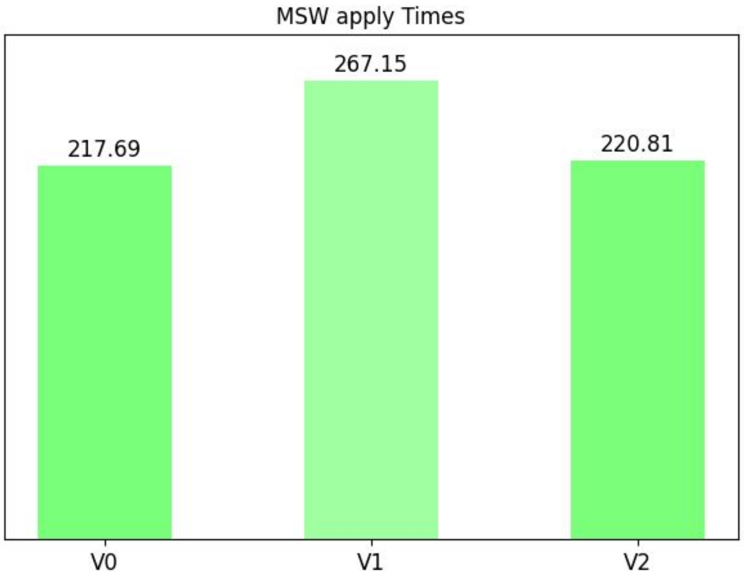
/ --threads-per-process=1

/ --accelerator-mode=rocsparse

/ --matrix-add-well-contributions=false

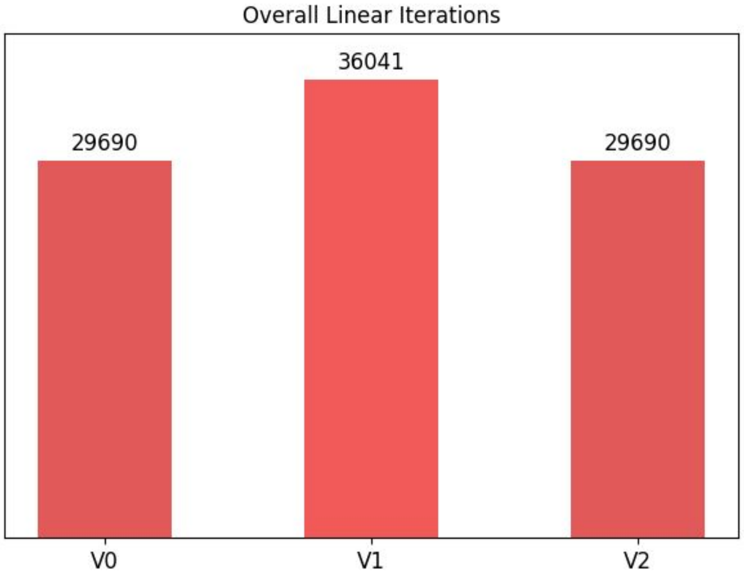
/ --linear-solver=ilu0

# Partial Results



Machine 1 - release/2024.04; 194.06 iterations

# Partial Results



Machine 1 - release/2024.04; 41517 iterations