UiO : University of Oslo

Andreas Thune

High performance computing for reservoir simulation

Thesis submitted for the degree of Philosophiae Doctor

Department of Informatics
Faculty of Mathematics and Natural Sciences

Simula Research Laboratory

2023

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 1234

ISSN 1234-5678

Cover: Hanne Baadsgaard Utigard.
Print production: Reprosentralen, University of Oslo.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of *Philosophiae Doctor* at the University of Oslo. The research presented here was conducted at Simula Research Laboratory under the supervision of Professor Xing Cai, Dr. Alf Birger Rustad, Professor Tor Skeie and Professor Ernst Gunnar Gran. This work was partially supported by the Norwegian Research Council through grant 237898.

The thesis is a collection of three papers, presented in chronological order of writing. The papers are preceded by an introductory chapter that relates them to each other and provides the background information and motivation for the work. All papers in this thesis are joint papers.

Acknowledgements

I would like to express my sincere gratitude to my main supervisor Xing Cai for his invaluable guidance and support throughout the project. I value your knowledge, experience and attention to detail, and I feel deep appreciation for the assistance you have given me over the years.

Many thanks also go to Alf B. Rustad, whose scientific and practical contributions I could not have been without. Your genuine interest in the performance of *Flow* has been a great source of motivation. Next, I thank Tor Skeie for his encouragement and involvement throughout the years and Ernst Gunnar Gran for his efforts in developing the project.

I want to thank the broad OPM community. Open and active developers make code contributions a very smooth process. Markus Blatt deserves a special mention for helping with implementation and valuable comments.

I am grateful to Simula Research Laboratory for providing an exceptional learning environment during my PhD studies. I am especially thankful to all my collogues in the HPC department and my office mates James D. Trotter, Masoud Hemmatpour, Tore H. Larsen, Erik H. Sæternes, José Rocher-González and Asep Maulana for many scientific discussions, coffee breaks and the occasional out of office activities. Tore also needs recognition for his hard work on maintaining the eX3 platform.

Finally, I give thanks to my parents for their unwavering support.

Andreas Thune
Oslo, April 2023

List of Papers

Paper I

Thune, A., Cai, X., and Rustad, A.B. "On the impact of heterogeneity-aware mesh partitioning and non-contributing computation removal on parallel reservoir simulations". In: *Journal of Mathematics in Industry.* Vol. 11, (2021), pp. 1-23. DOI: 10.1186/s13362-021-00108-5.

Paper II

Thune, A., Reinemo, S.-A., Skeie, T., and Cai, X. "Detailed modeling of heterogeneous and contention-constrained point-to-point MPI communication". In: *IEEE Transactions on Parallel and Distributed Systems.* (2023) DOI: 10.1109/TPDS.2023.3253881.

Paper III

Thune, A., Blatt, M., Cai, X., and Rustad, A.B. "Distributed wells for enhancing the efficiency of parallel reservoir simulation". In preparation

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Reservoir simulation is an important tool in the petroleum industry that involves the use of computational models to predict and analyse the flow of fluids in oil and gas reservoirs. Computer simulations complement information gained by field observations, laboratory tests and analytical models, and provide essential assistance in determining the potential amount of producible hydrocarbons in a field. These estimates of potential value are used in the industry to guide investment decisions [15]. Reservoir simulation also helps engineers in positioning and management of wells to optimize oil and gas recovery. In the domain of carbon capture and storage (CCS), reservoir simulation is used to predict the flow of CO_2 stored in underground reservoirs. CCS is an important climate mitigation measure on the path to net zero emissions [21, 33]. To ensure the success of CO_2 storage, the long-term impact of the CO_2 injection into subsurface reservoirs must be understood so that leaks are avoided.

Achieving acceptable accuracy in reservoir simulations requires reasonably high mesh resolution, and because typical petroleum reservoirs are gigantic in size, a grid representation of the reservoirs can contain many millions of cells. In addition, the span of oil and gas production is decades long, and in the case of CCS, the time scale of interest is even longer. Coupled with complex mathematical models, this makes reservoir simulation computationally challenging, and the need for and benefit of high performance computing (HPC) are self-evident. Speeding up simulations of reservoir models of a moderate size also improves the workflow of reservoir engineers, who often need to run multiple realizations of the same oil field to test different scenarios.

The topic of this thesis is HPC for reservoir simulation, and the aim is to improve and understand the performance of an established reservoir simulator of relevant real-world reservoir models. The research work of this thesis contributes to the development of an open-source reservoir simulation software, that has both commercial and scientific users, which means that the work of the thesis will have an industrial impact. The need for computing power is not limited to the domain of reservoir simulation, and many challenges to achieving good performance and successful large-scale reservoir simulations are shared between other applications. Although the research of this thesis is aimed at reservoir simulation-specific aspects, it also has relevance in the more broad field of HPC for scientific computing.

In this study of HPC for reservoir simulations, the research followed three main directions. First, we considered domain decomposition strategies for parallel reservoir simulations and their impact on performance and numerical effectiveness. Because of heterogeneity in the geological properties of petroleum reservoirs, specialized partitioning strategies are needed for good convergence in

the parallel linear solver. We introduced a graph partitioning edge-weighting scheme that incorporates reservoir heterogeneity while still partially maintaining the aspect of communication volume minimization in graph partitioning.

Second, we investigated how to perform large-scale reservoir simulations efficiently on modern computing platforms. Understanding the underlying hardware architecture is useful when aiming to achieve resource efficient simulations. Modern parallel computing platforms are increasingly heterogeneous, combining CPUs and accelerators, such as GPUs. Heterogeneity is also present within CPUs, that can be made up of multiple sockets and NUMA domains, and in networks where interconnect bandwidth and latency between nodes depend on their relative placement in the network topology.

Third, we studied efficient parallelization of linear solvers for reservoir simulation, and the impact of the preconditioning strategy on the overall performance. For large-scale reservoir simulations, solving sparse and ill-conditioned linear systems makes up the majority of simulator execution time. The heterogeneity present in linear systems, arising in simulations of fluid in petroleum reservoirs, creates challenges when parallelizing the linear solver.

The rest of the thesis introduction is organized as follows. Section 1.1 gives a brief background to the relevant topics considered throughout the thesis. In Section 1.2, the main research questions are presented. Then, in Section 1.3, we summarize the three research papers that make up the thesis. Section 1.4 discusses the limitations of the thesis before concluding remarks follow in Section 1.5.

1.1 Background

In this section, some topics important for understanding this thesis are presented.

1.1.1 Black-oil model

The most common model for describing fluid flow in petroleum reservoirs is the black-oil model [3, 12]. This is a model for three-phase flow in porous media, the three fluid phases being water, oil and gas. To allow for mass transfer between the oil and gas phases, the model operates with three components, one for each phase. The equations of the black-oil model are derived from Darcy's law and conservation of mass for each component. The phase quantities are denoted by subscripts $\alpha = \{w, o, g\}$ representing water, oil and gas.

$$\frac{\partial}{\partial t}\left[\frac{\phi S_w}{B_w}\right] = \nabla \cdot \left[\frac{\lambda_w K}{B_w}\nabla\Phi_w\right] + q_w, \tag{1.1}$$

$$\frac{\partial}{\partial t}\left[\frac{\phi S_o}{B_o}\right] = \nabla \cdot \left[\frac{\lambda_o K}{B_o}\nabla\Phi_o\right] + q_o, \tag{1.2}$$

$$\frac{\partial}{\partial t}\left[\phi\left(\frac{R_s S_o}{B_o} + \frac{S_g}{B_g}\right)\right] = \nabla \cdot \left[R_s\frac{\lambda_o K}{B_o}\nabla\Phi_o + \frac{\lambda_g K}{B_g}\nabla\Phi_g\right] + R_s q_o + q_{fg}. \tag{1.3}$$

Here ϕ and K are rock porosity and permeability, while S_α, B_α, λ_α, R_s and q_α respectively denote saturation, formation volume, mobility, gas solubility and a source/sink term. The phase potential Φ_α is defined using pressure p_α, density ρ_α, gravitational constant γ and reservoir depth z

$$\Phi_\alpha = p_\alpha + \rho_\alpha \gamma z. \tag{1.4}$$

The unknowns of the system are the fluid phase saturations S_α and pressures p_α. The saturation unit relation and empirical relations between saturation and capillary pressure relations are used to close the system.

$$S_w + S_o + S_g = 1, \tag{1.5}$$
$$p_w = p_o - p_{cow}(S_w), \qquad p_g = p_o + p_{cog}(S_g). \tag{1.6}$$

The capillary pressures p_{cow} and p_{cog} describe the discontinuity in pressure between the water and oil phase and between the oil and gas phase. Most of the fluid and rock properties in the black-oil equations can be expressed as functions of saturation or pressure, making the (1.1)-(1.3) equations non-linear.

Discretization and Transmissibility

Industry standard reservoir simulators favour the use of lower-order finite volume schemes in space and fully implicit in-time finite difference schemes for the discretization of the black-oil equations. The stability of implicit in-time schemes allows the use of large time steps, that are needed to complete simulations of the decade-long production of petroleum reservoirs in a reasonable time, but also mean that the discretized and linearized black-oil equations will take the form of a large linear system of equations, that is complicated to solve.

The reservoir simulator focused on in this thesis implements a cell-centred finite-volume discretization of the black-oil equations that use a two-point flux approximation with upwind mobility weighing [25]. In time, we use the backward Euler scheme. Let $S_\alpha^{\ell,i}$ be the saturation of phase α at time-step ℓ in cell C_i, a discretized version of the water equation (1.1) in cell C_i will then look as follows:

$$\frac{V_i \phi^i}{\Delta t}\left(\frac{S_w^{\ell+1,i}}{B_w^{\ell+1,i}} - \frac{S_w^{\ell,i}}{B_w^{\ell,i}}\right) - \sum_{j \in nab(i)} \lambda_w^{\ell+1,ij} T_{ij} (\Phi_w^{\ell+1,i} - \Phi_w^{\ell+1,j}) = V_i q_w^{\ell+1,i}. \tag{1.7}$$

V_i represents the volume of cell C_i, while $nab(i)$ contains the indices of cells with a geometric connection to cell C_i. The ij indexation means the quantity is defined on the face Γ_{ij} connecting cell C_i and C_j.

Water saturation S_w, oil pressure p_o and gas saturations S_g are used as primary unknowns of the discretized black-oil equations. When the gas phase is dissolved in the oil phase, the gas solubility factor R_s is used instead of gas saturation.

The term T_{ij} represents transmissibility. This is a static quantity defined by permeability in the adjoining cells and the geometry of C_i, C_j and Γ_{ij}.

$$T_{ij} = m_{ij}|\Gamma_{ij}| \left(\frac{\|\vec{c}_i\|^2}{\vec{n}_i K_i \vec{c}_i} + \frac{\|\vec{c}_j\|^2}{\vec{n}_j K_j \vec{c}_j} \right)^{-1}. \tag{1.8}$$

The transmissibility multiplier m_{ij} makes it possible to incorporate the presence of faults in the reservoir. Because it is static and derived by permeability, the transmissibility defined in (1.8) is an important quantity in reservoir simulations. The heterogeneity in permeability in the reservoir is reflected in the transmissibility, which will to a large degree, determine the heterogeneity of the linear systems of equations formed by the discretized and linearized black-oil equations. The transmissibility is static in the sense that it remains constant throughout the simulation, and can therefore be used when we partition the mesh at the beginning of each parallel simulation.

The standard well model

Wells connect the surface to the subsurface reservoirs, and understanding how wells impact flow in the reservoir is a key aspect of reservoir simulation. In the black-oil equations in (1.1)-(1.3), wells are represented in the source/sink terms q_α. Typically, the wells impact flow near the wellbore on a finer scale than the resolution of the reservoir mesh. Special well models are included to better capture the well-reservoir interaction.

The standard well model for single-phase flow, formulated in [30], expresses the flow rate of a well located inside a grid block in terms of the difference in pressure between the wellbore and the average grid block pressure. The wellbore pressure in the grid cell that contains the endpoint of the well is referred to as the bottom-hole pressure p_{bhp}, and flow rate q_i in the well connected cell i is defined in terms of bottom-hole pressure and cell pressure p_i:

$$q_i = J_i(p_i - (p_{bhp} + H_i)). \tag{1.9}$$

The factor J_i is based on connection transmissibility and mobility. Mobility is calculated using the pressure-dependent fluid viscosity and formation volume factor properties, as well as relative permeability in the multi-phase case. H_i is the pressure difference between bottom-hole depth and pressure at the depth of connection i.

For wells that perforate multiple grid cells, we end up with a flow rate for each well completion. These relate to each other through the inflow performance relationship presented in [45], using the total surface flow rate Q_t. By conservation, Q_t equals the sum of the flow rates q_i over all well connected cells:

$$Q_t - \sum_i q_i = 0. \tag{1.10}$$

The well-reservoir interaction for single-phase flow is modelled by introducing bottom-hole pressure p_{bhp} and total surface flow rate Q_t. However, the relationships in (1.9)-(1.10) combine to a single equation. The system is closed by introducing well controls that govern one of the two parameters. These controls are either a desired upper surface flow rate or a lower bottom-hole pressure threshold:

$$p_{bhp} - p_{bhp}^{target} = 0, \tag{1.11}$$

$$Q_t - Q_t^{target} = 0. \tag{1.12}$$

The standard well model for single-phase flow represented in (1.9)-(1.12) is extended to three-phase flow in [20]. Here, the total flow rate is redefined, and two new variables, weighted water and gas fraction of the total well flow F_w and F_g, are introduced:

$$Q_t = \sum_\alpha g_\alpha Q_\alpha, \quad F_w = \frac{g_w Q_w}{Q_t}, \quad F_g = \frac{g_g Q_g}{Q_t} \tag{1.13}$$

Q_α and g_α are flow rate and weighting factor of phase component α.

1.1.2 Numerical solution strategy

The discretization of the black-oil equations described in Section 1.1.1 results in a large, fully implicit system of non-linear equations. Newton's method is used to solve the non-linear system, which can be expressed in a compact residual form as $F(y) = 0$. Starting with an initial guess y_0, a new solution estimate is produced with Newton's method by solving the equation

$$F'(y_{n-1})(y_n - y_{n_1}) = -F(y_{n-1}). \tag{1.14}$$

The iteration stops when $F(y_n)$ is smaller than a given tolerance. The main computational challenge of each Newton iteration is first the creation of the Jacobian matrix $F'(y_{n-1})$, and then solving the linear system of equations. The work of this thesis primarily concerns the second part, related to solving the linear system.

The structure of a complete linearized system of equations for a case with L wells will take the following form:

$$\begin{bmatrix} A & C_1 & C_2 & \cdots & C_L \\ B_1 & D_1 & 0 & \cdots & 0 \\ B_2 & 0 & D_2 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ B_L & 0 & 0 & \cdots & D_L \end{bmatrix} \begin{pmatrix} x \\ x_1^w \\ x_2^w \\ \vdots \\ x_L^w \end{pmatrix} = \begin{pmatrix} f \\ f_1^w \\ f_2^w \\ \vdots \\ f_L^w \end{pmatrix}. \tag{1.15}$$

The A matrix contains the linearized fluid flow equations and $D_1, D_2, ..., D_L$, the well equations for the L wells. The entries of the B and C matrices represent

couplings of the fluid and well models. Instead of forming the full system as stated in (1.15), the well equations are incorporated by forming the Schur complement:

$$\left(A - \sum_{i=1}^{L} C_i D_i^{-1} B_i \right) x = f - \sum_{i=1}^{L} C_i D_i^{-1} f_i^w. \tag{1.16}$$

Using the Schur complement formulation is practical because the system $S = A - \sum_{i=1}^{L} C_i D_i^{-1} B_i$ will have the same size as the fluid flow part of the system.

1.1.2.1 Iterative linear solvers

The linear systems that arise from the discretized and linearized black-oil equations are sparse, large, non-symmetric and ill-conditioned. Iterative methods [40], rather than direct solvers, are well suited for solving these linear systems. The Krylov-subspace iterative linear solvers, such as the conjugate gradient (CG) method, are the most efficient class of linear solvers. Because the systems that arise in reservoir simulation are not symmetric nor positive definite, the CG method is inapplicable. Instead, the generalized minimum residual (GMRES) method [39] and the stabilized biconjugate gradient (BiCGSTAB) method [41] are appropriate and popular alternatives for use in reservoir simulations.

Solving the linear systems takes up a large part of the overall simulation execution time. A good linear solver is therefore key to reservoir simulator performance. Although different iterative methods differ in details, they are all made up of a few important numerical kernels, which are: vector additions, inner producst, sparse-matrix vector products (SpMV) and preconditioning operations. The execution time of the solver is governed by the performance of these four kernels, especially the SpMV and preconditioning operations because these operations involve the system matrix. Preconditioning is of particular importance because it, to a large degree, determines the convergence rate of the linear solver.

Preconditioning for reservoir simulations

A preconditioner is necessary to be able to efficiently solve the large and ill-conditioned linear systems that arise in simulations of flow in petroleum reservoirs. Preconditioning works by modifying the original system to make it easier to solve. The system

$$Ax = b,$$

where $x, b \in \mathbb{R}^N$ and $A \in \mathbb{R}^{N \times N}$, is preconditioned by applying a matrix M to both sides of the equation:

$$MAx = Mb.$$

Ideally, one wants the matrix M to be an approximation of the inverse of A, making MA easier to invert. Computationally, M should also be cheap to apply and construct, at least in comparison to calculating the exact inverse A^{-1}.

Many preconditioning techniques have been successfully used in the context of reservoir simulation, some of them are general and the others are more tailored for use on the black-oil equations. Nested factorization [2] is one example of the former and is used in commercial reservoir simulation software. In the context of this thesis, we focused on three preconditioners, incomplete LU (ILU) factorization, algebraic multigrid (AMG) [36] and the constrained pressure residual (CPR) method [43], which is especially well suited for tackling the black-oil equations.

ILU factorization is a simple preconditioning technique, where an incomplete lower upper (LU) factorization of the system matrix is created on the basis of the matrix sparsity pattern. There exist multiple versions of ILU, the most basic being zero-level fill-in ILU, referred to as ILU0. In an ILU0 factorization of a matrix, the non-zero pattern of the incomplete lower and upper factors exactly matches the non-zero pattern of the original matrix. The combined non-zero pattern of a complete LU factorization would, in contrast, be a full matrix. More non-zero entries can be added to the ILU factors by adding one or more levels of fill-in. These factorizations are denoted as ILU(n). ILU0 is simple, but compared to more complicated alternatives, it is cheaper to construct and apply. Despite slower convergence, ILU preconditioned linear solvers often outperform AMG and CPR preconditioned solvers. In addition, ILU0 can be used as a smoother in both AMG and CPR preconditioners.

One major drawback of ILU factorization is that its construction and application are inherently sequential. There exist some methods for extracting parallelism, such as level scheduling [40] and graph colouring [22], but these approaches often yield limited potential for scalability. An alternative approach is to combine ILU with a block-Jacobi preconditioner. The block-Jacobi method is a variation of the Jacobi method, where the linear system is divided into multiple diagonal blocks. These blocks can then be inverted exactly or approximately with a preconditioning method such as ILU. Because the blocks are independent, the block-Jacobi method offers a high degree of parallelism. However, the convergence of a block-Jacobi preconditioned linear solver will often deteriorate with the number of blocks.

The AMG preconditioning technique works by taking the linear system and constructing a hierarchy of lower-resolution linear systems. The size of the coarsest linear system allows for the use of a direct solver. Interpolation and restriction operators are then used to transfer information between the levels of the hierarchy. Smoothing operators may be applied to reduce the error before moving up or down in the hierarchy. Examples of common smoothers are Jacobi and Gauss-Seidel, but other preconditioning operations, such as ILU, can be used. The AMG preconditioner has favourable convergence properties, but constructing the hierarchy and operators, as well as applying the preconditioner, are computationally costly.

AMG is particularly well suited to handle linear systems originating from elliptic PDEs. The black-oil equations are, however, only elliptic in pressure and hyperbolic in saturation. This is what CPR-type preconditioners try to exploit by decoupling the system to create a "pressure block". AMG can then be applied

to the elliptic part of the system, while a cheaper preconditioner, such as ILU factorization, is applied to the entire system. An illustration of the CPR method used in this thesis, represented as a two-level method, is presented in Figure 1.1.

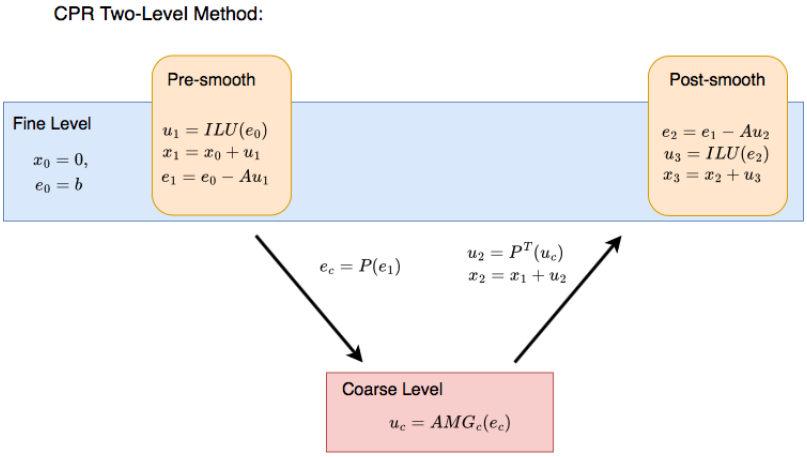


Figure 1.1: CPR preconditioner represented as a two-level method. The restriction and interpolation operators P and P^T moves information between the full system and "pressure block". ILU is used as pre- and post-smoother and AMG is applied to the coarse "pressure block" system.

1.1.3 Mesh partitioning

The first step of the parallelization of a reservoir simulator is to create a disjoint division of the reservoir grid cells. The partitioning of the mesh enables parallelization of system assembly and the linear solver, by assigning the disjoint subdivisions of the mesh to multiple processes. In general, the quality of the partitioning is evaluated against two performance measures: load balancing and minimization of communication overhead. Good load balancing means that the computational tasks of the simulations are fairly evenly divided between the parallel processes. The second partitioning goal is related to communication overhead and means minimizing inter-process communication and computations induced by, for example, overlap layers.

Heterogeneity in reservoir geology results in highly heterogeneous linear systems. Because of this heterogeneity, mesh partitioning can significantly impact the convergence properties of parallel preconditioners, so preconditioner performance should be considered when evaluating partitioning quality.

When dealing with unstructured grids, like the typical grid formats used for reservoir simulations, the standard partitioning approach is to apply graph partitioning on a graph representation of the unstructured grid. How to translate a grid to a graph depends on how the mesh entities relate to the unknowns in the discretized equations. For cell-centred finite volume discretization, the

graph is constructed by turning each cell in the grid into a graph vertex. An edge between two vertices i and j is added if the underlying cells are connected, either through a shared interface or a well.

Graph partitioning

A graph $G = (V, E)$ is composed of a set of vertices V and a set of edges $E \subset V \times V$ connecting pairs of vertices in V. If weights are assigned to each member of V and E through weighting functions $\sigma : V \to \mathbb{R}$ and $\omega : E \to \mathbb{R}$, then we get a weighted graph $G = (V, E, \sigma, \omega)$. The P-way graph partitioning problem involves dividing the set of vertices V into P approximately equal subsets $V_1, V_2, \ldots, V_P$ while minimizing the sum of the weights of the cut edges. Cut edges are defined as $e = (v_i, v_j) \in E$ where v_i and v_j belong to different subsets of V. Suppose the set of cut edges is referred to as $\mathcal{C}$. The graph partitioning problem can be formulated more precisely as a constrained optimization problem, where the objective function J is the sum of the weights of all members of $\mathcal{C}$, also called *edge-cut*:

$$\min \; J(\mathcal{C}) = \sum_{e \in \mathcal{C}} \omega(e), \tag{1.17}$$

$$\text{subject to} \quad \frac{\max_p \sum_{v \in V_p} \sigma(v)}{\frac{1}{P} \sum_{v \in V} \sigma(v)} < \epsilon, \tag{1.18}$$

where $\epsilon \geq 1$ is the imbalance tolerance of the load balancing constraint, which ensures that the size of the largest subgraph is close to the average subgraph size of the partition.

When the edge-weight function ω is uniform, that is, each edge has a unit weight, the edge-cut is a quantitative measure of connectivity between subgraphs of the partition. Because strongly connected subgraphs mean that more communication is required to perform parallel tasks on the data structures represented by the graph, the edge-cut with uniform edge-weights is related to the communication overhead.

When the edge-weight function ω is non-uniform, however, the objective function is no longer an approximation of communication overhead. As demonstrated in, for example, [13, 42], adopting non-uniform edge weights in the partitioning graph can be beneficial when partitioning linear systems with highly heterogeneous coefficients. Assigning edge weights based on the "between-cell coupling strength" can improve the quality of parallel preconditioners, such as block-Jacobi.

Because the graph partitioning problem in (1.17)-(1.18) is NP-complete, solving it exactly is practically impossible. Many existing algorithms find good approximate solutions, and implementations of these are available in several open software libraries such as Metis [23, 24], Scotch [31] and Zoltan [11].

1.1.4 OPM Flow

Flow [34] is an open-source reservoir simulator developed and maintained by the open porous media project (OPM). The *Flow* simulator solves standard black-oil problems, specialized enhanced oil recovery extensions and two-phase CO_2 storage models [38]. Made for use in both research and industry, *Flow* reads and writes industry standard ECLIPSE I/O files.

The corner-point grid format [32] is implemented in *Flow* using the DUNE grid interface. DUNE [5, 6] is a software framework for numerically solving PDEs with grid-based methods. In addition to the grid interface, *Flow* makes use of DUNE's dune-istl module [9] for linear algebra. The dune-istl module offers an AMG implementation [10] well suited for the non-symmetric and highly heterogeneous matrices arising in reservoir simulation models. *Flow* is parallelized using the Message Passing Interface (MPI) library, and graph partitioning is performed by Zoltan. Shared memory parallelism with OpenMP threading is available for matrix assembly and I/O, and efforts to offload the linear solver to hardware accelerators have been initiated [19], but not considered in this thesis.

1.2 Research questions

In this section, we present the research questions that have guided the work of this thesis. In addition to the research questions, the work of the thesis was focused on achieving the overall goal of improving the parallel performance of *Flow*.

RQ1: What is an appropriate domain decomposition strategy for parallel reservoir simulation?

This question is considered in Paper I, where we introduce a graph edge-weighing scheme that tries to balance preconditioner quality and communication volume minimization. Standard partitioning approaches, be it graph or hypergraph partitioning, aim at minimizing communication volume. Because of the heterogeneous linear systems that arise in reservoir simulation, the uniform edge-weight approach results in poor linear solver convergence. Transmissibility edge-weights are therefore often used to improve parallel preconditioner performance, at the cost of increased communication overhead.

In Paper III, we present a distributed well implementation that allows for splitting wells between subdomains. Wells connect remote parts of petroleum reservoirs, and not allowing the splitting of wells restricts the partitioning of the mesh. Experiments demonstrate that the partitioning quality improves when we allow distributed wells. We also observe improved overall performance in parallel simulations reservoir models with millions of grid cells.

Understanding the cost of the inter-process communication that arises when performing parallel SpMV, a key component of any iterative linear solver, is valuable when determining partitioning quality. The staircase model for point-to-point MPI communication introduced in Paper II makes it possible to predict data

transfer cost given a communication pattern and hardware-specific parameters. With this model, the partitioning quality can be more accurately assessed.

RQ2: How to ensure good computational efficiency in the underlying linear solver of a reservoir simulator?

For large-scale reservoir simulations, the computational bottleneck of the simulation is solving the system of linear equations in each Newton step. An efficient parallelization of the linear solver is therefore crucial for the overall parallel performance. Parallel linear solver efficiency is a topic in both Paper I and III. In Paper I, we demonstrate that parallelization with a layer of overlap results in non-contributing computations in the linear solver. Because a partitioning scheme that prioritizes heterogeneity awareness over communication overhead minimization is beneficial in reservoir simulations, the impact of the overlap-layer-related non-contributing computations on performance can not be ignored. With an appropriate ordering of the degrees of freedom, we show how to avoid non-contributing computations, and through numerical experiments, demonstrate improved parallel performance.

A good preconditioner is important for ensuring good Krylov-subspace iterative solver convergence when solving linear systems arising from the black-oil equations or other reservoir simulation models. In Paper I, we only use the block-Jacobi preconditioner with ILU0 in our parallel simulation. When simulating real-world reservoir models, the effectiveness of the block-Jacobi approach depends on the use of heterogeneity-aware partitioning. More advanced preconditioners are considered in Paper III. Here parallel simulator performance using ILU0, AMG and CPR preconditioned linear solvers are compared on a million-cell black-oil case and a nine-million cell CO_2 model case. We demonstrate that although AMG and CPR lead to improved convergence, their relative computational cost means that ILU0 may achieve better or comparable performance, at least for the black-oil model.

RQ3: How to achieve a resource-efficient use of modern computing platforms for parallel reservoir simulation?

The work on mesh partitioning in Paper I and III relates to this research question, because reducing communication volume is crucial to achieving efficient reservoir simulations. Combining heterogeneity awareness with communication volume minimization through logarithmic edge-weights, and simplifying the partitioning by allowing distributed wells, both contribute to making parallel reservoir simulations run more efficiently on modern CPU architectures.

Resource efficient simulations on modern multicore CPUs motivate the introduction of the staircase model for point-to-point MPI communication in Paper II. On computing clusters consisting of multicore CPUs, there are multiple layers of connectivity between the CPU cores, both intra- and inter-node. Within a node, the data transfer bandwidth is lower for inter-socket traffic compared

with intra-socket. Traffic speed between cores on different nodes can depend on network setup and even core placement relative to the network card. In addition to the different connective layers, competition between MPI processes for the same hardware resources results in contention that also impacts the overall bandwidth. In Paper II we present a strategy for predicting the cost of many-pair point-to-point communication that accounts for both connection-heterogeneity and contention.

1.3 Summary of Papers

This section contains a summary of the three research papers that make up this thesis. In addition to the three papers that make up the thesis, two other publications relevant to the thesis are added as appendices. These include the reference paper of the *Flow* reservoir simulator, and a benchmark study of the large CO_2 storage model also used in Paper III. They are listed below:

- Rasmussen AF, Sandve TH, Bao K, Lauser A, Hove J, Skaflestad B, Klöfkorn R, Blatt M, Rustad AB, Sævareid O, Lie KA, Thune A. The open porous media flow reservoir simulator. Computers & Mathematics with Applications. 2021 Jan 1;81:159-85.

- Sandve TH, Rustad AB, Thune A, Nazarian B, Gasda S, Rasmussen AF. Simulators for the Gigaton Storage Challenge. A Benchmark Study on the Regional Smeaheia Model. InEAGE GeoTech 2022 Sixth EAGE Workshop on CO2 Geological Storage 2022 Apr 4 (Vol. 2022, No. 1, pp. 1-5). EAGE Publications BV.

Despite authorship on both publications, the level of contribution does not warrant inclusion in the thesis.

Summary of Paper I

On the impact of heterogeneity-aware mesh partitioning and non-contributing computation removal on parallel reservoir simulations. The first paper details efforts to improve the efficiency of parallel reservoir simulation. Two topics are considered. First, we explain how to avoid non-contributing computations that arise when subdividing the reservoir mesh between multiple processes. Second, we look at how heterogeneity-aware mesh partitioning impact partitioning quality and overall simulation performance. We are especially interested in achieving a good balance between preconditioner efficiency and communication overhead minimization. Numerical experiments on real-world petroleum reservoir models demonstrate improvement in parallel simulation performance when using the *OPM Flow* reservoir simulator.

Motivation and outline

Parallel computing is needed to perform large-scale simulations of fluid flow in petroleum reservoirs. The heterogeneous geological properties, and the presence of faults and wells, introduces reservoir simulation-specific challenges to parallelization. In particular, the ill-conditioned linear systems that arise from the discretized black-oil equations become difficult to solve without heterogeneity-aware mesh partitioning. However, including heterogeneity-awareness in the partitioning comes at the cost of increased communication overhead.

Previous work on partitioning for reservoir simulation [14, 17, 26, 27, 46], have often focused on comparing different partitioning strategies and software libraries, with limited attention placed on reservoir-specific challenges, such as wells and geological heterogeneity. Heterogeneity-aware partitioning through edge-weighted graph partitioning is not, however, a novel idea in reservoir simulation. It is, for example, described in [47]. In [44], edge-weighted graph partitioning is used to improve linear solver convergence for computational fluid dynamics. Here the authors also consider the impact of edge-weights on communication overhead.

In this paper, we approached the issue of communication overhead in reservoir simulation from two directions. Firstly, by analysing the impact of non-contributing computations that arise when a layer of overlap is added to the subdomain boundaries. An implementation in the *OPM Flow* reservoir simulation that avoids these non-contributing computations is described, and experiments demonstrate improved parallel performance.

Secondly, we considered ways of balancing heterogeneity-awareness and minimization of communication overhead in the mesh partitioning. Edge-weighted graph partitioning is used to partition the reservoir grid. The heterogeneity-awareness is included through transmissibility edge-weights, which means that communication volume minimization, the aim of non-edge-weighted graph partitioning is lost. An edge-weighting scheme based on taking the logarithm of the transmissibility is presented, and represents a compromise between heterogeneity-awareness and minimization of communication overhead.

Efficient parallelization of reservoir simulation

A non-overlapping division of computational entities, such as grid cells, among multiple processes will induce a need for inter-process communication. Adding a layer of overlap of cells belonging to other processes on each subdomain boundary reduces the amount of needed inter-process communication. This added layer of *ghost cells* enables communication-free discretization and linear system assembly. In addition, the ghost layer allows for the reuse of sequential discretization code.

However, there are some disadvantages associated with the inclusion of a ghost layer. If implemented without care, the overlap layer may lead to wasteful computations. While the partitioner ensures load balancing in terms of process *interior cells*, the distribution of ghost cells can be unbalanced, which means that the non-contributing computations are extra detrimental

Table 1.1: Time measurements of SpMV and preconditioning operators for an increasing number of MPI processes, based on matrices from a 9-million cell 2-phase CO_2 model. Columns display results using either a natural ordering (NO) of DoFs or a ghost-last ordering (GL) of DoFs that avoid non-contributing computations.

	SpMV		ILU0		AMG		CPR	
N	NO	GL	NO	GL	NO	GL	NO	GL
16	0.064	0.057	0.060	0.056	0.29	0.26	0.44	0.40
32	0.042	0.036	0.040	0.036	0.21	0.18	0.31	0.27
64	0.036	0.029	0.035	0.030	0.19	0.17	0.30	0.25
128	0.018	0.013	0.019	0.015	0.11	0.090	0.16	0.12
256	0.0098	0.0074	0.011	0.0086	0.063	0.053	0.092	0.075
512	0.0050	0.0033	0.0064	0.0050	0.047	0.037	0.061	0.047

to parallel performance. We identify non-contributing operations in four linear algebra kernels: Vector addition, inner product, sparse matrix-vector multiplication (SpMV) and preconditioning operations. These operations are the main ingredients in every iterative Krylov-subspace linear solver, including GMRES and BiCGStab. The key to avoiding non-contributing computations in these operations is a clear separation of interior- and ghost-cell entries.

Through numerical experiments on the Norne and refined Norne models, we demonstrate how avoiding non-contributing computations yields improvements in the execution time of vector addition, inner product, SpMV and ILU0 forward/backward sweep. Ghost-cell-related computation will also impact the cost of other preconditioners, and we observe improvements in AMG and CPR apply time when avoiding ghost-cell-related computations. We present examples of these improvements in Table 1.1, where experiments on matrices obtained from a 9-million cell CO_2 model are presented. The table displays the execution time of SpMV and preconditioning apply time when using implementations that either avoid or do not avoid non-contributing computations. These experiments were executed on dual 64-core Kunpeng920-6428 CPUs.

Mesh partitioning

Mesh partitioning is the important first step in parallelizing a reservoir simulator. Because of the unstructured corner-point grid format, the standard partitioning strategy is to solve the optimization problem in (1.17)-(1.18), utilizing an edge-weighted graph representation of the mesh. The vertices of the graph represent grid cells, and an edge connects two vertices if the underlying cells have a non-empty intersection. Two reservoir-specific aspects complicate the creation of the partitioning graph: Well completions connecting distant regions of the reservoir and faults create barriers between physically connected cells. To account for faults, edges corresponding to fault intersections are removed from the partitioning graph. The standard well model, and other more involved well

models, create dependencies between the DoFs in all grid cells perforated by the same well. These dependencies are included in the partitioning graph by adding edges between two vertices if a well is completed in both underlying cells. We denote the set of well-related edges as E_w and the set of regular intersection edges as E_f, and define the weight of edge e as $\omega(e)$:

$$\omega(e) = \begin{cases} \infty & e \in E_w, \\ \theta & e = (v_i, v_j) \in E_f. \end{cases} \tag{1.19}$$

In Paper I, we used a version of *Flow* that did not allow for wells split between multiple subdomains. To avoid partitioning results with split wells, well edge-weights are given set to ∞, or in practice, a very large number. An alternative approach is considered in Paper III.

We consider three different edge-weighing schemes for the intersection edges in E_f: Uniform, transmissibility and logarithmic transmissibility. With uniform weights, the intersection edges are all given a weight equal to 1, which means that the objective function in the graph partitioning optimization problem represents the edge-cut. Logarithmic and transmissibility weights are both based on the transmissibility defined in (1.8). The transmissibility edge-weights can have a very large range of values.

$$\theta = \begin{cases} 1 & \text{Uniform} \\ \log\left(\frac{T_{ij}}{T_{\min}}\right) & \text{Logarithmic} \\ T_{ij} & \text{Transmissibility} \end{cases} \tag{1.20}$$

As expected, the three edge-weighing schemes yield partitioning results that differ in shape and quality. In Figure 1.2, 8-way partitioning, with the weights in (1.20), of the Norne reservoir model grid is presented. Here we see that partitions attained with uniform weights have more straight and regular boundaries, then what is observable for logarithmic and particularly transmissibility weights.

Numerical experiments

To demonstrate the effect of avoiding non-contributing computations and choice of edge-weighing scheme on parallel reservoir simulation, we conducted numerical experiments using *OPM Flow* on the open Norne case and a refined version of the Norne case. Parallel performance comparisons between *OPM Flow* and the commercial *Eclipse* and *Intersect* reservoir simulators, favouring *OPM Flow*, are also included in the paper.

When comparing the different edge-weighing schemes, we consider three metrics: Communication volume, linear solver convergence and, finally, overall simulation execution time. We present communication volume and linear solver convergence results for the refined Norne black-oil model in Figure 1.3. The communication volume plot in Figure 1.3a was created by dividing the refined Norne mesh into 2 to 240 subdomains, using graph partitioning with the

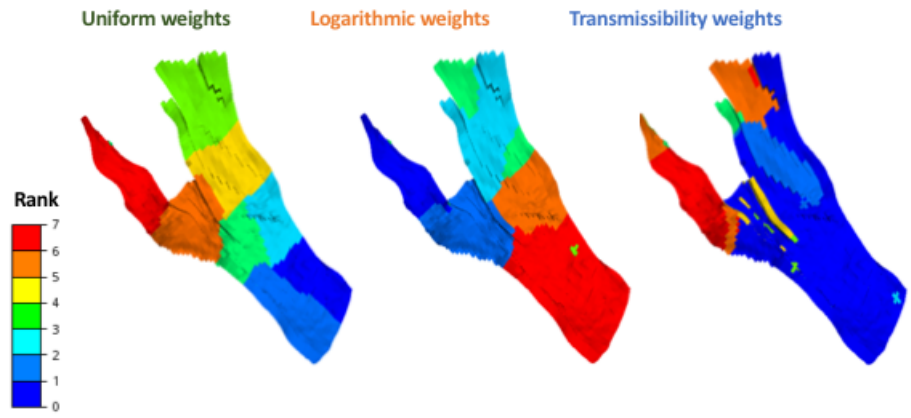
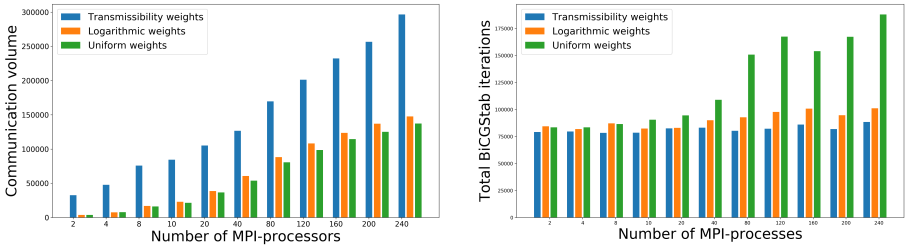


Figure 1.2: Resulting subdomains when performing 8-way partitioning on the Norne mesh using uniform, logarithmic and transmissibility edge weights. Each color represents one subdomain.

three edge-weight options in (1.20). Figure 1.3b displays the total number of block-Jacobi/ILU0 preconditioned BiCGStab iterations required to complete the simulation of the refined Norne when using 1 to 240 processes and partitioning with the three edge-weighing schemes.



(a) (b)

Figure 1.3: The plot in (a) displays communication volume when partitioning the refined Norne mesh for an increasing number of subdomains. In (b), the total number of BiCGStab iterations required to complete the refined Norne case is plotted against the number of processes.

The plots in Figure 1.3 illustrate the properties of the edge-weighing strategies. Transmissibility edge-weights yield significantly higher communication volume than uniform and logarithmic weights, but the simulation iteration count remains almost constant with increasing number of processes. In contrast, uniform weights achieve the lowest communication volume while resulting in a larger number of linear iterations for increasing number of processes. For the refined Norne case, using the logarithmic edge-weights results in a communication volume only

slightly higher than uniform weights and, similarly, an iteration count marginally higher than the transmissibility edge-weights.

The results in Figure1.3, that appear favourable to logarithmic edge-weights, are reflected in Figure 1.4, where total *Flow* simulation time is displayed for 2 to 240 processes. The simulations in Figure 1.4 where conducted on the Saga cluster, made up of dual-socket Intel Xeon-Gold 6138 2.0GHz 20-core CPUs. In Figure 1.4, we observe that simulations using logarithmic edge-weights in the partitioning complete faster than the alternatives for all processes except 2 and 8. We also notice that the relative difference in favour of logarithmic edge-weighing increases with the number of processes.

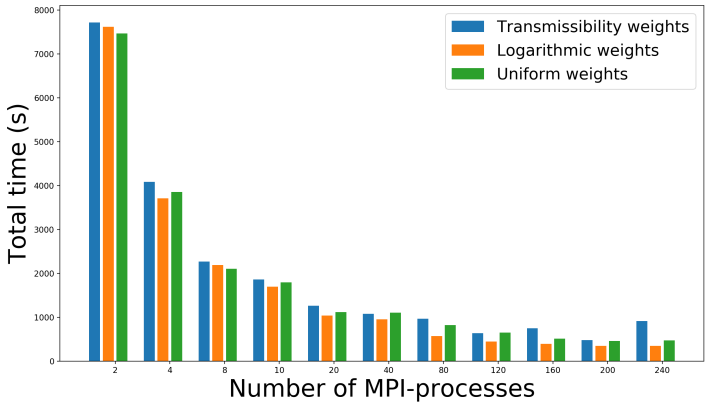


Figure 1.4: Execution time of simulations of The refined Norne case. X-axis shows the number of MPI processes, and the color of the bar represents the three edge-weighing schemes.

Conclusions

In Paper I, we considered strategies for more efficient parallelization of reservoir simulations. An improved implementation for avoiding non-contributing computations in the linear solver, based on a reordering of subdomain grid cells, was proposed and tested. We also studied the impact of different graph partitioning edge-weighting schemes on partition quality, preconditioner effectiveness and overall simulator performance. A new edge-weighing scheme was introduced with the aim of maintain a balance between communication volume minimization and heterogeneity-awareness.

Summary of Paper II

Detailed modeling of heterogeneous and contention-constrained point-to-point MPI communication.

In Paper I, we studied the impact of heterogeneity-aware mesh partitioning on communication overhead and overall reservoir simulation performance. The

second paper presents a new approach to estimating many-pair point-to-point MPI communication cost on multicore CPU clusters, and the new model can for example be used to better quantify the partition quality of a partitioned reservoir mesh. We target modern CPU architectures where inter-processes connectivity has at least two layers, and our model also incorporates contention arising from multiple processes competing for the same hardware resources. Part of the methodology is that architect-specific bandwidth and latency parameters are attained using simple ping-pong-type micro-benchmarks.

Motivation

The max-rate model represents the start-of-the-art quantitative model for estimating point-to-point MPI communication. Max-rate [16] is based on the simple postal model that uses start-up latency and bandwidth to estimate data transfer cost between two processes. In contrast to the postal model [4, 18], the max-rate model allows and is intended for multiple pairs of message-sending processes. The max-rate model also includes a bandwidth estimate that is linearly proportional to the number of message-sending processes, with an upper achievable data transfer bandwidth, or *max-rate*. Further extensions of the max-rate model were detailed in [7, 8]. This latest version of the max-rate model considers variations in message size and number of messages per process. The cost estimate for sending s_{total} amount of data with largest per-process delivery size being s_{max} is given as:

$$T = M \cdot \tau + \max\left(\frac{s_{\text{total}}}{BW_{\text{max}}}, \frac{s_{\text{max}}}{BW_{\text{SP}}}\right). \tag{1.21}$$

Here, M is the maximum number of messages sent and τ is the latency parameter. BW_{max} and BW_{SP} is respectively maximum bandwidth and single process bandwidth.

Although the max-rate model in (1.21) is simple to use, we identify several weaknesses. First, actual bandwidth may not be linearly proportional to active message-sending processes. Second, bandwidth contention when processes have large variations in the number and size of messages is not considered in the bandwidth modeling. Third, the max-rate model does not include a per-process estimate, only a total message transfer cost.

Measuring bandwidth

We want accurate measurements of intra- and inter-node data transfer bandwidth and latency on different hardware, when using multiple process pairs to send and receive MPI messages. To obtain these measurements, we use a micro-benchmark, which is a modified version of the `osu_bibw` benchmark of MVAPICH [28]. In this benchmark, messages of increasing sizes are bi-directionally sent between P pairs of processes. This allows us to measure bandwidth and message latency for

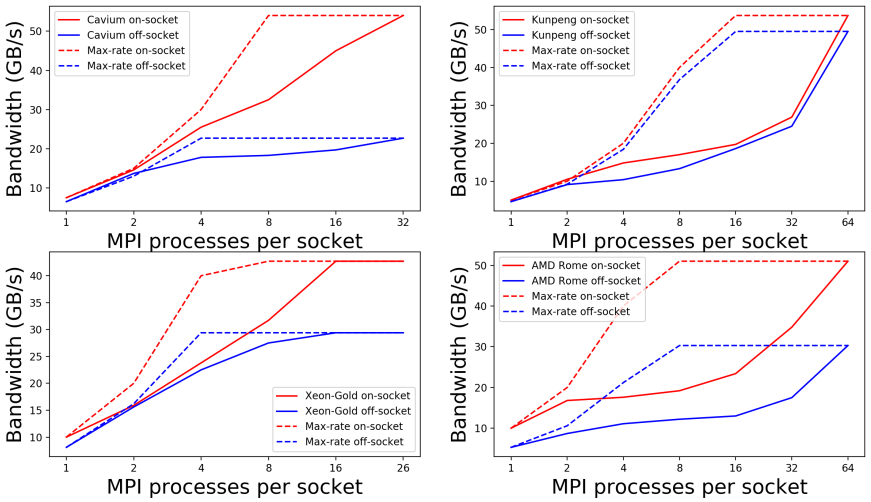


Figure 1.5: Comparison between measurement-pinpointed bandwidth values (solid curves) and linearly proportional bandwidth estimates of the max-rate model (dashed curves). The measurements are obtained on four dual-socket CPU machines: ARM Cavium ThunderX2 (top-left), ARM Kunpeng (top-right), Intel Xeon-Gold (bottom-left) and AMD Epyc Rome (bottom-right).

different numbers of processes and MPI message protocols. Intra-node bandwidth measurements obtained on four CPU architectures are presented in Figure 1.5.

In addition to bandwidth measurements, the plots in Figure 1.5 also contain max-rate bandwidth estimates, which are linearly proportional to the number of processes actively sending and receiving MPI messages, and calculated using single-process and maximum bandwidth. We observe that actual measurements for both intra- and inter-socket are lower than the max-rate estimate.

Staircase model strategy

As seen in Figure 1.5, bandwidth changes with the number of active message-sending processes. We also observe significant differences in intra- and inter-socket bandwidth on all architectures. In general, many-pair point-to-point MPI communication processes send and receive varying amounts of data to and from different connectivity levels, and processes will therefore complete at different times. Our staircase modeling strategy tries to describe the dynamically changing scenarios of competition and effective bandwidth. The strategy is expressed through the following principles:

1. Per-process completion time is determined by the maximum of time needed to receive its outgoing or send its incoming messages.

2. Order of incoming messages being completely received is decided by message size, the smallest message completing first and so on. Completion estimates of individual messages therefore follow a *staircase* strategy.

3. A second staircase principle is used to estimate per-process receive completion time when multiple processes compete for the same network connection. The process with the least amount of incoming communication completes first, allowing the remaining processes to continue receiving their communication. This repeats until every process has fully received all of its incoming messages. Bandwidth is shared equally among active processes.

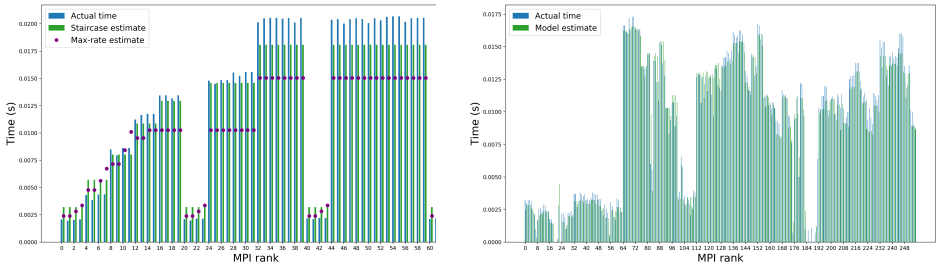
Based on the above principles, we formulate three models with increasing levels of generality. We first consider the simple case where each MPI process has a single outgoing and incoming message, but message sizes are allowed to vary between processes. In addition, all messages are communicated on the same level, either intra-socket, inter-socket or inter-node. The second model, building on the first, considers the scenario where the process sends and receives multiple messages. The third model is aimed at the general intra-node scenario, where messages are of both intra- and inter-socket type. In the mixed message type scenario, the per-process communication bandwidth estimate is calculated using the ratio of intra- and inter-socket traffic to total intra-node traffic. Because this ratio may vary between processes, the order in which each process completes receiving its messages is determined step-wise.

For the most general point-to-point case, where there is a mix of intra- and inter-node communication, we adopt the same approach as taken in [8], where the total cost estimate is given as the sum of separate intra-node and inter-node estimates.

Experiments

Several experiments conducted on four CPU architectures demonstrate the accuracy of the staircase model. We also compare max-rate and staircase estimates for point-to-point communication with messages on a single level. One max-rate/staircase comparison is presented in the plot in Figure 1.6a. The plot displays results from an intra-socket communication experiment obtained on AMD 64-core Epyc Rome-7742 CPUs. Blue bars represent per-process execution time, while green bars and purple dots show per-process staircase and max-rate estimates. The results of this example indicate that the staircase model produces better estimates than the max-rate model, which underestimates the communication cost.

In Figure 1.6b, results from an experiment with messages being sent intra- and inter-node, conducted on ARM 64-core Kunpeng 920-6426 CPUs, is presented in a bar plot. The communication pattern used in this experiment originates from a large reservoir simulation model. Again actual measurements and staircase estimates are represented as blue and green bars, and we observe good agreement between measurements and model.



(a) (b)

Figure 1.6: Measured time (blue bars) and staircase model estimate (green bars) for two experiments conducted on AMD Epyc Rome (a) and ARM Kunpeng (b) nodes. The plot in (a) also a per-process show max-rate estimate.

The experiments in Figure 1.6 were conducted on one and two CPU nodes, respectively. We also ran larger experiments on up to 64 Epyc Rome nodes and 8192 active processes. Here the staircase model achieved a prediction accuracy of 83.8%.

Conclusion

Max-rate bandwidth estimates are inaccurate for modeling point-to-point intra-node communication. Using intra- and inter-socket bandwidth measurements obtained with a ping-pong benchmark, the staircase modeling strategy proposed in this paper shows good agreement with experiments conducted on four different CPU architectures. Our staircase estimate incorporates the dynamically changing situation of point-to-point MPI communication, and is therefore able to more accurately model bandwidth and predict data transfer cost.

Summary of Paper III

Distributed wells for enhancing the efficiency of parallel reservoir simulation.

In Paper III, we return to challenges in reservoir simulation related to mesh partitioning and domain decomposition. The edge-weighting schemes in Paper III used ∞-weights on well edges representing dependencies between cells perforated by a common well. This was done because the implementation did not allow for split wells. In Paper III, a distributed well implementation that enables and allows split wells is presented. Without the constraint that wells must be contained on a single partition, we can remove the well edges from the partitioning graph, and with, that improve partitioning quality. Numerical experiments on a million-cell black-oil model and 9-million-cell CO_2 storage model [37] demonstrate a positive impact on overall simulation when allowing distributed wells. The

paper also includes comparisons of different preconditioning strategies for parallel simulations on the two mentioned models.

Background

Well models are an integral part of simulations of subsurface fluid flow in petroleum reservoirs. The well models describe flow in and out of the reservoir and create dependencies between cells perforated by the same well. These dependencies, which can connect remote parts of the reservoir, represent a challenge to the parallelization of reservoir simulations, because they complicate the mesh partitioning and domain decomposition strategies. One approach is to force wells to remain on a single partition. This simplifies implementation and avoids well-related communication overhead. Not allowing wells distributed among multiple processes may limit potential scaling and cause poor partitioning quality. Paper III presents a distributed implementation of the standard well model and an investigation of its impact on communication overhead, partitioning quality and overall simulation performance. We consider two reservoir models in our experiments, one million-cell black-oil model and one 9-million-cell CO_2 model.

Distributed wells

The implementation of distributed wells in *Flow* is outlined in Paper III. Our distributed-well parallelization is non-overlapping. Each process only retains information on well-cells owned by the process. This means that communication is needed when assembling and solving the well equations. The well model is included in the reservoir model through the Schur complement shown in (1.16). For each well i we need to assemble matrices B_i, C_i and D_i^{-1} to compute:

$$C_i D_i^{-1} B_i x \quad \text{and} \quad C_i D_i^{-1} f_i. \tag{1.22}$$

For a black-oil model with N cells, the B, C and D matrices have dimension $3n \times 4$, $4 \times 3N$ and 4×4. The B and C matrices are very sparse, with non-zero entries only in rows and columns associated with cells perforated by the well they are related to. In *Flow*'s parallel distributed-well implementation, each process will only store and assemble entries of B_i and C_i that correspond to perforated cells owned by the process. The D_i matrices will, however, be constructed on all processes that possess at least one cell perforated by well i. B_i and C_i can be assembled without communication using well and reservoir properties stored locally on each process. The calculation of the D_i matrix requires global properties stored on every process that owns a cell perforated by well i. One complicating factor is that some of the calculations required in the well setup are sensitive to the perforation order and must be performed sequentially in a serial fashion.

With the well operators B_i, C_i and D_i assembled and stored locally, applying them in parallel is uncomplicated. The C and D operators can be applied to

local vectors without communication, while B_i requires a collective reduction over all processes that contain cells perforated by well i.

With the ability to cut wells, well connections can be ignored in the mesh partitioning strategy. Specificity, the infinity edge-weights in (1.19) between vertices perforated by the same well, used in Paper I, can be removed or changed. In Paper III, we test the effect of removing well edge-weights on two models. In our experiments, we use transmissibility edge-weights in our partitioning of the mesh, and as explained and demonstrated in Paper I, these weights improve preconditioner quality at the cost of increased communication overhead. Because preconditioning is only applied to the flow equation part of the Schur complement, the well equations do not impact preconditioner quality. This motivates the removal of well edges in the partitioning graph.

The experiments in Paper III demonstrate improved partitioning quality when removing well edges in the partitioning graph. The total communication volume and the total number of domain neighbours are reduced when partitioning the million-cell black-oil reservoir model. The parallelization of the wells also introduces extra communication, but we show that this is negligible in comparison with the reduction in communication volume. In addition, removing well edge-weights improve well-cell load imbalance.

We also observe a positive impact on overall performance when removing well edges. This improvement is presented in Figure 1.7, where the simulation execution time of the million-cell black oil model with and without well edge-weights are compared.

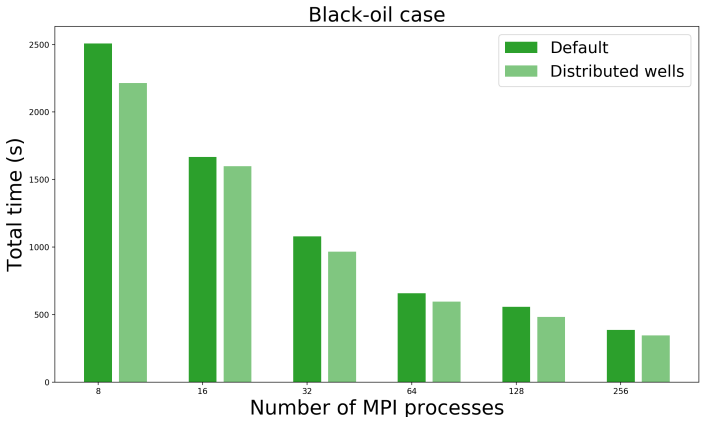


Figure 1.7: Million-cell black-oil model simulation execution time with (dark green) and without (light green) well edge-weights.

The improved simulation execution recorded in Figure 1.7 is around 5-10% for all number of processes. Although removing well edge-weights in the partitioning graph has a positive impact on partitioning quality, reduced communication overhead alone does not explain the results of Figure 1.7. Instead, the improved performance can be explained by a lower number of Newton iterations in

simulations where well edges are removed in the mesh partitioning. Based on the experiments, we argue that the reduction in Newton iterations is related to more favourable time-stepping. The adaptive time-stepping algorithm generates larger time steps for distributed well simulations. An experiment using hardcoded time steps in simulations with and without well edges demonstrates this. In this experiment, hardcoded time steps obtained from simulations without well edges in the partitioning are used in simulations where infinity edge-weights are included in the partitioning. The simulations with hardcoded time steps require fewer Newton iterations to complete than simulations using adaptive time steps.

Parallel preconditioner performance

Paper III includes a quantitative study of parallel preconditioner performance. We consider three preconditioning strategies, ILU0, AMG and CPR, and compare their performance on two reservoir models, one black-oil and one two-phase CO_2 storage model. An ILU0 pre- and post-smoother is used in both AMG and CPR on all hierarchy-levels. In addition to an ILU0 apply, the smoother operation performs one SpMV. Assuming that the execution time of ILU0 apply and SpMV are similar, we can create an estimate of AMG and CPR computational cost relative to ILU0.

We measure setup, update and application time of the three preconditioners and observe that the AMG and CPR setup costs give ILU0 a competitive advantage in overall performance. However, if AMG-hierarchy setup can be reused for several Newton iterations, the update costs of AMG and CPR are much less expensive operations. Measurements of preconditioner application time reveal that ILU0 is around 7.5-8.5 and 9.7-10.7 times faster than AMG and CPR for the black-oil and CO_2 models, respectively.

Simulations with AMG or CPR preconditioned solvers can only outperform ILU0 simulations if they can complete the simulation with significantly fewer linear iterations. This is illustrated in Figure 1.8, where execution times and linear iteration counts for parallel simulations of the million-cell black-oil case are displayed when using the three different preconditioners. In Figure 1.8 we observe that CPR outperforms ILU0 and AMG when using less than 512 processes, and that the reason is a much lower iteration count. AMG, despite completing the simulation with fewer iterations than ILU0, have lower overall performance than ILU0 for all number of processes. The ILU0 simulations have better scaling than the alternatives but only outperform CPR when using 512 processes.

Conclusions

Paper III deals with how well models impact mesh partitioning for reservoir simulation. The paper pairs well with Paper I, where the other reservoir-specific partitioning aspect, extreme heterogeneity, was considered. A distributed-well implementation was presented, and its positive impact on partitioning quality and overall simulation performance was tested and documented.

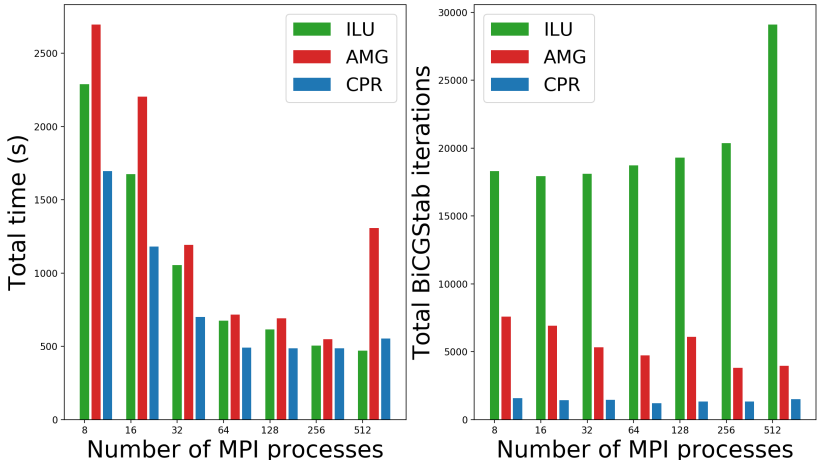


Figure 1.8: Million-cell black-oil model simulation execution time and total iteration count with ILU0, AMG and CPR preconditioning.

The second aspect of Paper III is a study on how the preconditioning strategy impacts parallel reservoir simulation. We considered three options, ILU0, AMG and CPR, and performed numerical experiments on one black-oil and one CO_2 storage model. The study includes an estimate of the relative computational cost of the tree preconditioners. We observed that the optimal choice of preconditioning strategy differs between the two models. CPR preconditioning yields the optimal performance on the black-oil model, while AMG outperforms the contenders on the CO_2 storage model.

1.4 Limitations

This section considers some limitations common or specific to the papers in the thesis. The lack of open large-scale real-world reservoir simulation models means that the experiments and results often relied on the medium-sized Norne model [29, 35]. The larger models that were used in Paper III are not open, creating challenges for reproducibility.

The MPI-all-the-way approach to parallelism was the only paradigm considered in the thesis. Threading and mixed MPI and threading were not considered except of some experiments using the OpenMP threading option in *Flow* for the linear system assembly. Hardware accelerators were also not considered in the thesis, only multicore CPUs.

The work of the thesis focused on mesh partitioning and aimed at reducing time spent on solving the linear systems that arise when performing reservoir simulations. Computational cost associated with linear system assembly was only partially considered in Paper III, despite having a considerable impact on the overall simulation performance.

The distributed-well implementation and experiments conducted in Paper III are only valid for the simple standard well model. Parallelizations of more complicated multi-segment well models are not implemented. In addition, the current distributed-well implementation only works when the flow and well parts of the Schur complement in (1.16) are handled separately.

1.5 Discussion and conclusions

This section concludes the introduction by returning to the research questions presented in Section 1.2, and considers to what extent the content of the thesis answered them. We also give a summary of the main contributions of the thesis and outline possible future work.

Heterogeneity-aware and communication overhead minimizing graph partitioning

Much of the work of this thesis concerns mesh partitioning and its impact on parallel reservoir performance. Both Paper I and Paper III contain results that demonstrate the importance of the initial partitioning step to the overall simulation performance. Mesh partitioning relates to all three research questions, particularly the first, regarding appropriate domain decomposition strategies for reservoir simulations. We summarize the thesis' contributions to mesh partitioning in the list below:

- A study of the impact of heterogeneity-aware graph partitioning on partitioning quality, linear solver convergence, communication overhead and overall simulator performance.
- A new logarithmic-transmissibility edge-weight scheme that aims to make a compromise between heterogeneity-awareness and communication overhead minimization.
- An investigation of how a new distributed well implementation impacts partitioning quality and overall performance.

The simulations conducted in Paper I, using different edge-weighing schemes, show that using pure transmissibility edge-weights is not the optimal strategy. For example, in the plot in Figure 1.4, logarithmic transmissibility edge-weights result in lower execution time for all numbers of processes when simulating the refined Norne model. Despite yielding better convergence, transmissibility edge-weights result in simulations that often complete even after the simulations using uniform edge-weights. In the experiments in Paper I, the simulations using logarithmic edge-weights achieved the best performance when using more than 8 MPI processes. However, we do not conclude that the logarithmic transmissibility edge-weighing scheme is the optimal strategy, or that it will always yield partitioning results that outperform pure transmissibility edge-weighting. Instead, we argue

that both heterogeneity-awareness and communication overhead minimization should be considered in mesh partitioning for reservoir simulation.

Partitioning quality impacts linear solver performance and therefore relate to the second research question. Inter-process communication is needed to perform the SpMV operation and some preconditioners in parallel, and poor partitioning quality results in higher communication overhead in the linear solver. Even though transmissibility edge-weight achieves faster convergence than the alternatives, the larger communication volume results in lower parallel efficiency in the linear solver and therefore lower overall performance.

In Paper III, we consider how well models should be handled in parallel reservoir simulations. A parallel implementation of the standard well model that allows wells to be split among multiple domains is presented and compared with a simple approach, where wells are only allowed to be present on a single subdomain. We observe that allowing distributed wells improves the partitioning quality, meaning lower communication volume. For the reservoir models that we consider in Paper III, the overall performance also benefits from the distributed well implementation.

Efficient parallelization of reservoir simulation

The second research question is also related to the second topic of Paper I, detailing how to avoid non-contributing computations related to ghost-cells. Ghost cells are included in the partitions of the reservoir mesh to achieve communication-free linear system assembly but can introduce extra overhead if incorrectly treated. In Paper I, we describe how an implementation can avoid non-contributing computations by exploiting a "ghost last" ordering of local cells. Experiments demonstrate that avoiding the non-contributing computations enhances the parallel efficiency of the linear solver and the overall performance of the simulator.

In Paper I, we only considered the ILU0 preconditioning operation, but the results in Table 1.1 prove that avoiding non-contributing computations also improves the performance of the more advanced AMG and CPR preconditioning strategies. The observed improvements in AMG and CPR are not unexpected, because the main computational component of applying these preconditioners is the smoother, which is illustrated for CPR in Figure 1.1.

A hybrid parallelism approach, combining threading and MPI, is an alternative that could further limit the impact of ghost-cell-related non-contributing computations. Introducing thread parallelism in the linear solver in *Flow* is a possible direction for future work. Although hybrid parallelism has already been implemented in other reservoir simulators [1], topics related to appropriate hybrid parallelization of ILU0 and CPR preconditioning could be further studied.

Modeling point-to-point MPI communication

Research question number three, concerning resource-efficient reservoir simulations on modern computing platforms, is approached indirectly in Paper II. The topic of Paper II is modeling of point-to-point MPI communication on multicore CPUs. The staircase model introduced in Paper II enables a more accurate and architecture-specific assessment of partitioning quality. Because of the challenges associated with reservoir simulation and mesh partitioning, the staircase model can assist in explaining simulation performance on modern multicore CPU computing platforms.

The staircase model improves the existing models of point-to-point communication in several ways. First, tabulated bandwidth values are used instead of less accurate max-rate estimates. Second, contention between MPI processes sending and receiving varying numbers of messages and sizes over the same interconnect level is modeled using a staircase principle. Third, an approach to modeling the overall bandwidth when messages are sent intra- and inter-socket is introduced. Experiments in Paper II demonstrate that the staircase modeling strategy produces better estimates than the previous models, especially for intra-node messages.

Another benefit of the staircase model is that it is capable of giving per-process estimates of the communication cost. A possible next step could therefore be to incorporate the staircase model in mesh partition strategies, either in the partitioning algorithm, for example, by using it in the objective function of the graph partitioning minimization problem, or to find an optimal mapping of processes post partitioning.

References

[1] AlOnazi, A. et al. “Performance Assessment of Hybrid Parallelism for Large-Scale Reservoir Simulation on Multi-and Many-core Architectures”. In: *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE. 2018, pp. 1–7.

[2] Appleyard, J. R. and Cheshire, I. M. “Nested factorization”. In: *SPE Symposium on Reservoir Simulation. 7*. 1983, pp. 315–324.

[3] Aziz, K. and Settari, A. *Petroleum Reservoir Simulation*. London: Applied Science Publishers, 1979.

[4] Bar-Noy, A. and Kipnis, S. “Designing broadcasting algorithms in the postal model for message-passing systems”. In: *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*. 1992, pp. 13–22.

[5] Bastian, P. et al. “A generic grid interface for parallel and adaptive scientific computing. Part I: abstract framework”. In: *Computing* vol. 82, no. 2 (2008), pp. 103–119.

[6] Bastian, P. et al. “A generic grid interface for parallel and adaptive scientific computing. Part II: implementation and tests in DUNE”. In: *Computing* vol. 82, no. 2-3 (2008), pp. 121–138.

[7] Bienz, A., D. Gropp, W., and N. Olson, L. “Improving Performance Models for Irregular Point-to-Point Communication”. In: *Proceedings of the 25th European MPI Users’ Group Meeting*. 2018, pp. 1–8.

[8] Bienz, A., Gropp, W. D., and Olson, L. N. “Reducing communication in algebraic multigrid with multi-step node aware communication”. In: *The International Journal of High Performance Computing Applications* vol. 34, no. 5 (2020), pp. 547–561.

[9] Blatt, M. and Bastian, P. “The Iterative Solver Template Library”. In: *Applied Parallel Computing. State of the Art in Scientific Computing*. Ed. by Kågström, B. et al. Vol. 4699. Lecture Notes in Computer Science. Springer, 2007, pp. 666–675.

[10] Blatt, M., Ippisch, O., and Bastian, P. “A massively parallel algebraic multigrid preconditioner based on aggregation for elliptic problems with heterogeneous coefficients”. In: *arXiv preprint arXiv:1209.0960* (2012).

[11] Boman, E. et al. “Zoltan 3.0: parallel partitioning, load-balancing, and data management services; user’s guide”. In: *Sandia National Laboratories, Albuquerque, NM* vol. 2, no. 3 (2007).

[12] Chen, Z., Huan, G., and Ma, Y. *Computational Methods for Multiphase Flows in Porous Media*. Vol. 2. Philadelphia: SIAM, 2006.

[13] Cullum, J. K., Johnson, K., and Tůma, M. “Effects of problem decomposition (partitioning) on the rate of convergence of parallel numerical algorithms”. In: *Numerical Linear Algebra with Applications* vol. 10, no. 5-6 (2003), pp. 445–465.

[14] Elmroth, E. “On grid partitioning for a high-performance groundwater simulation software”. In: *Simulation and Visualization on the Grid*. Berlin, Heidelberg: Springer, 2000, pp. 221–234.

[15] Fanchi, J. R. *Principles of applied reservoir simulation*. Elsevier, 2005.

[16] Gropp, W., Olson, L. N., and Samfass, P. “Modeling MPI communication performance on SMP nodes: Is it time to retire the ping pong test”. In: *Proceedings of the 23rd European MPI Users’ Group Meeting*. 2016, pp. 41–50.

[17] Guo, X., Wang, Y., and Killough, J. “The application of static load balancers in parallel compositional reservoir simulation on distributed memory system”. In: *Journal of Natural Gas Science and Engineering* vol. 28 (2016), pp. 447–460.

[18] Hockney, R. W. and Jesshope, C. R. *Parallel Computers Two: Architecture, Programming and Algorithms*. 2nd. IOP Publishing Ltd., 1988.

[19] Hogervorst, T. et al. “Hardware Acceleration of High-Performance Computational Flow Dynamics Using High-Bandwidth Memory-Enabled Field-Programmable Gate Arrays”. In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* vol. 15, no. 2 (2021), pp. 1–35.

[20] Holmes, J. A. “Enhancements to the strongly coupled, fully implicit well model: wellbore crossflow modeling and collective well control”. In: *SPE Reservoir Simulation Symposium*. OnePetro. 1983.

[21] IEA. *World Energy Outlook 2022*. 2022.

[22] Jones, M. T. and Plassmann, P. E. “A parallel graph coloring heuristic”. In: *SIAM Journal on Scientific Computing* vol. 14, no. 3 (1993), pp. 654–669.

[23] Karypis, G. and Kumar, V. “A fast and high quality multilevel scheme for partitioning irregular graphs”. In: *SIAM Journal on scientific Computing* vol. 20, no. 1 (1998), pp. 359–392.

[24] Karypis, G. and Kumar, V. “A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices”. In: *University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN* vol. 38 (1998).

[25] Lie, K.-A. *An Introduction to Reservoir Simulation using MATLAB/GNU Octave*. Cambridge: Cambridge University Press, 2019.

[26] Ma, Y. and Chen, Z. “Parallel computation for reservoir thermal simulation of multicomponent and multiphase fluid flow”. In: *Journal of Computational Physics* vol. 201, no. 1 (2004), pp. 224–237.

[27] Maliassov, S., Shuttleworth, R., et al. “Partitioners for parallelizing reservoir simulations”. In: *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers. 2009.

[28] *MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE*. https://mvapich.cse.ohio-state.edu/benchmarks/.

[29] *Norne reservoir simulation benchmark*. https://github.com/OPM/opm-data. 2019.

[30] Peaceman, D. W. “Interpretation of well-block pressures in numerical reservoir simulation (includes associated paper 6988)”. In: *Society of Petroleum Engineers Journal* vol. 18, no. 03 (1978), pp. 183–194.

[31] Pellegrini, F. and Roman, J. “Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs”. In: *International Conference on High-Performance Computing and Networking*. Springer. 1996, pp. 493–498.

[32] Ponting, D. K. “Corner point geometry in reservoir simulation”. In: *ECMOR I - 1st European Conference on the Mathematics of Oil Recovery*. 1989.

[33] Pörtner, H.-O. et al. *Climate change 2022: Impacts, adaptation and vulnerability*. IPCC Geneva, Switzerland: 2022.

[34] Rasmussen, A. F. et al. “The Open Porous Media Flow reservoir simulator”. In: *Computers & Mathematics with Applications* vol. 81 (2021), pp. 159–185.

[35] *Refined Norne reservoir simulation deck*. https://github.com/andrthu/opm-data/tree/refined-222-norne/refined-norne. 2020.

[36] Ruge, J. W. and Stüben, K. “Algebraic multigrid”. In: *Multigrid methods*. SIAM, 1987, pp. 73–130.

[37] Sandve, T. H. et al. “Simulators for the Gigaton Storage Challenge. A Benchmark Study on the Regional Smeaheia Model.” In: *EAGE GeoTech 2022 Sixth EAGE Workshop on CO2 Geological Storage*. Vol. 2022. European Association of Geoscientists & Engineers. 2022, pp. 1–5.

[38] Sandve, T. H., Rasmussen, A. F., and Rustad, A. B. “Open reservoir simulator for CO2 storage and CO2-EOR”. In: *14th Greenhouse Gas Control Technologies Conference*. 2018.

[39] Saad, Y. and Schultz, M. H. “GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems”. In: *SIAM Journal on Scientific and Statistical Computing* vol. 7, no. 3 (1986), pp. 856–869.

[40] Saad, Y. *Iterative Methods for Sparse Linear Systems*. Philadelphia: SIAM, 2003.

[41] van der Vorst, H. A. “Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems”. In: *SIAM Journal on scientific and Statistical Computing* vol. 13, no. 2 (1992), pp. 631–644.

[42] Vecharynski, E., Saad, Y., and Sosonkina, M. “Graph partitioning using matrix values for preconditioning symmetric positive definite systems”. In: *SIAM Journal on Scientific Computing* vol. 36, no. 1 (2014), A63–A87.

[43] Wallis, J. R. “Incomplete Gaussian elimination as a preconditioning for generalized conjugate gradient acceleration”. In: *SPE Reservoir Simulation Symposium*. OnePetro. 1983.

[44] Wang, M. et al. “Mesh partitioning using matrix value approximations for parallel computational fluid dynamics simulations”. In: *Advances in Mechanical Engineering* vol. 9, no. 11 (2017), p. 1687814017734109.

[45] Williamson, A. S. and Chappelear, J. E. “Representing wells in numerical reservoir simulation: Part 1–Theory”. In: *Society of Petroleum Engineers Journal* vol. 21, no. 03 (1981), pp. 323–338.

[46] Wu, Y.-S. et al. “An efficient parallel-computing method for modeling nonisothermal multiphase flow and multicomponent transport in porous and fractured media”. In: *Advances in Water Resources* vol. 25, no. 3 (2002), pp. 243–261.

[47] Zhong, H. “Development of a New Parallel Thermal Reservoir Simulator”. PhD thesis. University of Calgary, 2016.

Papers

Paper I

On the impact of heterogeneity-aware mesh partitioning and non-contributing computation removal on parallel reservoir simulations

I

Andreas Thune, Xing Cai, Alf Birger Rustad

Published in *Journal of Mathematics in Industry*, June 2021, volume 11, pp. 1-23.
DOI: 10.1186/s13362-021-00108-5.

Abstract

Parallel computations have become standard practice for simulating the complicated multi-phase flow in a petroleum reservoir. Increasingly sophisticated numerical techniques have been developed in this context. During the chase of algorithmic superiority, however, there is a risk of forgetting the ultimate goal, namely, to efficiently simulate real-world reservoirs on realistic parallel hardware platforms. In this paper, we quantitatively analyse the negative performance impact caused by non-contributing computations that are associated with the "ghost computational cells" per subdomain, which is an insufficiently studied subject in parallel reservoir simulation. We also show how these non-contributing computations can be avoided by reordering the computational cells of each subdomain, such that the ghost cells are grouped together. Moreover, we propose a new graph-edge weighting scheme that can improve the mesh partitioning quality, aiming at a balance between handling the heterogeneity of geological properties and restricting the communication overhead. To put the study in a realistic setting, we enhance the open-source *Flow* simulator from the OPM framework, and provide comparisons with industrial-standard simulators for real-world reservoir models.

I.1 Introduction and motivation

Computer simulation is extensively used in the oil industry to predict and analyse the flow of fluids in petroleum reservoirs. The multi-phased flow in such porous media is mathematically described by a complicated system of partial differential

equations (PDEs), only numerically solvable for realistic cases. At the same time, the quest for realism in reservoir simulation leads to using a large number of grid cells, thereby many degrees of freedom. Parallel computing is thus indispensable for achieving large scales and fast simulation time.

The fundamental step of parallelization is to divide the total number of degrees of freedom among multiple hardware processing units. Each processing unit is responsible for computing its assigned degrees of freedom, in collaboration with the other units. To use distributed-memory mainstream parallel computers for mesh-based computations, such as in a reservoir simulation, the division of the degrees of freedom is most naturally achieved by partitioning the global computational mesh. Each processing unit is therefore assigned with a sub-mesh consisting of two types of grid cells: *interior* and *ghost*. The distributed ownership of the interior cells gives a *disjoint* division of all the degrees of freedom among the processing units. The ghost cells per sub-mesh, which constitute one or several layers around the interior cells, are needed to maintain the PDE-induced coupling between the neighboring sub-meshes.

One major benefit of such a work division, based on mesh partitioning, is that each processing unit can *independently* discretize the PDEs restricted to its assigned sub-mesh. Any global linear or nonlinear system will thus only exist logically, collectively represented by a set of sub-systems of linear or nonlinear equations. The overall computing speed of a parallel simulator, however, hinges upon the quality of mesh partitioning. Apart from the usual objective of minimizing the inter-process communication volume, it is also desirable to avoid that strongly coupled grid cells are assigned to different processes. The latter is important for the effectiveness of parallel preconditioners that are essential for iteratively solving the linear systems arising from the discretized PDEs. The two objectives are not easy to achieve simultaneously.

It is therefore necessary to revisit the topic of mesh partitioning as the foundation of parallel reservoir simulations. In particular, the interplay between minimizing communication overhead and maximizing numerical effectiveness, especially in the presence of reservoir-characteristic features, deserves a thorough investigation. The novelty of this paper includes a study of how different edge-weighting schemes, which can be used in a graph-based method of mesh partitioning, will influence numerical effectiveness and communication overhead. We also quantify the negative performance impact caused by non-contributing computations that are associated with the ghost degrees of freedom. This subject is typically neglected by the practitioners of parallel reservoir simulations. Moreover, we present a simple strategy to avoid the non-contributing computations based on a reordering of the interior and ghost grid cells per subdomain.

The remainder of the paper is organized as follows. Section I.2 gives a very brief introduction to the mathematical model and the numerical solution strategy for reservoir flow simulations. Then, Section I.3 explains the parallelization with a focus on how to avoid non-contributing computations related to the unavoidable ghost grid cells. Thereafter, Section I.4 devotes its attention to the details of mesh partitioning and the corresponding graph partitioning problem, with

the presentation of a new edge-weighting scheme. The impacts of removing non-contributing computations and applying the new edge-weighting scheme are demonstrated by numerical experiments in Section I.5, whereas Sections I.6 and I.7, respectively, addresses the related work and provides concluding remarks.

I.2 Mathematical model and numerical strategy

In this section, we will give a very brief introduction to the most widely used mathematical model of petroleum reservoirs and a standard numerical solution strategy that is based on corner-point grids and cell-centered finite volume discretization.

I.2.1 The black-oil model

The standard mathematical model used in reservoir simulation is the black-oil model [1, 4]. It is a system of nonlinear PDEs governing three-phase fluid flow in porous media. The equations are derived from Darcy's law and conservation of mass. The model assumes that the different chemical species found in the reservoir can be separated into three categories of fluid phases $\alpha = \{w, o, g\}$: water (w), oil (o) and gas (g). There are consequently three main equations in the black-oil model, one for each phase:

$$\frac{\partial}{\partial t}\left[\frac{\phi S_o}{B_o}\right] = \nabla \cdot \left[\frac{k_{ro}K}{\mu_o B_o}\nabla\Phi_o\right] + q_o, \tag{I.1}$$

$$\frac{\partial}{\partial t}\left[\frac{\phi S_w}{B_w}\right] = \nabla \cdot \left[\frac{k_{rw}K}{\mu_w B_w}\nabla\Phi_w\right] + q_w, \tag{I.2}$$

$$\frac{\partial}{\partial t}\left[\phi\left(\frac{R_s S_o}{B_o} + \frac{S_g}{B_g}\right)\right] = \nabla \cdot \left[R_s\frac{k_{ro}K}{\mu_o B_o}\nabla\Phi_o + \frac{k_{rg}K}{\mu_g B_g}\nabla\Phi_g\right] + R_s q_o + q_{fg}. \tag{I.3}$$

Here, ϕ, K and R_s are porosity, permeability and gas solubility. They describe the geological properties of a reservoir. For each phase α, the terms S_α, μ_α, B_α and $k_{r\alpha}$ denote saturation, viscosity, formation volume factor and relative permeability. The phase potential Φ_α is defined by the phase pressure p_α and phase density ρ_α:

$$\Phi_\alpha = p_\alpha + \rho_\alpha \gamma z, \tag{I.4}$$

where γ and z are the gravitational constant and reservoir depth. The unknowns of the black-oil model are the saturation and pressure of each phase, so the following three relations are needed to complete Eqs. I.1-I.3:

$$S_o + S_w + S_g = 1, \tag{I.5}$$

$$p_w = p_o - p_{cow}(S_w), \tag{I.6}$$

$$p_g = p_o + p_{cog}(S_g). \tag{I.7}$$

The dependencies of the capillary pressures p_{cow} and p_{cog} upon the saturations S_w and S_g, used in Eq. I.6 and Eq. I.7, are typically based on empirical models.

I.2.2 Well modelling

The right-hand sides of the black-oil model (Eqs. I.1-I.3) contain source/sink terms q_α, which represent either production or injection wells in a reservoir. The wells affect the fluid flow on a much finer scale than what is captured by the resolution of the computational mesh for the reservoir. Special well models, such as the Peaceman model [19], are incorporated to model important phenomena, such as stark pressure drops, in proximity to well in- and outflow. In the Peaceman well model, the pressure drop is modelled by introducing new variables and equations in cells that contain a well bottom-hole. The related well equations numerically couple all the grid cells perforated by each well.

I.2.3 Corner-point grid and discretization

It is common to use the 3D corner-point grid format [21] to represent a reservoir mesh. A corner-point grid is a set of hexahedral cells logically aligned in a Cartesian fashion. The actual geometry of the grid is defined by a set of inclined vertical pillars, such that each grid cell in the mesh is initially formed by eight corner points on four of these pillars. Deformation and shifting of the sides of a cell are allowed independently of the horizontal neighboring cells. Moreover, a realistic reservoir may turn some of the cells to be inactive. The combined consequence is that the resulting computational mesh is unstructured. For example, a cell can have fewer than six sides, and there can be more than one neighboring cell on each side.

A standard cell-centred finite volume scheme, using two-point flux approximation with upwind mobility weighing [14], can be applied on a corner-point grid to discretize the PDEs of the black-oil model. The time integration is fully implicit to ensure numerical stability. Take for instance the water equation (Eq. I.2). Let $S_w^{\ell,i}$ denote S_w at time step ℓ inside cell C_i, which has V_i as its volume. Suppose the neighboring cells of C_i are denoted as $C_{j_0}, \ldots, C_{j_i}$, the discretization result of Eq. I.2 restricted to cell C_i is thus

$$V_i \frac{\phi^i}{\Delta t} \left(\left(\frac{S_w}{B_w} \right)^{\ell+1,i} - \left(\frac{S_w}{B_w} \right)^{\ell,i} \right) - \sum_{j=j_0}^{j_i} \lambda_w^{\ell+1,ij} T_{ij} (\Phi_w^{\ell+1,i} - \Phi_w^{\ell+1,j}) = V_i q_w^{\ell+1,i}. \tag{I.8}$$

Here, $\lambda_w^{\ell+1,ij} = \frac{k_{rw}^{\ell+1,ij}}{\mu_w^{\ell+1,ij} B_w^{\ell+1,ij}}$ denotes the water mobility on the face intersection Γ_{ij} between a pair of neighboring cells C_i and C_j, whereas T_{ij} is the *static transmissibility* on Γ_{ij}:

$$T_{ij} = m_{ij} |\Gamma_{ij}| \left(\frac{\|\vec{c}_i\|^2}{\vec{n}_i K_i \vec{c}_i} + \frac{\|\vec{c}_j\|^2}{\vec{n}_j K_j \vec{c}_j} \right)^{-1}. \tag{I.9}$$

The m_{ij} term denotes a transmissibility multiplier, for incorporating the effect of faults. For example, when a fault acts as a barrier between cells C_i and C_j, we have $m_{ij} = 0$. Fig. I.1 illustrates the geometric terms $\vec{c}_i$, $\vec{n}_i$ and Γ_{ij} involved in the transmissibility calculation. A typical scenario of reservoir simulation is

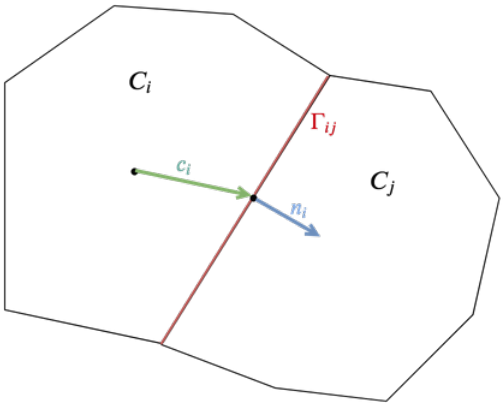


Figure I.1: A sketch of the geometric properties needed to calculate the static transmissibility (Eq. I.9) between two neighboring grid cells C_i and C_j.

that s_w, s_g and p_o are chosen as the primary unknowns, and the cell-centered finite volume method is applied to the three main equations Eqs. I.1-I.3 on all the grid cells. As result we get a system of nonlinear algebraic equations per time step. The total number of degrees of freedom is three times the number of active grid cells. Newton iterations are needed at each time level, such that a series of linear systems $Ax = b$ will be solved by an iterative method, such as BiCGStab or GMRES [25], which is accelerated by some preconditioner. The linear systems are sparse, often ill-conditioned, and non-symmetric due to the influence of well models. Although the nonzero values in the matrix A change with the time level and Newton iteration, the sparsity pattern remains unchanged (as long as the corner-point grid is fixed). This allows for a static partitioning of the computational mesh needed for parallelization. In this context, the static transmissibility T_{ij} defined in Eq. I.9 is an important measure of the coupling strength between a pair of neighboring cells C_i and C_j.

I.3 Efficient parallelization of reservoir simulation

To parallelize the numerical strategy outlined in the preceding section, several steps are needed. The main focus of this section will be on two topics. First, we explain why ghost grid cells need to be added per sub-mesh after the global computational mesh is non-overlappingly partitioned. Second, we pinpoint the various types of non-contributing computations that arise due to the ghost cells, and show how these can be avoided for a better computational efficiency. We remark that the exact amount and spread of ghost cells among the sub-meshes are determined by the details of the mesh-partitioning scheme, which will be the subject of Section I.4.

I.3.1 Parallelization based on division of cells

Numerical solution of the black-oil model consists of a time integration procedure, where during each time step several Newton iterations are invoked to linearize the nonlinear PDE system in Eqs. I.1-I.3. The linearized equations are then discretized and solved numerically. The main computational work inside every Newton iteration is the construction and subsequent solution of a linear system of the form $Ax = b$. Typically, the 3D corner-point grid remains unchanged throughout the entire simulation. It is thus customary to start the parallelization by a static, non-overlapping division of the grid cells evenly among a prescribed number of processes. Suppose N denotes the total number of active cells in the global corner-point grid, and N_p is the number of cells assigned to process p, then we have

$$\sum_{p=1}^{P} N_p = N,$$

where P denotes the total number of processes. Process p is responsible for computing the $3N_p$ degrees of freedom that live on its N_p designated cells. The global linear system $Ax = b$ that needs to be calculated and solved inside each Newton iteration will only exist *logically*, i.e., collectively composed by the $3N_p$ rows of A and the $3N_p$ entries of x and b that are owned by every process $p = 1, 2, \ldots, P$.

I.3.2 The need for ghost cells

The non-overlapping cell division gives a "clean-cut" distribution of the computational responsibility among the P processes, specifically, through a divided ownership of the x entries. There are however two practical problems for parallel computing associated with such a non-overlapping division.

First, we recall that the numerical coupling between two neighboring cells C_i and C_j is expressed by the static transmissibility T_{ij} as defined in Eq. I.9. In case C_i and C_j belong to two different processes, inter-process exchange of data is required in the parallel solution procedure. If each process has a local data structure storing only its designed $3N_p$ entries of x, the inter-process communication will be in the form of many individual 3-value exchanges, resulting in a drastic communication overhead. It is thus common practice to let each process extend its portion of the x vector by $3N_p^G$, where N_p^G denotes the number of *ghost* cells that are not owned by process p but border its internal boundary. For the finite volume method considered in this paper, only one layer of ghost cells is needed. Fig. I.2 demonstrates how ghost cells are added to the local grids of each process. The extended local data structure will allow aggregated inter-process communication, i.e., all the values needed by process p from process q are sent in one batch, at a much lower communication overhead compared with the individual-exchange counterpart. Specifically, whenever x has been distributedly updated, process p needs to receive in total $3N_p^G$ values of x from its neighbors. To distinguish between the two types of cells, we will from now

on denote the originally designated N_p cells from the non-overlapping division as *interior* cells on process p.

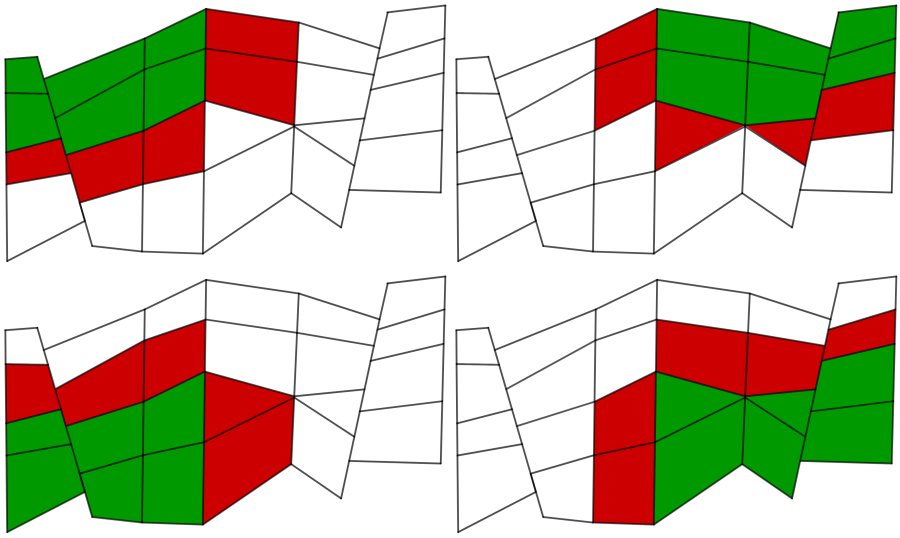


Figure I.2: An illustrative example of 4-way mesh partitioning. The interior cells of each of the four subdomains are colored green, while ghost cells are colored red.

Second, and perhaps more importantly, if a local discretization is carried out on process p by restricting to its designated N_p interior cells, the resulting local part of A, which is of dimension $3N_p \times 3N_p$, will be incomplete on the rows that correspond to the cells that have one or more of their neighboring cells owned by a process other than p. A similar problem also applies to the corresponding local entries in the vector b. Elaborate inter-process communication can be used to expand the sub-matrix on process p to be of dimension $3N_p \times 3(N_p + N_P^G)$, for fully accommodating the numerical coupling between process p and all its neighboring processes. However, a *communication-free* and thereby more efficient local discretization approach is to let each process also include its ghost cells. More specifically, the local discretization per process is independently done on a sub-mesh that comprises both the interior and ghost cells. This computation can reuse a sequential discretization code, without the need of writing a specialized subdomain discretization procedure. The resulting sub-matrix A_p will therefore be of dimension $3(N_p + N_p^G) \times 3(N_p + N_P^G)$ and the sub-vector b_p of length $3(N_p + N_p^G)$. We note that the $3N_p^G$ "extra" rows (or entries) in A_p (or b_p) that correspond to the N_p^G ghost cells will be incomplete/incorrect, but they do not actively participate in the parallel computation later. One particular benefit of having a square A_p is when a parallelized iterative solver for $Ax = b$ relies on a parallel preconditioner that adopts some form of incomplete factorization per process. The latter typically requires each local matrix A_p to be of a (logically)

square shape.

In the following, we will discuss what types of computation and memory overhead can arise due to the ghost cells and how to alleviate them.

I.3.3 Non-contributing computation and memory overhead due to ghost cells

While promoting communication-free discretizations per sub-mesh and aggregated inter-process exchanges of data, the ghost cells (and the associated ghost degrees of freedom) on every sub-mesh do bring disadvantages. If not treated appropriately, these can lead to wasteful computations that are discarded later, as well as memory usage overhead. Such issues normally receive little attention in parallel reservoir simulators. To fully identify these performance obstacles, we will now dive into some of the numerical and programming details related to solving $Ax = b$ in parallel.

For any Krylov-subspace iterative solver for $Ax = b$, such as BiCGStab and GMRES [25], the following four computational kernels must be parallelized:

- Vector addition: $w = u + v$. If all the involved vectors are distributed among the processes in the same way as for x and b, then no inter-process communication is needed for a parallel vector addition operation. Each process simply executes $w_p = u_p + v_p$ independently, involving the sub-vectors. However, unless the result vector w is used as the input vector to a subsequent matrix-vector multiplication (see below), the floating-point operations and memory traffic associated with the ghost-cell entries are wasted. It is indeed possible to test for each entry of w_p whether it is an interior-cell value or not, thus avoiding the non-contributing floating-point operations, but such an entry-wise `if-test` may dramatically slow down the overall execution of the parallel vector addition. Moreover, the memory traffic overhead due to the ghost-cell entries cannot be avoided on a cacheline based memory system, if the ghost-cell and interior-cell entries are "intermingled" in memory.

- Inner product: $u \cdot v$. Again, we assume that both sub-vectors u_p and v_p have $3(N_p+N_p^G)$ entries on process p. It is in fact numerically *incorrect* to let each process simply compute its local inner product $u_p \cdot v_p$, before summing up the local contributions from all the processes by a collective communication (such as the `MPI_Allreduce` function). The remedy is to let each process "skip" over the ghost-cell entries in u_p and v_p. In a typical scenario that the ghost-cell entries are mixed with interior-cell entries in u_p and v_p, some extra implementation effort is needed. For example, an assistant integer array named `mask` can be used, which is of length $N_p + N_p^G$, where `mask[i]==1` means cell i is interior and `mask[i]==0` means otherwise. Assume the three degrees of freedom per cell are stored contiguously in memory, the following code segment is a possible implementation of the parallel inner product:

```
double sub_dot_p = 0, global_dot_p;
```

```
for (int i=0; i<sub_num_cells; i++) {
  sub_dot_p += mask[i]*(sub_u[3*i]*sub_v[3*i]
                        +sub_u[3*i+1]*sub_v[3*i+1];
                        +sub_u[3*i+2]*sub_v[3*i+2]);
}
MPI_Allreduce (&sub_dot_p, &global_dot_p, 1, MPI_DOUBLE,
               MPI_SUM, MPI_COMM_WORLD);
```

For example, the well-known DUNE software framework [2] adopts a similar implementation. It is clear that the floating-point operations associated with the ghost-cell entries in u_p and v_p, as well as all the multiplications associated with the array `mask`, are non-contributing work. Allocating the array `mask` also incurs memory usage and traffic overhead.

- Sparse matrix-vector multiplication: $u = Av$. Here, we recall that the global matrix A is logically represented by a sub-matrix A_p per process, arising from a communication-free discretization that is restricted to a sub-mesh comprising both interior and ghost cells. The dimension of A_p is $3(N_p + N_p^G) \times 3(N_p + N_p^G)$. Moreover, we assume that all the ghost-cell entries in the sub-vector v_p are consistent with their "master copies" that are owned by other processes as interior-cell entries. This can be ensured by an aggregated inter-process data exchange. Then, a parallel matrix-vector multiplication can be easily realized by letting each process independently execute $u_p = A_p v_p$. We note that the ghost-cell entries in u_p will not be correctly computed (an aggregated inter-process data exchange is need if, e.g., u_p is later used as the input to another matrix-vector multiplication). Therefore, the floating-point operations and memory traffic associated with the ghost-cell entries in u_p and the ghost-cell rows in A_p are non-contributing.

- Preconditioning operation: $w = M^{-1}u$. For faster and more robust convergence of a Krylov-subspace iterative solver, it is customary to apply a preconditioning operation to the result vector of a preceding matrix-vector multiplication. That is, a mathematically equivalent but numerically more effective linear system $M^{-1}Ax = M^{-1}b$ is solved in reality. The action of a parallelized preconditioner M^{-1} is typically applying $w_p = \tilde{A}_p^{-1}u_p$ per sub-mesh, where $\tilde{A}_p^{-1}$ denotes an inexpensive numerical approximation of the inverse of A_p. One commonly used strategy for constructing $\tilde{A}_p^{-1}$ is to carry out an incomplete LU (ILU) factorization [25] of A_p. Similar to the case of parallel matrix-vector multiplication, the floating-point operations and memory traffic associated with the ghost-cell entries in w_p and the ghost-cell rows in A_p are non-contributing.

I.3.4 Handling non-contributing computation and memory overhead

The negative impact on the overall parallel performance, caused by the various types of non-contributing computation and memory usage/traffic overhead, can be large. This is especially true when the non-overlapping mesh partitioning is of insufficient quality (details will be discussed in Section I.4). It is thus desirable to eliminate, as much as possible, the non-contributing computation and memory overhead.

A closer look at the four kernels that are needed in the parallel solution of $Ax = b$ reveals the actual "evil". Namely, the interior-cell entries and ghost-cell entries are intermingled. This is a general situation if the interior and ghost cells of a sub-mesh are ordered to obey the original numbering sequence of the corresponding cells in the global 3D corner-point grid. (This is standard practice in parallel PDE solver software.) Hence, the key to avoiding non-contributing computation and memory overhead is a separation of the interior-cell entries from the ghost-cell counterparts in memory. This can be achieved per sub-mesh by deliberately numbering all the ghost cells after all the interior cells, which only needs to be done once and for all. If such a local numbering constraint is enforced, the non-contributing computation and memory overhead can be almost completely eliminated.

Specifically, the parallel vector addition and inner-product can now simply stop at the last interior degree of freedom. The array `mask` is thus no longer needed in the parallel inner-product operation. For the parallel matrix-vector multiplication, the per-process computation can stop at the last interior-cell row of A_p. In effect, the local computation only touches the upper $3N_p \times 3(N_p + N_p^G)$ segment of A_p. The last $3N_p^G$ rows of A_p are not used. This also offers an opportunity to save the memory storage related to these "non-contributing" rows. More specifically, each of the last $3N_p^G$ rows can be zeroed out and replaced with a single value of 1 on the main diagonal. As a result, the sub-matrix A_p on process p is of the following new form:

$$A_p = \begin{bmatrix} A_p^{II} & A_p^{IG} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \tag{I.10}$$

where the A_p^{II} block is of dimension $3N_p \times 3N_p$ and stores the numerical coupling among the $3N_p$ interior degrees of freedom, whereas the A_p^{IG} block is of dimension $3N_p \times 3N_p^G$ and stores the numerical coupling between the $3N_p$ interior degrees of freedom and the $3N_p^G$ ghost degrees of freedom.

The "condensed" sub-mesh matrix A_p in Eq. I.10 is still of a square shape. This is mainly motivated by the situations where an incomplete factorization (such as ILU) of A_p is used as M^{-1} restricted to sub-mesh p in a parallel preconditioner setting. Clearly, having only a nonzero diagonal for the last $3N_p^G$ rows of A_p means that there is effectively no computational work associated with these rows in an ILU, which is a part of the preparation work of a Krylov-subspace solver before starting the linear iterations. Moreover, the forward-backward

substitutions, which are executed within each preconditioning operation, also have negligible work associated with the "ghost" rows in the condensed A_p. Compared with the non-condensed version of A_p, which arises directly from a local discretization on the sub-mesh comprising both interior and ghost cells, the condensed A_p is superior in the amount of computational work, the amount of memory usage and traffic, as well as the preconditioning effect. The latter is due to the fact that a "natural" non-flux boundary condition is implicitly enforced on the ghost rows of the non-condensed version of A_p. This is, e.g., incompatible with a parallel Block-Jacobi preconditioner that effectively requires M^{-1} to be on the form:

$$M^{-1} = \begin{pmatrix} \left(\widetilde{A}^{II}_{p=1}\right)^{-1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \left(\widetilde{A}^{II}_{p=2}\right)^{-1} & \cdots & \mathbf{0} \\ \vdots & \cdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \cdots & \left(\widetilde{A}^{II}_{p=P}\right)^{-1} \end{pmatrix}. \tag{I.11}$$

I.4 Mesh partitioning

As mentioned in the previous section, the first step of parallelizing a reservoir simulator is a disjoint division of all the cells in the global corner-point grid, i.e., a non-overlapping mesh partitioning. We have shown how to eliminate the non-contributing computation and memory overhead, which are associated with the necessary inclusion of one layer of ghost cells per sub-mesh. The actual amount of ghost cells per sub-mesh depends on the non-overlapping division, which will be the subject of this section. One aim is to keep the number of resulting ghost cells low, for limiting the overhead of aggregated inter-process communication. At the same time, we want to ensure good convergence effectiveness of a parallel preconditioner such as the Block-Jacobi method that uses ILU as the approximate inverse of the sub-matrix A_p (Eq. I.10) per process.

For the general case of an unstructured global corner-point grid, the standard strategy for a disjoint division of the grid cells is through partitioning a corresponding graph. The graph is translated from the global grid by turning each grid cell into a graph vertex. If grid cells i and j share an interface, it is translated into a (weighted) edge between vertex i and vertex j in the graph. This standard graph-based partitioning approach is traditionally focused on load balance and low communication volume. The convergence effectiveness of a resulting parallel preconditioner is normally not considered. We will therefore propose a new edge-weighting scheme to be used in the graph partitioner, which targets specifically the reservoir simulation scenario. The objective is to provide a balance between the pure mesh-partitioning quality metrics and the convergence effectiveness.

I.4.1 Graph partitioning

A graph $G = (V, E)$ is composed of a set of vertices V and a set of edges $E \subset V \times V$ connecting pairs of vertices in V. If weights are assigned to each member of V and E through weighting functions $\sigma : V \to \mathbb{R}$ and $\omega : E \to \mathbb{R}$, then we get a weighted graph $G = (V, E, \sigma, \omega)$. The P-way graph partitioning problem is defined as follows: Partition the vertex set V into P subsets $V_1, V_2, \ldots, V_P$ of approximately equal size, while minimizing the summed weight of all the "cut edges" $e = (v_i, v_j) \in E$ connecting vertices belonging to different vertex subsets. Suppose $\mathcal{C}$ denotes the cut set of a partitioned G, containing all the cut edges. The graph partitioning problem can be formulated more precisely as a constrained optimization problem, where the objective function J is the sum of the weights of all members of $\mathcal{C}$, also called *edge-cut*:

$$\min \; J(\mathcal{C}) \qquad = \sum_{e \in \mathcal{C}} \omega(e). \tag{I.12}$$

$$\text{Subject to} \qquad \frac{\max_p \sum_{v \in V_p} \sigma(v)}{\frac{1}{P} \sum_{v \in V} \sigma(v)} < \epsilon, \tag{I.13}$$

where $\epsilon \geq 1$ is the imbalance tolerance of the load balancing constraint.

When the edge-weight function ω is uniform, i.e., each edge has a unit weight, the edge-cut is an approximation of the total volume of communication needed, e.g., before each parallel matrix-vector multiplication (see Section I.3.I.3.3).

When the edge-weight function ω is non-uniform, however, the objective function is no longer an approximation of communication overhead. As demonstrated in e.g. [5, 27], adopting non-uniform edge weights in the partitioning graph can be beneficial when partitioning linear systems with highly heterogeneous coefficients. Assigning edge weights based on the "between-cell coupling strength" can improve the quality of parallel preconditioners, such as Block-Jacobi (Eq. I.11).

Because the graph partitioning problem in Eqs. I.12-I.13 is NP-complete, solving it exactly is practically impossible. Many existing algorithms find good approximate solutions, and implementations of these are available in several open software libraries such as Metis [13], Scotch [20] and Zoltan [3].

I.4.2 Edge-weighting strategies in graph partitioning for reservoir simulation

Although the black-oil equations are nonlinear and time dependent, and the values in the global linear system $Ax = b$ vary with each Newton iteration and time step, the global corner-point grid remains unchanged. The required mesh partitioning can thus be done once and for all, at the beginning of the simulation. When translating the corner-point grid to a corresponding graph, there are two *reservoir-specific* tasks. The first is that in case two neighboring cells i and j have a zero value for the static transmissibility T_{ij} (e.g., due to a barrier fault), the corresponding edge in the graph is removed, because such a pair of cells is not

numerically coupled. The second is to include additional edges connecting vertex pairs corresponding to all the cells penetrated by a common well, because these grid cells are numerically coupled. We denote the set of well-related edges as E_w. The set containing the other regular edges is denoted by E_f. It is practical to avoid dividing a well (the penetrated cells) among multiple subdomains, and one way to achieve this is to ascribe a large edge weight to the edges in E_w. A corresponding uniform edge-weighting strategy for the edges in E_f, while ensuring that no well is partitioned between subdomains, is defined as follows:

$$\omega_u(e) = \begin{cases} \infty & e \in E_w, \\ 1 & e \in E_f. \end{cases} \tag{I.14}$$

In the above formula we ascribe weights of ∞ to the well edges. When implementing the edge-weighting scheme, the ∞-weights must be replaced by a large numerical value. We choose to use the largest possible value on a computer for the edge-weight data type.

To ensure good convergence effectiveness of a parallel preconditioner, we can modify the above uniform edge-weighting strategy by using the static transmissibility T_{ij} that lives on each cell interface (Eq. I.9). This is because T_{ij} can be used to estimate the between-cell flux, hence directly related to the magnitude of off-diagonal elements in A. Therefore, T_{ij} can be considered to describe the strength of the between-cell coupling. A commonly used edge-weighting strategy based on transmissibility is thus

$$\omega_t(e) = \begin{cases} \infty & e \in E_w, \\ T_{ij} & e = (v_i, v_j) \in E_f. \end{cases} \tag{I.15}$$

The transmissibility values can vary greatly in many realistic reservoir cases. One example of transmissibility heterogeneity can be seen with the Norne case, displayed in the histogram plot in Fig. I.3b. Here, we observe a factor of more than 10^{12} between the smallest and largest transmissibility values. Using these transmissibilities directly as the edge weights, we can get partitioning results with a potentially large communication volume. Down-scaling the weights in Eq. I.15 may help to decrease the communication overhead, while still producing partitions that yield better numerical performance than the uniform-weighted graph partitioning. We therefore propose an alternative edge-weighting strategy by using the logarithm of T_{ij} as the weight of edge $e = (v_i, v_j)$:

$$\omega_l(e) = \begin{cases} \infty & e \in E_w \\ \log\left(\frac{T_{ij}}{T_{\min}}\right) & e = (v_i, v_j) \in E_f. \end{cases} \tag{I.16}$$

I.5 Numerical experiments

In this section, we will investigate the effect of removing the non-contributing computations in solving $Ax = b$ (see Section I.3), as well as using different edge-weighting strategies for graph partitioning (see Section I.4), on parallel

simulations of a realistic reservoir model. The main objective of the experiments is to quantify the impact on the overall simulation execution time. Moreover, for the different edge-weighting strategies, we will also examine the resulting numerical effectiveness and parallel efficiency. We have conducted our experiments on the publicly available Norne model, that we will describe in Section I.5.I.5.1. In Section I.5.I.5.4 we will compare the parallel performance of a thus improved open-source simulator with industry-standard commercial reservoir simulators.

We employ the open-source reservoir simulator *Flow* [22] to conduct our experiments and test our alternative implementations and methods. *Flow* is provided by the Open Porous Media (OPM) initiative [18], which is a software collaboration in the domain of porous media fluid flow. *Flow* offers fully-implicit discretizations of the black-oil model, and accepts the industry standard ECLIPSE input format. The linear algebra and grid implementations are based on DUNE [2]. In our experiments we restrict ourselves to the BiCGStab iterative solver, combined with a parallel Block-Jacobi preconditioner that uses ILU(0) as the approximate subdomain solver. Although *Flow* supports more sophisticated preconditioners, we stick to *Flow*'s recommended default option. As of yet, experiences with *Flow* show that ILU(0) achieves better overall performance than the alternatives on most relevant models.

Parallelization of *Flow* is mostly enabled by using the Message Passing Interface (MPI) library. For matrix assembly and I/O, shared memory parallelism with the OpenMP threading library is also available. Graph partitioning in *Flow* is performed by Zoltan. The well implementation in *Flow* does not allow for the division of a well over multiple subdomains. In addition to the well related edge-weights that discourage cutting wells, a post-processing procedure in *Flow* ensures that the cells perforated by a well always are contained on a single subdomain.

I.5.1 The Norne field model

To study reservoir simulation in a realistic setting we require real-world reservoir models. One openly available model that fits this criterion is the Norne benchmark case [17], which is a black-oil model case based on a real oil field in the Norwegian Sea. The model grid consists of 44420 active cells, and has a heterogeneous and anisotropic permeability distribution. The model also includes 36 wells, which have changing controls during the simulation. Fig. I.3 depicts the Norne mesh colored by the x-directed permeability values, plus a histogram of the static transmissibility values.

In the histogram in Fig. I.3b, we can see a huge span of the transmissibilities (Eq. I.9) of the Norne benchmark case. The majority of transmissibilities have a value between 10^{-16} and 10^{-9}. This large variation is a result of the heterogeneous distribution of permeability and cell size shown in Fig. I.3a.

Because of the modest grid size of the Norne model, we will also consider a refined version [23], where the cells are halved in each direction, resulting in a model with 8 times as many grid cells as the original model. The number of active cells in the refined Norne model is 355360.

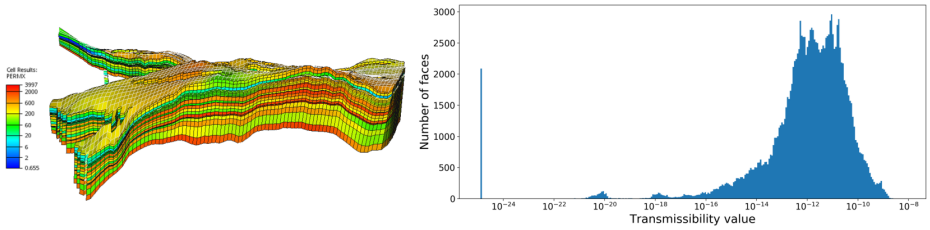


Figure I.3: Permeability distribution (a) in the x direction of the Norne grid. A histogram plot (b) of the transmissibility associated with cell interfaces from the Norne reservoir simulation case. The spike in the left end of the plot represents transmissibilities with value 0.

We conducted our experiments of the Norne models on two computing clusters: Abel [**Abel**] and Saga [**Saga**]. The Abel cluster consists of nodes with dual Intel Xeon E5-2670 2.6GHz 8-core CPUs interconnected with an FDR InfiniBand (56 Gbits/s) network, whereas Saga is a cluster of more modern dual socket Intel Xeon-Gold 6138 2.0GHz 20-core CPUs interconnected with an EDR InfiniBand network (100 Gbits). The nodes on Abel and Saga have a total of 16 and 40 cores respectively. All experiments that use more MPI-processes than there are cores available on a single node are conducted on multiple nodes.

In all experiments using *Flow* on the original and refined Norne models, we have turned off the OpenMP multithreading for system assembly inside *Flow*.

I.5.2 Impact of removing non-contributing computations

We study the impact of non-contributing computations on the performance of linear algebra kernels and the overall performance of *Flow*, by carrying out experiments of the original Norne model described above. For these experiments we have used the transmissibility edge weights in Eq. I.15, the default choice of *Flow*, in the graph partitioning scheme.

Fig. I.4 shows the ratio of ghost cells, i.e., $(\sum_{p=1}^{P} N_p^G)/N$, as a function of the number of subdomains used. We can see that the ghost cells can make up a significant proportion. For example, with $P = 128$, the ghost cell ratio is as high as 55%. As discussed in Section I.3, non-contributing computations can arise due to the ghost cells. Because ghost cells increase as a proportion of total cells with an increasing number of MPI-processes, avoiding ghost cell related non-contributing computations is crucial for achieving good strong scaling.

To show the negative impact of non-contributing computations on the performance of solving $Ax = b$, we present the execution time of the linear algebra kernels used in the original Norne case in Fig. I.5. It displays time measurements of sparse matrix-vector multiplication (SpMV), ILU's forward-backward substitution and inner product (IP), with and without the non-

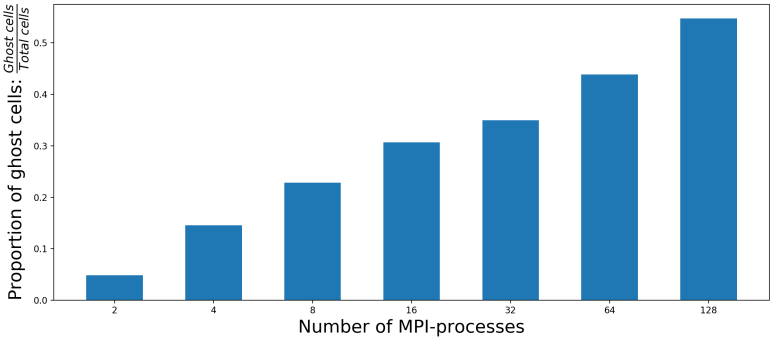


Figure I.4: Ratio of the ghost cells related to partitioning the Norne reservoir mesh.

contributing computations. These time measurements are attained on the Abel cluster using selected matrices and vectors from the *Flow* Norne simulation. Using these matrices and vectors, SpMV, ILU forward-backward substitution and inner product operations are executed and timed.

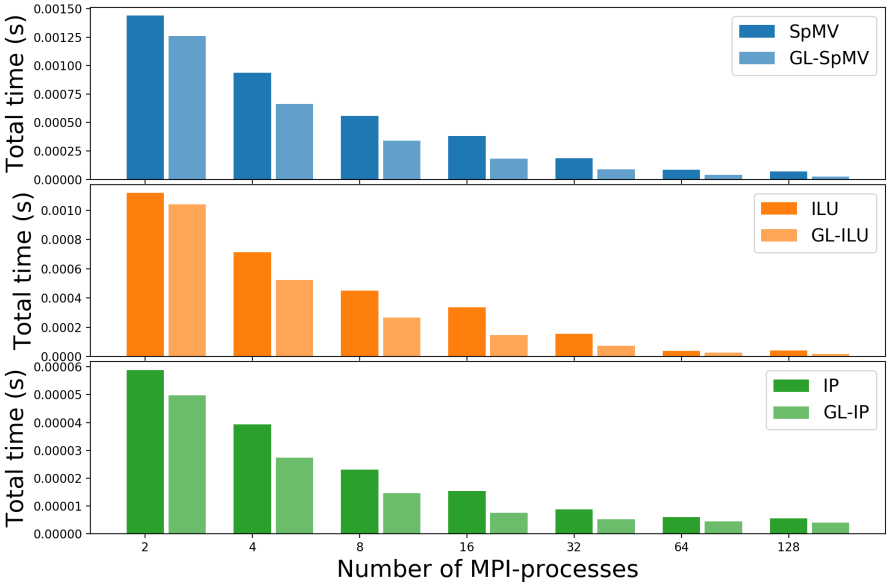


Figure I.5: Time measurements (in seconds) of parallel linear algebra kernels for the Norne case, obtained on the Abel cluster, with and without the non-contributing computations. The GL prefix means the ghost-related non-contributing computations are removed.

Fig. I.5 shows a significant time improvement for the linear algebra kernels,

due to removing the non-contributing computations as proposed in Section I.3. Because the IP operation includes a collective reduction communication, it does not scale as well as the SpMV and ILU operations. We therefore observe that the relative improvement attained by removing the non-contributing IP computations start to decrease when the number of processes is higher than 16. A closer look at the performance of the linear algebra kernels is presented in Fig. I.6 for the case of 16 MPI-processes. Here, the per-process execution times of SpMV, ILU and IP are displayed, with and without the non-contributing computations.

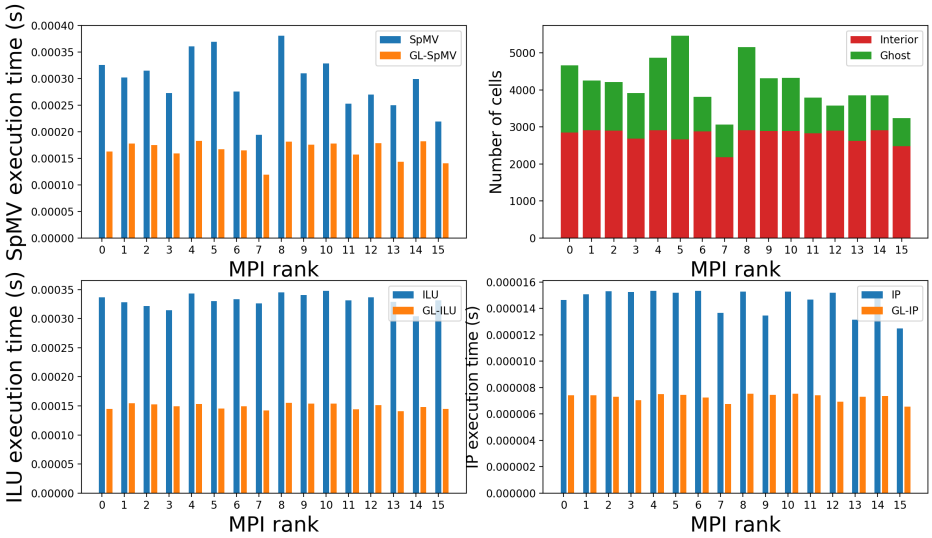


Figure I.6: Per-process time usage of linear algebra kernels on an Abel node, and distribution of ghost/interior cells when the number of MPI-processes is 16.

The top right plot of Fig. I.6 displays the per-process numbers of interior and ghost cells. We notice a very uneven distribution of ghost cells among the processes. We also observe that the load imbalance induced by the ghost cells impacts the performance of the linear algebra operations, especially for SpMV.

The results displayed in Fig. I.5 and Fig. I.6 show the impact of ghost cells on the performance of key linear algebra operations, and the benefit of avoiding the non-contributing computations. To see the impact on the overall simulation time, we compare our improved implementation of *Flow* with the 2018.10 release of OPM's *Flow*, which is the latest release that does not contain any of the optimizations mentioned in this paper. The comparison results are displayed in Fig. I.7, where the overall execution time and total number of BiCGStab iterations are presented.

In Fig. I.7 we observe that our improved implementation of *Flow* achieves a significant improvement in both execution time and total iteration count. The latter is due to using the condensed local sub-matrix A_p of form Eq. I.10, instead

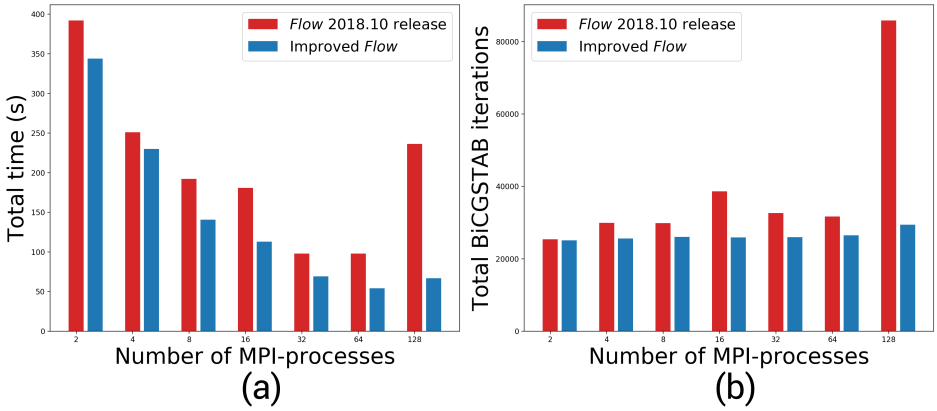


Figure I.7: Overall execution time in seconds (a) and total number of BiCGStab iterations (b) of the 2018.10 release of *Flow* and our improved implementation, when applied to the Norne model. The simulations were conducted on the Abel cluster, and when the number of MPI-processes exceeded 16 we used multiple computational nodes.

of the non-condensed version of A_p, in the ILU subdomain solver. This leads to better convergence of the Block-Jacobi preconditioner. Note that removing ghost related non-contributing computations and improving convergence only impact the performance of the simulators linear solver. The main computational work of the reservoir simulator also consists of a system assembly part. Therefore, for example, we only achieve a speedup of around 3.5 in the $P = 128$ case, despite 2.9 times smaller iteration count, and about 2 to 3 times faster SpMV and ILU operations.

I.5.3 Impact of different edge-weighting strategies

In this subsection we will study and compare the different edge-weighting strategies presented in Section I.4. We are ultimately interested in how the uniform weights (Eq. I.14), transmissibility weights (Eq. I.15) or logarithmic transmissibility weights (Eq. I.16) impact the overall performance of the reservoir simulation, but we will also consider their impact on partitioning quality and numerical effectiveness. The strategies are enforced before passing the graph to the Zoltan graph partitioner inside *Flow*, and in our experiments we have used a 5% imbalance tolerance ($\epsilon = 1.05$). All simulation results in this subsection are attained with the improved *Flow* where we use the ghost-last procedures described in Section I.3. Fig. I.8 displays a visualization of a $P = 8$ partitioning of the Norne mesh for the different edge-weighting strategies. We observe that subdomains generated by the logarithmic and transmissibility edge-weighted partitioning schemes are less clean cut and more disconnected, than the subdomains resulting from the uniform edge-weighted partitioning scheme.

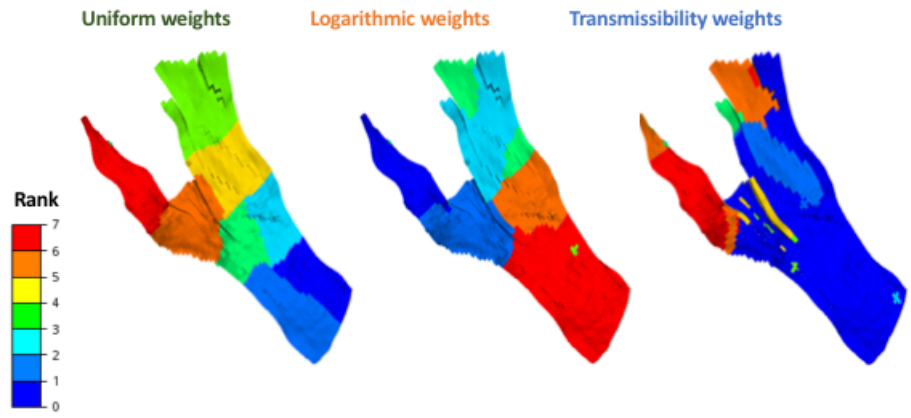


Figure I.8: Resulting subdomains when partitioning the Norne mesh into 8 parts using uniform, logarithmic and transmissibility edge weights. Each color represent one subdomain.

I.5.3.1 Mesh partitioning quality

We start our tests by focusing on how the different edge-weighting strategies affect partitioning quality, when used to partition the original and refined Norne meshes. In Fig. I.9 we report the total communication volume for the three partitioning schemes. The total communication volume is equal to the sum of the DoFs associated with ghost cells over all processes, $3\sum_p N_p^G$, multiplied with the data size of double precision floats, which is 8 bytes. The partitioning results for the original Norne mesh are displayed in Fig. I.9a, whereas Fig. I.9b is for the refined Norne mesh.

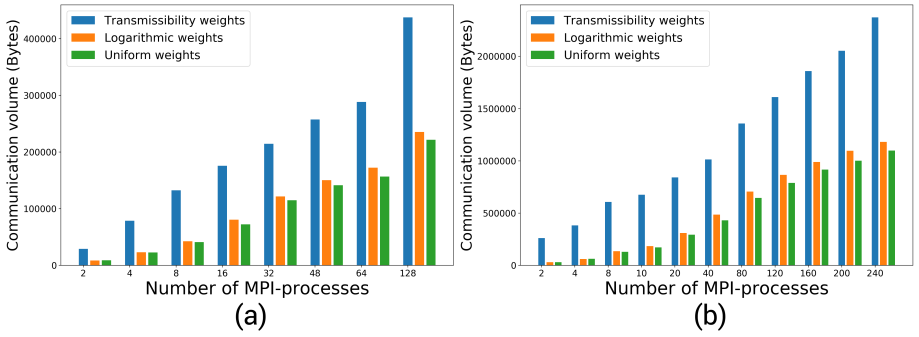


Figure I.9: Total communication volume in bytes for the partitioned Norne mesh (a) and the refined Norne mesh (b) when using different edge-weighting strategies.

In the plots of Fig. I.9 we observe that the partitions obtained using the

transmissibility edge-weights yield significantly higher communication volume than the two other alternatives. This holds for both the original and the refined Norne models, and for all counts of MPI-processes P. We also notice that although the uniform edge-weighting strategy outperforms the logarithmic scheme, the differences in the resulting communication volume are relatively small.

To precisely measure how the difference in communication volume affects the actual communication overhead, we consider the execution time of the MPI data transfer operations required to perform before a parallel SpMV. The data transfer operations are implemented in the DUNE function `copyOwnerToAll`. In Fig. I.10 we present the execution time of `copyOwnerToAll` related to the original and refined Norne meshes on the Abel and Saga clusters, corresponding to the communication volumes shown in Fig. I.9.

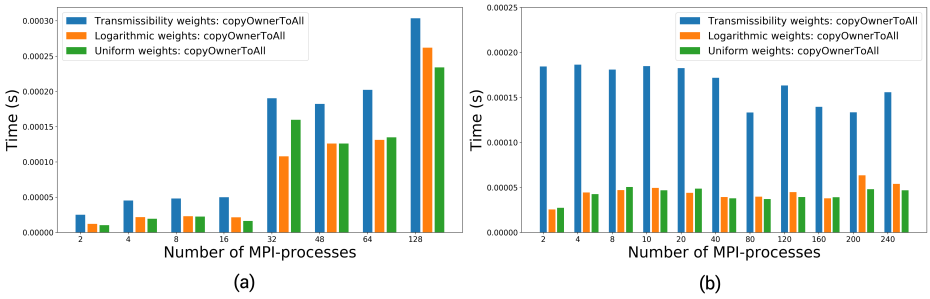


Figure I.10: Communication overhead, i.e., time usage of DUNE's `copyOwnerToAll` function on the Abel (a) and Saga (b) clusters. Multiple computational nodes are used when the number of MPI-processes exceeded 16 on Abel (a) and 40 on Saga (b).

The plots in Fig. I.10 demonstrate that using transmissibility edge-weights yields larger communication overhead than the uniform and logarithmic alternatives on both the Abel and Saga clusters. The relatively high execution time of `copyOwnerToAll`, resulting from the transmissibility edge-weighted partitioning scheme, can partially be explained by the communication volume displayed in Fig. I.9. However, the `copyOwnerToAll` execution time is also affected by the hardware. For example, the jump in execution time between $P = 16$ and $P = 32$, observed in Fig. I.10a, occurs when we start using two instead of one computational node, and there is thus inter-node communication over the network. A similar jump in execution time is not observed in Fig. I.10b on the Saga cluster between $P = 40$ and $P = 80$, because the Saga interconnect is better than the Abel interconnect (100 Gbits vs. 56 Gbits).

I.5.3.2 Numerical and overall performance

We can find the impact of the edge-weighting schemes on *Flow*'s numerical performance by looking at the total number of Block-Jacobi/ILU0 preconditioned

BiCGStab iterations needed to complete each simulation of the original Norne benchmark case. This iteration count is displayed in Fig. I.11 for the three edge-weighting strategies.

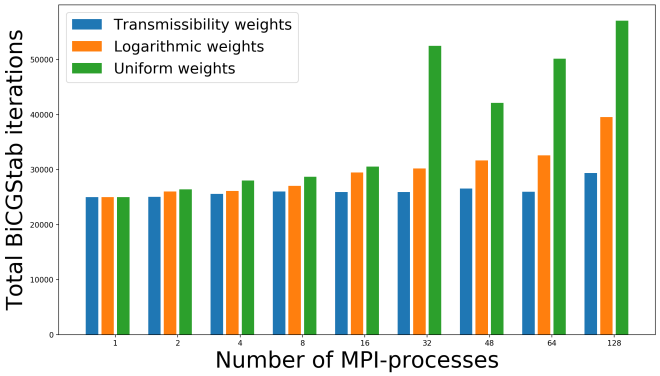


Figure I.11: Total number of BiCGStab iterations to run the Norne black-oil benchmark case for three different partitioning strategies and varying numbers of MPI-processes.

One interesting finding from Fig. I.11 is that the transmissibility weighted partitioning strategy can keep the number of BiCGStab iterations almost completely independent of number of subdomains. It means that the transmissibility edge weights are indeed good for the convergence of the parallel Block-Jacobi preconditioner. On the opposite side, the uniform edge-weighting scheme leads to a large increase in the BiCGStab iterations, when the number of subdomains is large. This is contrary to its ability of keeping the communication overhead low. The logarithmic transmissibility edge-weighting scheme seems a good compromise, which is confirmed by Fig. I.12 showing the total simulation time.

In Fig. I.12 we observe that simulations using the logarithmic edge-weight strategy outperforms simulations using the transmissibility and uniform edge-weights for all numbers of processes. Although significant, the improvements achieved by logarithmic weights in comparison to transmissibility weights are modest in absolute terms. However, the relative improvement in execution time increase with the number of processes involved. For 2 processes we observe a 4.7% reduction in simulation execution time. For 48 processes the reduction is 24.5%. The BiCGStab iterations count required to complete the simulation of the Norne case is significantly higher when using logarithmic and uniform edge-weights instead of transmissibility edge-weights, especially when using more than 16 processes. Despite higher iteration counts, logarithmic and uniform edge-weights yield equally good or better performance than transmissibility edge-weights. There are two reasons for this. First, as demonstrated in Fig. I.10, using logarithmic and uniform edge-weights results in lower communication overhead, which results in lower execution time per BiCGStab iteration. Second,

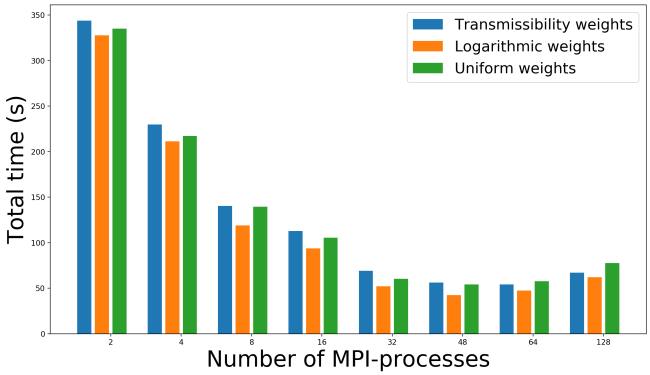


Figure I.12: Overall time usage of the parallel *Flow* reservoir simulator when applied to the Norne benchmark case, measured on the Abel cluster. Beyond 16 MPI-processes, the simulations were conducted on multiple computational nodes.

the uniform and logarithmic edge-weights result in a lower number of per-process ghost cells. This has a positive impact on system assembly performance.

We also notice little or no reduction in execution time beyond 48 processes for all edge-weighting strategies. From $P = 64$ to $P = 128$ we even see an increase in simulation execution time. This is not unexpected, because at this point the mesh partitioning produces an average of only $44420/128 \approx 347$ cells per process, which correspond to around 1041 DoFs. When DoFs per process reaches this point we are beyond the strong scaling limit, so adding more hardware resources yields no benefit.

Performance results for the refined Norne model, measured on the Saga cluster, are displayed in Fig. I.13. Here, we have used $P = 2, 4, 8, 10, 20, 40$ MPI-processes on a single compute node, as well as $P = 80, 120, 160, 200, 240$ MPI-processes on two to six nodes. Execution times are presented in Fig. I.13a and BiCGStab iterations in Fig. I.13b.

The results for the refined Norne model attained on the Saga cluster, displayed in Fig. I.13, are similar to the Norne results on the Abel cluster. Transmissibility edge-weights yield better linear solver convergence than the logarithmic and uniform alternatives, but the overall performance improves when using logarithmic and uniform edge-weights. We observe that simulations ran with logarithmic edge-weights had the lowest execution time for all number of MPI-processes except $P = 2$ and $P = 8$. The improvement over pure transmissibility edge-weights again appears quite modest. However, the benefit increases with the number of processes. For $P = 2$, 4 and 8 logarithmic transmissibility edge-weights yield a 1.3%, 9.1% and 3.6%, reduction in execution time, while for $P = 80$, 120 and 160 the improvement is respectively 40.1%, 29.5% and 47.6%.

In Fig. I.13a we observe an expected diminishing parallel efficiency for an

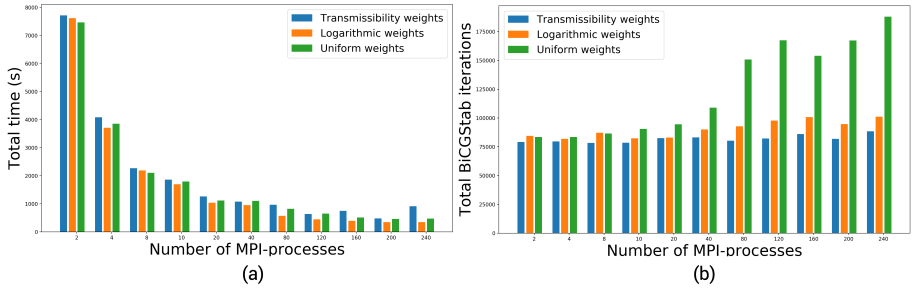


Figure I.13: Execution time (a) and iteration count (b) of the *Flow* reservoir simulator when applied to the refined Norne benchmark case on the Saga cluster. Beyond 40 MPI-processes the simulations were conducted on multiple computational nodes.

increasing number of MPI-processes. Simulation execution time increases for all edge-weight schemes between 200 and 240 processes. At $P = 240$ there is around 1500 cells and 4500 DoFs per process, and we have reached the strong scaling limit.

I.5.4 Comparing *Flow* with industry-standard simulators

The Norne model exhibits several features rarely found in academically available data sets. It is therefore interesting to compare the performance of the improved *Flow* simulator, when applied to Norne, with commercial alternatives. We consider two industry-standard simulators, *Eclipse* 100 version 2018.2 and *Intersect* version 2019.1. For our experiments *Eclipse* and *Intersect* were used in their default configurations. All simulations presented in this subsection were done on a dual socket workstation with two Intel E5-2687W processors with a total of 16 cores available. The system has 128GB of memory, which was sufficient for all simulations. The CPUs have a base frequency of 3.1GHz and a turbo frequency rating of 3.8GHz. The operating system was Red Hat 6.

We should note that the three simulators have different numerics internally. Unfortunately, only *Flow* has open source code, so we cannot investigate the implementation details of the other two. We refer the reader to the publicly available information on the numerics of the two proprietary simulators. While mpirun is handled internally by *Eclipse* and *Intersect*, we use mpirun directly for parallel simulations with *Flow*. The only other command-line option used by mpirun is `-map-by numa`. This option is particularly important for runs with two processes, since it ensures that the two are distributed on different sockets, taking advantage of cache and memory bandwidth on both. Without the option, the runtime may put both MPI processes on the same NUMA-node. Results from the previous subsection have shown that the parallel *Flow* simulator works best with logarithmic transmissibility edge-weights for the Norne case. The *Flow* simulation results presented in this subsection therefore use the logarithmic

transmissibility edge-weighting scheme.

The MPI implementation in *Eclipse* is simplistic. According to the documentation it simply does domain decomposition by dividing the mesh cells evenly along one axis dimension. Nevertheless, for some models (the Norne model is actually one of them) this works well.

The comparison in parallel performance between *Flow*, *Eclipse* and *Intersect* on the Norne benchmark case is presented in Fig. I.14. In this plot we also include results from simulations where multithreading is activated with two OpenMP threads per MPI process for *Flow* and *Intersect*. Multithreading is not activated when 16 MPI-processes is used, because of a lack of remaining hardware resources.

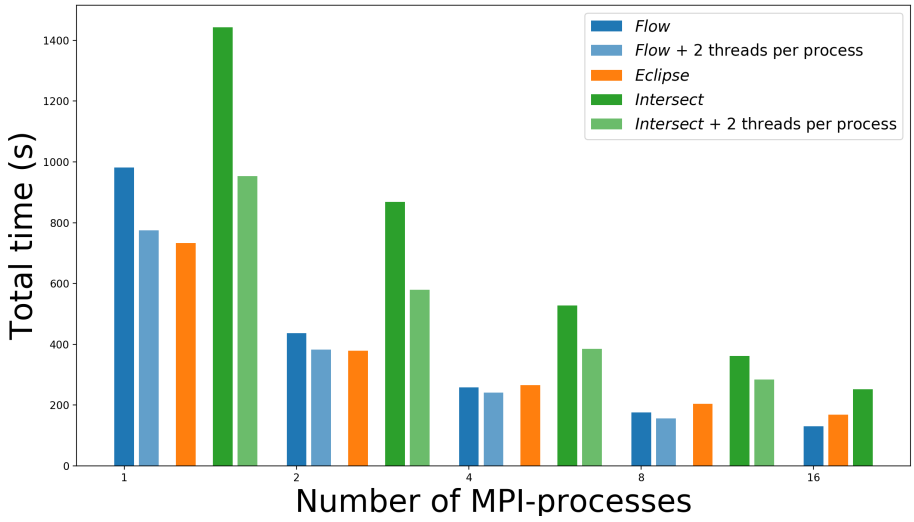


Figure I.14: Execution time (in seconds) of *Flow*, *Eclipse* and *Intersect* on the Norne benchmark case. We include results for *Flow* and *Intersect* simulations where multithreading with two OpenMP threads per MPI-process is activated.

The results presented in Fig. I.14 show that the parallel *Flow* simulator outperforms *Eclipse* and *Intersect*, for $P \geq 4$. Although *Eclipse* is faster than *Flow* for serial runs, *Flow* scales better than *Eclipse* on the Norne model. *Flow* achieves a speedup of 7.6 for $P = 16$ compared to *Eclipse*'s speedup of 4.4. *Intersect* performs rather poorly on Norne, but it scales better than *Eclipse*. Activating multithreading yields improved performance for both *Flow* and *Intersect*.

I.6 Related work

Because of the demand for large-scale reservoir simulations in the petroleum industry, there exist several commercial and in-house simulators that are able

to take advantage of parallel computing platforms. Examples include the Saudi Aramco POWERS [6, 8, 24] and GigaPOWERS [7] simulators, which are able to run simulations on reservoir grids with billions of cells.

Graph partitioning is often used to enable parallel reservoir simulation [10, 11, 16, 29]. However, no reservoir simulation specific considerations were made in these cases. In [15] the authors suggest two guiding principles for achieving good load balancing when performing mesh partitioning for thermal reservoir simulation. First, grid cells and simulation wells should be evenly distributed between the processors. Second, if faults are present in the reservoir, they should serve as subdomain boundaries between the processes.

In the PhD thesis [30] a mesh partitioning scheme based on edge-weighted graph partitioning is described. The edge weights are formed based on the transmissibility on the interface of the cell blocks in the reservoir mesh. Additionally, the presence of wells in the reservoir is accounted for by modifying the partitioning graph. In [28] the authors derive similar strategies for partitioning non-uniform meshes in the context of computational fluid dynamics. A graph partitioning approach with edge-weights corresponding to the cell face area is implemented. The aim of this approach is to improve solver convergence, by accounting for the coefficient matrix heterogeneity introduced by the non-uniform mesh. The authors of [28] do not only focus on how this edge-weighted approach can affect the numerical performance, but also consider measures of partitioning quality, such as edges cut and number of processor neighbors. Despite poorer partitioning quality, the edge-weighted graph partitioning scheme gave the best overall performance.

Several attempts at incorporating coefficient matrix information into the partitioning of the linear systems have been made [5, 9, 12, 26, 27]. In [5] the authors add coefficient-based edge weights to the partitioning graph, and demonstrate improved numerical performance in comparison with standard non-weighted schemes. The previously mentioned paper [27] presents a spectral graph partitioning scheme that outperforms standard graph partitioners, even with weighted edges, for symmetric positive definite systems with heterogeneous coefficients.

I.7 Conclusion

In this paper, we have given a detailed description of the domain decomposition strategy used to parallelize the simulation of fluid flow in petroleum reservoirs. We proposed an improved parallel implementation of the linear solver based on a local “ghost last" reordering of the grid cells. We also investigated the use of edge-weighted graph partitioning for dividing the reservoir mesh. A new edge-weighting scheme was devised with the purpose to maintain a balance between the numerical effectiveness of the Block-Jacobi preconditioner and the communication overhead.

Through experiments based on the Norne black-oil benchmark case, we showed that the ghost cells make up an increasing proportion of the cells in the

decomposed sub-meshes when the number of processes increases. Further we found that removing non-contributing calculations related to these ghost cells can give a significant improvement in the parallel simulator performance.

For the Norne black-oil benchmark case, which has extremely heterogeneous petrophysical properties, using edge-weights directly derived from these properties can have negative consequences for the overall performance. Although this approach yields satisfactory numerical effectiveness, it does not make up for the increase in the communication overhead. The large communication overhead associated with the default transmissibility-weighting scheme is due to poor partitioning quality, in particular with respect to the communication volume and number of messages. The scaled logarithmic transmissibility approach, which also uses non-uniform edge-weights, yields a much better partitioning quality. Even though the numerical effectiveness due to the logarithmic scheme may be lower than the transmissibility weighted scheme, it can still result in a better overall performance.

Acknowledgements

This research was conducted using the supercomputers Abel and Saga in Norway.

Funding

This research was funded by the SIRIUS Centre for Scalable Data Access.

Abbreviations

PDE, partial differential equation; ILU, incomplete LU; OPM, open porous media; MPI, Message Passing Interface; SpMV, sparse matrix-vector multiplication, IP, inner product.

Availability of data and materials

Data and source codes are available upon reasonable request.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

All authors contributed to the writing of the manuscript. AT implemented the improved version of *Flow* and performed most of the numerical experiments. XC and ABR provided research advice and guidance. ABR performed experiments

with industry-standard simulators. All authors read and approved the final manuscript.

References

[1] Aziz, K. and Settari, A. *Petroleum Reservoir Simulation*. London: Applied Science Publishers, 1979.

[2] Bastian, P. et al. "A generic grid interface for parallel and adaptive scientific computing. Part II: implementation and tests in DUNE". In: *Computing* vol. 82, no. 2-3 (2008), pp. 121–138.

[3] Boman, E. et al. "Zoltan 3.0: parallel partitioning, load-balancing, and data management services; user's guide". In: *Sandia National Laboratories, Albuquerque, NM* vol. 2, no. 3 (2007).

[4] Chen, Z., Huan, G., and Ma, Y. *Computational Methods for Multiphase Flows in Porous Media*. Vol. 2. Philadelphia: SIAM, 2006.

[5] Cullum, J. K., Johnson, K., and Tůma, M. "Effects of problem decomposition (partitioning) on the rate of convergence of parallel numerical algorithms". In: *Numerical Linear Algebra with Applications* vol. 10, no. 5-6 (2003), pp. 445–465.

[6] Dogru, A. H. et al. "A massively parallel reservoir simulator for large scale reservoir simulation". In: *SPE symposium on reservoir simulation*. 1999, pp. 73–92.

[7] Dogru, A. H. et al. "A next-generation parallel reservoir simulator for giant reservoirs". In: *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers. 2009.

[8] Dogru, A. H. et al. "A parallel reservoir simulator for large-scale reservoir simulation". In: *SPE Reservoir Evaluation & Engineering* vol. 5, no. 1 (2002), pp. 11–23.

[9] Duff, I. S. and Kaya, K. "Preconditioners based on strong subgraphs". In: *Electronic Transactions on Numerical Analysis* vol. 40 (2013), pp. 225–248.

[10] Elmroth, E. "On grid partitioning for a high-performance groundwater simulation software". In: *Simulation and Visualization on the Grid*. Berlin, Heidelberg: Springer, 2000, pp. 221–234.

[11] Guo, X., Wang, Y., and Killough, J. "The application of static load balancers in parallel compositional reservoir simulation on distributed memory system". In: *Journal of Natural Gas Science and Engineering* vol. 28 (2016), pp. 447–460.

[12] Janna, C., Castelletto, N., and Ferronato, M. "The effect of graph partitioning techniques on parallel block FSAI preconditioning: a computational study". In: *Numerical Algorithms* vol. 68, no. 4 (2015), pp. 813–836.

[13] Karypis, G. and Kumar, V. "A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices". In: *University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN* vol. 38 (1998).

[14] Lie, K.-A. *An Introduction to Reservoir Simulation using MATLAB/GNU Octave*. Cambridge: Cambridge University Press, 2019.

[15] Ma, Y. and Chen, Z. "Parallel computation for reservoir thermal simulation of multicomponent and multiphase fluid flow". In: *Journal of Computational Physics* vol. 201, no. 1 (2004), pp. 224–237.

[16] Maliassov, S., Shuttleworth, R., et al. "Partitioners for parallelizing reservoir simulations". In: *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers. 2009.

[17] *Norne reservoir simulation benchmark*. https://github.com/OPM/opm-data. 2019.

[18] *Open Porous Media Initiative*. http://opm-project.org. Accessed: 2023-04-11. 2017.

[19] Peaceman, D. W. "Interpretation of well-block pressures in numerical reservoir simulation (includes associated paper 6988)". In: *Society of Petroleum Engineers Journal* vol. 18, no. 03 (1978), pp. 183–194.

[20] Pellegrini, F. and Roman, J. "Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs". In: *International Conference on High-Performance Computing and Networking*. Springer. 1996, pp. 493–498.

[21] Ponting, D. K. "Corner point geometry in reservoir simulation". In: *ECMOR I - 1st European Conference on the Mathematics of Oil Recovery*. 1989.

[22] Rasmussen, A. F. et al. "The Open Porous Media Flow reservoir simulator". In: *Computers & Mathematics with Applications* vol. 81 (2021), pp. 159–185.

[23] *Refined Norne reservoir simulation deck*. https://github.com/andrthu/opm-data/tree/refined-222-norne/refined-norne. 2020.

[24] Al-Shaalan, T. M., Fung, L. S. K., Dogru, A. H., et al. "A scalable massively parallel dual-porosity dual-permeability simulator for fractured reservoirs with super-k permeability". In: *SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers. 2003.

[25] Saad, Y. *Iterative Methods for Sparse Linear Systems*. Philadelphia: SIAM, 2003.

[26] Saad, Y. and Sosonkina, M. "Non-standard parallel solution strategies for distributed sparse linear systems". In: *International Conference of the Austrian Center for Parallel Computation*. Springer. 1999, pp. 13–27.

[27] Vecharynski, E., Saad, Y., and Sosonkina, M. “Graph partitioning using matrix values for preconditioning symmetric positive definite systems”. In: *SIAM Journal on Scientific Computing* vol. 36, no. 1 (2014), A63–A87.

[28] Wang, M. et al. “Mesh partitioning using matrix value approximations for parallel computational fluid dynamics simulations”. In: *Advances in Mechanical Engineering* vol. 9, no. 11 (2017), p. 1687814017734109.

[29] Wu, Y.-S. et al. “An efficient parallel-computing method for modeling nonisothermal multiphase flow and multicomponent transport in porous and fractured media”. In: *Advances in Water Resources* vol. 25, no. 3 (2002), pp. 243–261.

[30] Zhong, H. “Development of a New Parallel Thermal Reservoir Simulator”. PhD thesis. University of Calgary, 2016.

Paper II

Detailed modeling of heterogeneous and contention-constrained point-to-point MPI communication

Andreas Thune, Sven-Arne Reinemo, Tor Skeie, and Xing Cai

Abstract

The network topology of modern parallel computing systems is inherently heterogeneous, with a variety of latency and bandwidth values. Moreover, contention for the bandwidth can exist on different levels when many processes communicate with each other. Many-pair, point-to-point MPI communication is thus characterized by heterogeneity and contention, even on a cluster of homogeneous multicore CPU nodes. To get a detailed understanding of the individual communication cost per MPI process, we propose a new modeling methodology that incorporates both heterogeneity and contention. First, we improve the standard max-rate model to better quantify the actually achievable bandwidth depending on the number of MPI processes in competition. Then, we make a further extension that more detailedly models the bandwidth contention when the competing MPI processes have different numbers of neighbors, with also non-uniform message sizes. Thereafter, we include more flexibility by considering interactions between intra-socket and inter-socket messaging. Through a series of experiments done on different processor architectures, we show that the new heterogeneous and contention-constrained performance models can adequately explain the individual communication cost associated with each MPI process. The largest test of realistic point-to-point MPI communication involves 8,192 processes and in total 2,744,632 simultaneous messages over 64 dual-socket AMD Epyc Rome compute nodes connected by InfiniBand, for which the overall prediction accuracy achieved is 84%.

II.1 Introduction

Modern platforms of parallel computing are heterogeneous at least with respect to the interconnect. Even on a system purely based on multicore CPUs, the

connectivity between the CPU cores has several layers. The cores that belong to the same CPU socket can communicate more efficiently than those between sockets, because the inter-socket memory bandwidth is lower than the intra-socket counterpart. At the cluster level, between any pair of compute nodes, the communication speed is even lower and can depend on the actual location of the nodes on the network setup. All these levels of interconnect heterogeneity will translate into vastly different values of the effective latency and bandwidth of point-to-point MPI communication.

Another complicating factor for many-pair, point-to-point MPI communication on today's parallel platforms is the potential competition between different MPI processes. This is because each CPU socket can, if needed, support a large number of concurrent MPI processes. Contention arises when multiple pairs of sending-receiving processes simultaneously communicate over the same connection. Such contention may exist, in different magnitudes, over the entire network. Moreover, the competition situation is often *dynamically* changing; for instance, an MPI process pair communicating a small message may complete before the other MPI process pairs, resulting in a reduced level of contention.

This paper aims to detailedly model the per-process overhead associated with realistic, many-pair, point-to-point MPI communication. We want to improve the state-of-the-art quantitative models of point-to-point MPI communication by a new methodology for modeling the bandwidth contention due to a large number of MPI processes that compete against each other. Here, communication can exist on different levels: intra-socket, inter-socket and inter-node. In addition, we target the real-world situation where different MPI processes can have different numbers of neighbors to exchange data with, while the size of each message is also highly non-uniform. One typical example of such a heterogeneous scenario arises from numerically solving a partial differential equation (PDE) over an irregular solution domain. The first step of parallelizing a mesh-based PDE solver is to partition an unstructured computational mesh, which covers the irregular solution domain, into a desirable number of subdomains each assigned to an MPI process. The usual partitioning result is that the number of the nearest neighbors varies from process to process, and so does the size of each MPI message.

Specifically, we will propose a new modeling methodology that quantifies both heterogeneity and contention. At the same time, we want to inherit a level of simplicity from the fundamental *postal* model [2, 12] that describes a single pair of MPI processes, and the successor max-rate model [11]. The elegantly simple postal model relies on only two parameters to quantify the cost of point-to-point communication, i.e., a start-up latency τ and a bandwidth value BW. The max-rate model lets BW depend linearly on the number of competing MPI processes while limited from above by a maximum bandwidth value, which is prescribed as the third parameter. Our new performance models are also based on a fair competition among the MPI processes, but the value of BW will depend *dynamically* on the actual number of competing MPI processes and how these processes affect each other across two specific levels: intra-socket and inter-socket. We focus our modeling and experiments on large messages,

that use the rendezvous MPI protocol. The contributions of our work are as follows:

- We extend the `osu_bibw` micro-benchmark of MVAPICH [18] to easily tabulate the various τ and BW values, both depend on the connection type and the latter is also a function of the number of competing MPI processes. These tabulated values serve as a characterization of the communication performance of a heterogeneous interconnect.
- We improve the accuracy of the max-rate model [11] for the case of multiple MPI process pairs concurrently exchanging equal-sized messages. Specifically, the tabulated BW values replace an often idealized relationship between the achievable bandwidth and the number of competing MPI pairs.
- We propose a "staircase" strategy to detail the contention between many MPI processes with varying numbers of neighbors and message sizes, when competing over a single level of interconnect.
- We extend the single-level "staircase" modeling to mixed-level "staircase" modeling that also quantifies the interaction between two particular communication levels: intra-socket and inter-socket.

The remainder of the paper is organized as follows. Section II.2 introduces a new bi-directional multi-pair micro-benchmark, which can be used to pinpoint the achievable bandwidth as a function of the competing send-receive pairs. Section II.3 is devoted to a new "staircase" modeling strategy that can be adopted to handle the various types of heterogeneity, more specifically, non-uniform message size, a varied number of messages per MPI process, and the interaction between intra-socket and inter-socket communication. Section II.4 tests the "staircase" strategy and the resulting new models in realistic cases of many-pair, point-to-point MPI communication. Section II.5 places our current work with respect to the existing work on modeling MPI point-to-point communication, whereas Section II.6 provides some concluding remarks. The source code that implements the mixed-level modeling strategy can be found in the appendix.

II.2 Detailing bandwidth contention

As mentioned above, the postal model [2, 12] provides an elegant and simple way of quantifying the time needed to pass a message between a single pair of MPI send-receive processes. Its formula is as follows:

$$T(s) = \tau + \frac{s}{BW_{\mathrm{SP}}}, \tag{II.1}$$

where the constant parameter τ denotes the start-up latency, s denotes the size of the MPI message, and the constant parameter BW_{SP} denotes the communication bandwidth. The subscript "SP" stands for single-pair and thus highlights the applicability of the postal model. Experiments (see e.g. [13]) have shown that

this two-parameter model can produce estimates of $T(s)$ that agree very well with the actual single-message time usages, as long as the message size s is within the regime of the same protocol (i.e., short, eager, or rendezvous). It also means that each protocol is associated with its specific set of τ and BW_{SP} parameters.

The max-rate model [11] extends the postal model by considering N competing MPI messages belonging to N send-receive process pairs. If all the messages are of the same size s, they will have the same time usage due to a fair competition for the bandwidth. The simplest formula of the max-rate model is as follows:

$$T(N, s) = \tau + \frac{N \cdot s}{BW_{\mathrm{MP}}(N)}. \tag{II.2}$$

In the above formula, $BW_{\mathrm{MP}}(N)$ is meant to model a shared bandwidth to be fairly competed among the N messages, and the subscript "MP" stands for multi-pair. The dependency of BW_{MP} on N is considered by the max-rate model in its simplest form as follows:

$$BW_{\mathrm{MP}}(N) = \min\left(N \cdot BW_{\mathrm{SP}}, BW_{\mathrm{max}}\right), \tag{II.3}$$

where BW_{max} denotes the upper limit of the achievable communication bandwidth, i.e., a *max rate*. The existence of BW_{max} illustrates a saturation effect, which applies to both communication over a network connection and communication through shared memory.

An extended max-rate model was presented in [6] to consider variations in the message size and number of messages per process. It mainly targets inter-node communication:

$$T = M \cdot \tau + \max\left(\frac{s_{\mathrm{total}}}{BW_{\mathrm{max}}}, \frac{s_{\mathrm{max}}}{BW_{\mathrm{SP}}}\right). \tag{II.4}$$

Here, M is the maximum number of messages per process, s_{total} and s_{max} denote, respectively, the total messaging volume per node and the maximum per-process volume. We note that (II.4) only models the slowest process per node.

While the max-rate model is simple to use, it has several weaknesses. First, the actual $BW_{\mathrm{MP}}(N)$ value may not be linearly proportional to N before hitting the upper limit BW_{max}, especially for intra- and inter-socket traffic. Second, the bandwidth contention may not be accurately modeled when the processes have largely varying message numbers and/or sizes. Third, only the slowest process is modeled by (II.4), not the earlier finishing processes.

In the remainder of this section we will improve the max-rate model with respect to its first shortcoming. This will be achieved by extending a micro-benchmark of MVAPICH [18]. The other two shortcomings will be addressed in Section II.3.

II.2.1 A new micro-benchmark

We want to pinpoint the actual BW_{MP} values through measurements obtained by a simple benchmark, instead of using the often idealized formula (II.3).

Specifically, we will adopt a new "bi-directional multi-pair" micro-benchmark, as described in Algorithm 1. It is a modification of the `osu_bibw` benchmark of MVAPICH [18]. The new micro-benchmark involves P processes to form $\frac{P}{2}$ sender-receiver pairs, each simultaneously handling two equal-sized messages of opposite directions. The total number of competing messages is thus P. To avoid the unwanted side-effect of MPI messages being cached, each repetition of a message is loaded/stored from/to a different location in a pre-allocated long buffer.

Algorithm 1 Bi-directional multi-pair benchmark

```
1: P, initial_size, max_size, increase_factor, num_repeats
2: size = initial_size
3: while size < max_size do
4:     if rank < P/2 then
5:         for i = 1, ..., num_repeats do
6:             MPI_Isend(send_data_buffer + i · size, size, rank + P/2)
7:             MPI_Irecv(recv_data_buffer + i · size, size, rank + P/2)
8:         end for
9:         MPI_Waitall()
10:    else
11:        for i = 1, ..., num_repeats do
12:            MPI_Irecv(recv_data_buffer + i · size, size, rank − P/2)
13:            MPI_Isend(send_data_buffer + i · size, size, rank − P/2)
14:        end for
15:        MPI_Waitall()
16:    end if
17:    size = increase_factor · size
18: end while
```

II.2.2 Example: Measuring $BW_{\mathrm{MP}}(N)$ on four machines

We have run the bi-directional multi-pair micro-benchmark (Algorithm 1) on four machines of dual-socket CPUs. The specific CPU types are: (1) ARM Cavium 32-core ThunderX2 CN9980; (2) ARM 64-core Kunpeng 920-6426; (3) Intel Xeon 26-core Gold-6230; and (4) AMD 64-core Epyc Rome-7742. OpenMPI v4.0.5 was used as the MPI installation. Compilation used GCC v10.1.0 with the `-O3 -march=armv8-a` options on the ThunderX2 and Kunpeng machines, and GCC v10.2.0 with the `-O3` option on the Intel Xeon and Epyc Rome machines. On each machine, we measure the time usage for a series of message sizes in the regime of the rendezvous protocol.[1] This is repeated for some chosen numbers of MPI processes. For each chosen value N, the series of time measurements (for the different message sizes) undergo a linear regression to recover the values of τ and $BW_{\mathrm{MP}}(N)$ that can be used in the max-rate model (II.2) or later in

[1] Time measurements of messages of size in the regime of the short or eager protocol will produce another set of τ and $BW_{\mathrm{MP}}(N)$ values.

Table II.1: Values of τ (in μs) and $BW_{\text{MP}}(N)$ (in GB/s), obtained from a linear regression of the time measurements of the bi-directional multi-pair micro-benchmark (Algorithm 1) on four dual-socket CPU machines. The tabulated $BW_{\text{MP}}(N)$ values will improve the accuracy of the max-rate model (II.2)-(II.3) for intra- and inter-socket communication.

CPU type	Level	τ	BW_{SP}	$BW_{\text{MP}}(2)$	$BW_{\text{MP}}(4)$	$BW_{\text{MP}}(8)$	$BW_{\text{MP}}(16)$	$BW_{\max}$
ThunderX2	intra-socket	2.3	7.5	14.6	25.5	32.5	45.0	54.0
CN9980	inter-socket	4.4	6.5	13.7	17.8	18.3	19.7	22.7
Kunpeng	intra-socket	3.8	5.0	10.5	14.8	17.0	19.7	53.7
920-6426	inter-socket	5.1	4.6	9.1	10.4	13.3	18.6	49.5
Intel Xeon	intra-socket	1.6	10.0	15.9	23.8	31.7	42.7	42.7
Gold 6230	inter-socket	2.7	8.1	15.6	22.5	27.5	29.4	29.4
AMD Epyc	intra-socket	1.7	10.2	16.8	17.6	19.2	23.4	51.0
7742 Rome	inter-socket	2.9	5.3	8.7	11.1	12.2	13.0	30.3

our new models to be introduced in Section II.3. On a given machine and for a specific communication level, the recovered τ values are very similar for the different choices of N. So we have consistently used the τ value associated with $N = 2$ in Table II.1 and later experiments. We remark that the models and experiments to be presented in this paper target message sizes in the regime of the rendezvous protocol. Our modeling approach remains the same for the other protocols.

The measurement-determined $BW_{\text{MP}}(N)$ values for all the four machines are summarized in Table II.1, where we also distinguish the scenarios of intra-socket and inter-socket. The former means that all sender-receiver pairs are placed on the same socket, whereas the latter means that each pair is split across two sockets. In this table, and also in the remainder of the paper, the value N denotes the number of active MPI processes per socket that concurrently receive incoming messages, i.e., running the micro-benchmark of Algorithm 1 with $P = N$ for the intra-socket measurements and $P = 2N$ for the inter-socket measurements. The reason for this definition of N is because the new performance models to be introduced in Section II.3 consider a socket as the "base unit" when modeling intra- and inter-socket communications. The $BW_{\max}$ values in Table II.1 are obtained from running the same number of MPI processes per socket as the CPU cores. The intra-socket BW_{SP} value is obtained by running two MPI processes on just one socket, with only one uni-directional message.

One clear observation from Table II.1 is that the τ measurements associated with the intra-socket and inter-socket cases confirm that intra-socket MPI communication is faster than the inter-socket counterpart. Another important observation is that the intra- and inter-socket $BW_{\text{MP}}(N)$ values do not follow the formula (II.3) of the max-rate model, clearly demonstrated in Fig. II.1. These tabulated $BW_{\text{MP}}(N)$ values will be heavily used in our new models of Section II.3 that are capable of handling varying numbers of incoming/outgoing messages per MPI process, non-uniform size per message, and the interaction

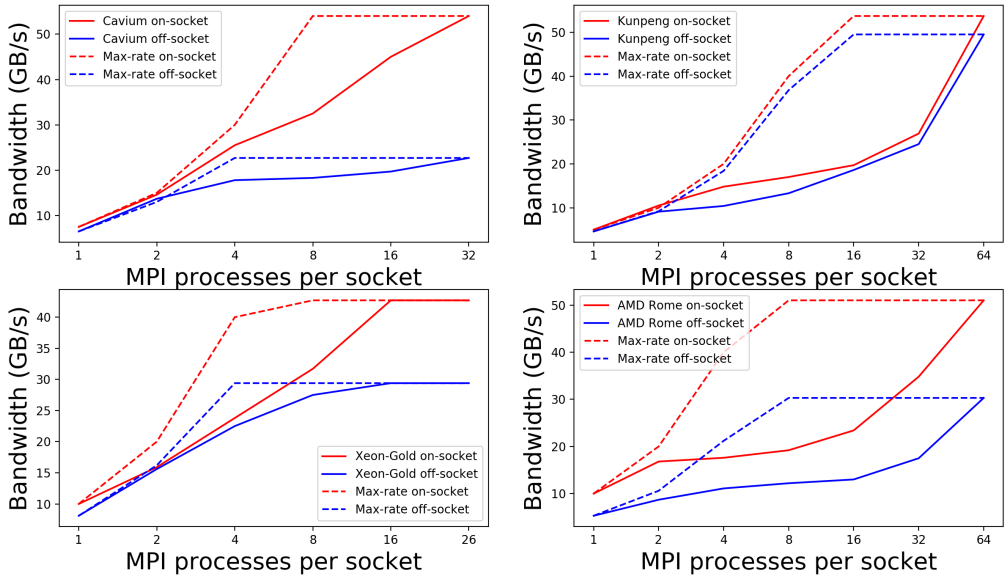


Figure II.1: Comparison between measurement-pinpointed $BW_{\mathrm{MP}}(N)$ values (solid curves) and those suggested by the formula (II.3) of the max-rate model (dashed curves). The measurements are obtained on four dual-socket CPU machines: ARM Cavium ThunderX2 (top-left), ARM Kunpeng (top-right), Intel Xeon-Gold (bottom-left) and AMD Epyc Rome (bottom-right).

between intra- and inter-socket traffic.

II.3 New performance models of many-pair, point-to-point communication

Apart from replacing the max-rate formula (II.3) with tabulated $BW_{\mathrm{MP}}(N)$ values that are determined by the new micro-benchmark of Algorithm 1, another improvement is to more accurately model the bandwidth contention when the competing MPI processes receive largely different volumes. We will thus develop in this section a new modeling strategy that detailedly quantify the *per-process* time usage for the cases where the message size is non-uniform and/or the number of neighbors per process varies. A more general situation is that many MPI messages of various sizes are concurrently communicated over multiple levels of a heterogeneous network. In total three new models of increasing complexity will be introduced.

Algorithm 2 Many-pair, point-to-point communication

1: On each process i: $\{\text{neigh}_j^{\text{in}}\}_{j=0}^{M_i^{\text{in}}-1}$, $\{\text{Rsize}_j\}_{j=0}^{M_i^{\text{in}}-1}$, $\{\text{neigh}_j^{\text{out}}\}_{j=0}^{M_i^{\text{out}}-1}$, $\{\text{Ssize}_j\}_{j=0}^{M_i^{\text{out}}-1}$
2: **for** $j = 0, \ldots, M_i^{\text{out}} - 1$ **do**
3: MPI_Isend(Sbuffer$_j$, Ssize$_j$, neigh$_j^{\text{out}}$, send_req$_j$)
4: **end for**
5: **for** $j = 0, \ldots, M_i^{\text{in}} - 1$ **do**
6: MPI_Irecv(Rbuffer$_j$, Rsize$_j$, neigh$_j^{\text{in}}$, recv_req$_j$)
7: **end for**
8: **for** $j = 0, \ldots, M_i^{\text{out}} - 1$ **do**
9: MPI_Wait(send_req$_j$)
10: **end for**
11: **for** $j = 0, \ldots, M_i^{\text{in}} - 1$ **do**
12: MPI_Wait(recv_req$_j$)
13: **end for**

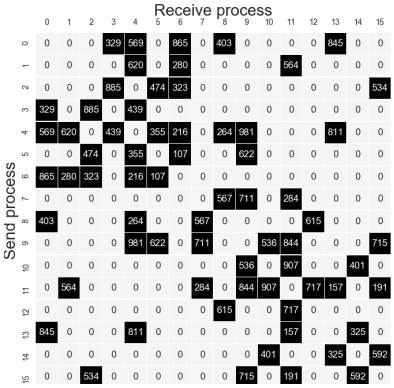


Figure II.2: An example communication matrix involving 16 MPI processes. Each black box represents an MPI message, and the number inside the box is the message size.

II.3.1 General many-pair, point-to-point communication

First let us define the general situation of many-pair, point-to-point MPI communication in Algorithm 2. More specifically, MPI process with rank i will simultaneously receive M_i^{in} messages from M_i^{in} different neighbors. Each process thus has a list of neighbor identities $\{\text{neigh}_j^{\text{in}}\}_{j=0}^{M_i^{\text{in}}-1}$. The neighbors can be of mixed types, i.e., some may reside on the same socket as process i, others may reside on a different socket but inside the same compute node, whereas the rest may reside on other compute nodes. Each process is also assumed to send M_i^{out} outgoing messages to M_i^{out} neighbors (can be different from the M_i^{in} "inward" neighbors). All the messages can be of varying sizes, and the sizes of the "inward" and "outward" messages do not have to match. For this purpose, we have for each process two sets of message sizes $\{\text{Rsize}_j\}_{j=0}^{M_i^{\text{in}}-1}$ and $\{\text{Ssize}_j\}_{j=0}^{M_i^{\text{out}}-1}$. An example of the resulting sparse *communication matrix* is Fig. II.2, which involves 16 processes. It should be remarked that Algorithm 2 describes irregular point-to-point MPI communications that frequently arise in scientific computations. One such example can be found in the `copyOwnerToAll` function of the DUNE software framework [3, 10, 24], in connection with parallelizing sparse matrix-vector multiplications.

II.3.2 A staircase modeling strategy

Before developing new performance models of increasing complexity for quantifying the per-process time usage associated with Algorithm 2, we need to first introduce a "staircase" modeling strategy. The purpose is to quantitatively describe the *dynamically* changing scenarios of competition, i.e., the number of competing processes and the available communication bandwidth are both dynamic. The following principles are associated with this modeling strategy:

1. The time needed by process i to complete all its communication tasks is determined by either how soon it can finish receiving all its M_i^{in} incoming messages, or how soon all its M_i^{out} outgoing messages are received at the destinations. The maximum of the two time usages will apply. The reason for also considering the delivery of the outgoing messages at the destinations is motivated by the most common situation that uses the rendezvous protocol without intermediate buffering, where experiments show that a sending process cannot complete until all its outgoing messages are delivered.

2. The "expected" order of which the M_i^{in} incoming messages are received by process i is determined by the sizes of these messages. (The actual order is stochastic, thus not modelable.) The smallest incoming message completes first, followed by the second smallest message, and so on. The completion time points for the different incoming messages can be estimated using a *staircase* strategy, which will be detailed by formula (II.9). If the M_i^{in} incoming messages are of the same size, they are considered to complete simultaneously, due to fairness.

3. When multiple receiving processes, each with one or more incoming messages, compete for the same network connection, another staircase principle applies as will be described in formula (II.5). The process with the least amount of incoming communication will complete first, letting the remaining processes continue with their remaining communication. This procedure repeats itself until all the processes are completed. At all times, the bandwidth is shared evenly between the still active processes, while the actually available bandwidth can depend on the number of concurrently competing processes.

II.3.3 Model for one-level, single-neighbor, non-uniform message size

The staircase modeling strategy will be first applied to the scenario of each MPI process having exactly one incoming message and one outgoing message. The messages sent/received by the different processes can be of various sizes. For this simple scenario we consider that all the MPI processes communicate on the same level, i.e., either all intra-socket, or all inter-socket, or all inter-node.

To model the per-process time usage, we first group the MPI processes that share the same bandwidth, i.e., the processes that reside on the same socket for

the level of intra-socket communication, or the processes on one socket that share the same inter-socket bandwidth, or the processes that reside in one compute node that share the same inter-node bandwidth. We let N denote the number of processes in a group. The bandwidth under contention is characterized by a set of pre-tabulated values of BW_{SP}, $BW_{\mathrm{MP}}(2)$,..., $BW_{\mathrm{MP}}(N)$, as well as a latency constant τ.

Modeling of the per-process time usage will start with sorting the N processes of each group with respect to the size of the per-process single incoming message: $s_0 \leq s_1 \leq \ldots \leq s_{N-1}$. Then, the per-process time usage T_i can be estimated as a "staircase", more specifically,

$$t_0^{\mathrm{recv}} = \frac{N \cdot s_0}{BW_{\mathrm{MP}}(N)},$$

$$t_i^{\mathrm{recv}} = t_{i-1}^{\mathrm{recv}} + \frac{(N-i)\cdot(s_i - s_{i-1})}{BW_{\mathrm{MP}}(N-i)}, \quad i = 1, 2, \ldots, N-1, \tag{II.5}$$

$$T_i = \tau + \max\left(t_i^{\mathrm{recv}}, t_i^{\mathrm{send}}\right). \tag{II.6}$$

The recursive formula in (II.5) accounts for a decreasing degree of bandwidth contention in the form of a staircase, when more and more processes complete. Also, we have used $BW_{\mathrm{MP}}(1) = BW_{\mathrm{SP}}$. In (II.6), t_i^{send} denotes the time needed for the outgoing message of process i to be delivered on the destination process with rank $k = \mathrm{neigh}_i^{\mathrm{out}}$ (see Algorithm 2), which is considered the same as the receiving time t_k^{recv} on process k. We note that the time usage model in (II.5)-(II.6) is only valid for single-message per-process communication, but it can be extended to account for multiple message sources and destinations per process. This will be covered in Section II.3.4 below.

Verification example

As the verification experiments for this new model, as well as for verifying the other new performance models later, we have always run a benchmark code that implements Algorithm 2. (The total number of MPI processes, the number of neighbors per process and the sizes of the messages may differ from experiment to experiment.) More specifically, each experiment executes Algorithm 2 ten times and the average time usage per iteration is recorded *individually* for each MPI process.

We have tested the new model (II.5)-(II.6) by quantifying the per-process time usage of 6 MPI processes running on one ARM Cavium ThunderX2 processor (all the messages are intra-socket). The 6 processes form three sender-receiver pairs, where each pair exchanges the same amount of data in the two opposite directions. However, the three pairs simultaneously work with three different message sizes: $\frac{s}{2}$, s and $2s$, with s ranging between 2 bytes and 2 MB. The actual time usages for the three pairs are plotted against the model estimates in Fig. II.3.

In Fig. II.3 we can see that the model estimates agree very well with the actual time measurements for most of the message sizes. The model (II.5)-(II.6)

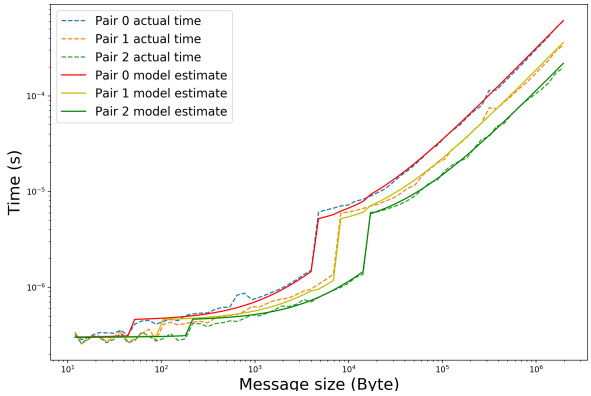


Figure II.3: Single-neighbor, intra-socket messages on an ARM Cavium ThunderX2 CPU. The experiment involves six processes exchanging messages bi-directionally in three pairs. The measured times and model estimates are plotted with dotted and solid lines against the "middle" message size s. Pairs 0, 1 and 2 exchange, respectively, $2s$, s and $\frac{s}{2}$ bytes of data.

is able to estimate the time usage of each individual process. The two processes of each pair have the same time measurements and estimates, thus we only see three sets of measurement-estimate curves. For each set, we can also observe two clear jumps in the time measurements and estimates, each corresponding to a change in the message protocol, first between the short and eager protocols, then between eager and rendezvous. For example, the eager-rendezvous protocol threshold at 8 KB is clearly visible for the mid-sized pair in the plot.

II.3.4 Model for one-level, multi-neighbor, non-uniform message size

Now let us consider a more general situation where each process communicates with multiple neighbors. For this situation, we modify the recursive formula in (II.5) by replacing the single-message size s_i with a per-process incoming message volume $V_i = \sum_{j=0}^{M_i^{\text{in}}-1} s_{i,j}$, where M_i^{in} denotes the number of "inward" neighbors for process i, and $s_{i,j}$ denotes the size per incoming message. The MPI processes are organized into groups as in the previous model. Again, N denotes the number of processes in a group, and the processes inside the group is now sorted with respect to $V_0 \leq V_1 \leq \ldots \leq V_{N-1}$. The time usage estimate in (II.5) for single-neighbor, non-uniform message size communication can now be extended to model the multi-neighbor situation:

$$t_0^{\text{recv}} = \frac{N \cdot V_0}{BW_{\text{MP}}(N)},$$

$$t_i^{\text{recv}} = t_{i-1}^{\text{recv}} + \frac{(N-i)\cdot(V_i - V_{i-1})}{BW_{\text{MP}}(N-i)}, \quad i = 1, 2, \ldots, N-1, \tag{II.7}$$

$$T_i = M_i^{\text{in}} \cdot \tau + \max\left(t_i^{\text{recv}}, t_{i,0}^{\text{send}}, t_{i,1}^{\text{send}}, \ldots, t_{i,M_i^{\text{out}}-1}^{\text{send}}\right). \tag{II.8}$$

Table II.2: Information on the verification examples presented in Fig. II.4. The columns display the communication type, number of MPI processes, total number of messages, maximum/minimum messages per process, total/maximum/minimum(non-zero) per-process communication volume, and the total relative prediction errors for, respectively, the staircase model and the extended max-rate model (II.10).

Communication type	#procs	#msgs	max/min	total/max/min volume	Staircase error	Max-rate error
intra-socket	64	192	3/3	16.5MB/0.5MB/0.05MB	11.5%	26.0%
inter-socket	64	218	7/0	19.4MB/1.0MB/0.04MB	13.0%	36.8%
inter-node	256	3958	58/0	461.7MB/4.3MB/0.3KB	12.9%	13.0%

In (II.8), $t_{i,j}^{\text{send}}$ denotes the delivery time needed by each outgoing message, and it equals the time needed by destination process (with rank $\text{neigh}_{i,j}^{\text{out}}$) to receive this message. Since each process can have multiple incoming messages, we thus need to "theoretically" pinpoint the individual time points at which the different messages are received on each destination process. Take for instance process i. It is the destination for M_i^{in} incoming messages, and the total receiving time (without the latency overhead) has been calculated as t_i^{recv} using (II.7). The following recursive formula, which is again based on a staircase principle, will be used to pinpoint the individual time points at which the M_i^{in} incoming messages are received:

$$
\begin{aligned}
t_{i,0}^{\text{recv}} &= \frac{M_i^{\text{in}} \cdot s_{i,0}}{V_i} \cdot t_i^{\text{recv}}, \\
t_{i,j}^{\text{recv}} &= t_{i,j-1}^{\text{recv}} + \frac{(M_i^{\text{in}} - j) \cdot (s_{i,j} - s_{i,j-1})}{V_i} \cdot t_i^{\text{recv}}, \quad j = 1, 2, \ldots, M_i^{\text{in}} - 1.
\end{aligned} \tag{II.9}
$$

In the above recursive formula, we have sorted the M_i^{in} incoming messages for process i with respect to the message sizes $s_{i,0} \leq s_{i,1} \leq \ldots \leq s_{i,M_i^{\text{in}}-1}$, where $\sum_{j=0}^{M_i^{\text{in}}-1} s_{i,j} = V_i$. We remark that the completion time point for the largest incoming message, which is theoretically expected to finish last, equals the total receiving time needed by process i, i.e., $t_{i,M_i^{\text{in}}-1}^{\text{recv}} = t_i^{\text{recv}}$.

Verification examples

To demonstrate the single-level, multi-neighbor model (II.7)-(II.9), we have conducted three experiments, each consisting exclusively of intra-socket, inter-socket or inter-node traffic. All the results were obtained on a cluster of Epyc Rome CPUs, described in Section II.2.2. For the intra-socket experiment, a synthetic communication pattern with 64 MPI processes, spread over two sockets, has been used. As shown in Fig. II.4b, each process sends and receives 3 intra-socket messages of different sizes. The per-process time estimates are plotted against the actual time measurements in Fig. II.4a. For comparison, the plot also includes per-process estimates produced by the extended max-rate model (II.4),

marked as purple dots. More specifically, we have used the following variation of (II.4):

$$T_i = M_i \cdot \tau + \max\left(\frac{\min\left(V_{\text{total}}, NV_i\right)}{BW_{\max}}, \frac{V_i}{BW_{\text{SP}}}\right). \tag{II.10}$$

That is, the total messaging volume s_{total} in (II.4) is replaced by $\min\left(V_{\text{total}}, NV_i\right)$. The maximum per-process volume $s_{\max}$ is replaced by per-process volume V_i. Note that (II.10) is the same as (II.4) for the slowest process.

The inter-socket experiment uses 64 MPI processes (also spread over two sockets) that exclusively send and receive inter-socket messages, see Fig. II.4d. The number of neighbors per MPI process and the message sizes arise from partitioning a realistic unstructured computational mesh into 64 pieces, where the maximum number of neighbors per process is 7, and two processes have no neighbors. The message sizes also vary considerably. The actual and estimated per-process times by (II.7)-(II.9) are displayed in Fig. II.4c. Max-rate estimates are also included in Fig. II.4c.

The inter-node experiment involves 256 MPI processes spread over four nodes, and the results and inter-node communication pattern are presented in Fig. II.4e-II.4f. This example also arises from a realistic case of partitioning an unstructured computational mesh. Every MPI process only sends and receives inter-node messages, where the number of neighbors per process and the message sizes vary considerably. (The maximum number of neighbors per process is 58 whereas the minimum is 0.) The extended max-rate model (II.4) has been used to estimate the slowest process time usage per node. Per-process estimates are not included to prevent cluttering the plot.

To quantify the accuracy of our model (II.7)-(II.9), we calculate a total relative error defined as follows:

$$\text{Total relative error} = \frac{\sum_{i=0}^{P-1} |T_i^{\text{actual}} - T_i|}{\sum_{i=0}^{P-1} T_i^{\text{actual}}}, \tag{II.11}$$

where T_i^{actual} denotes the measured actual time usage on process i and T_i denotes the per-process model estimate. Using this error definition, we have calculated the modeling error for the experiments shown in Fig. II.4 to be 11.5% for the intra-socket experiment, 13.0% for the inter-socket experiment and 12.8% for the inter-node experiment, see Table II.2.

As shown in Fig. II.4a-II.4c, the extended max-rate model under-estimates the per-process time for the intra- and inter-socket scenarios. This is not surprising because it incorrectly models the bandwidth contention. Thus our one-level model (II.7)-(II.9) gives better accuracy than the extended max-rate models for intra- and inter-socket traffic. For the third experiment, shown in Fig. II.4e, our model has approximately the same accuracy as the extended max-rate model. This result is expected because two competing MPI processes can already saturate the maximum inter-node bandwidth on this system. A more elaborate model of the bandwidth contention thus does not pay off.

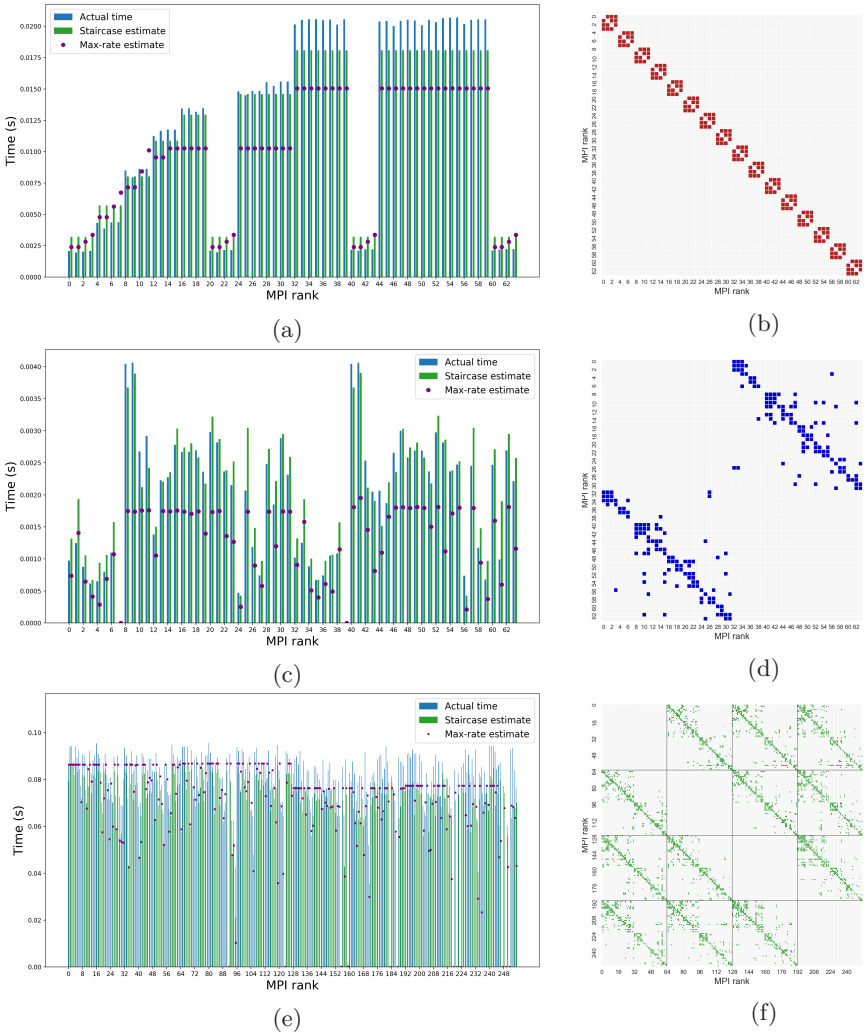


Figure II.4: Actual time measurement (blue bar), staircase model estimate (green bar) and max-rate estimate (purple dots) for each process on Epyc Rome CPUs. The intra-socket experiment (a) uses a synthetic communication pattern shown in (b), while the inter-socket (c) and inter-node (e) results are obtained using realistic communication patterns shown in (d) and (f), respectively.

II.3.5 Model for mixing intra- and inter-socket messages

The two new models (II.5)-(II.6) and (II.7)-(II.9) can be used to quantify the per-process time usage when all the point-to-point communication happens on one level. However, communications that concurrently take place on different levels can affect each other. We will thus develop a third model to handle a

mixture of intra-socket and inter-socket (intra-node) messages. This is important in practice because point-to-point MPI communications on these two levels are particularly subject to the bandwidth contention, which is in essence the contention for the shared memory bandwidth.

The main idea is that the available communication bandwidth, which is to be shared among N competing processes, is neither the pure intra-socket $BW_{\mathrm{MP}}^{\mathrm{on}}(N)$ value nor the pure inter-socket $BW_{\mathrm{MP}}^{\mathrm{off}}(N)$ value. For process i, its achievable bandwidth will depend on the ratio between these two traffic types, more specifically,

$$BW_{\mathrm{MP}}^{\mathrm{mix}}(N,\theta_i) = \frac{\theta_i}{N}\cdot BW_{\mathrm{MP}}^{\mathrm{on}}(N) + \frac{1-\theta_i}{N}\cdot BW_{\mathrm{MP}}^{\mathrm{off}}(N),$$
$$\theta_i = \frac{V_i^{\mathrm{on}}}{V_i^{\mathrm{on}} + V_i^{\mathrm{off}}}, \tag{II.12}$$

where V_i^{on} and V_i^{off} denote, respectively, the total intra- and inter-socket incoming volumes on process i.

To model the per-process receiving time usage t_i^{recv}, when intra- and inter-socket traffic is mixed, we start with dividing all the MPI processes into groups based on which socket they reside on. As in the two preceding models, N denotes the number of processes in a group. The challenge now is that we can no longer easily see beforehand the exact order in which the processes will finish receiving their incoming traffic. This is due to a more dynamic competition for the available bandwidth. We will thus adopt a modeling algorithm that is based on yet another staircase strategy consisting of N steps, where in each step we can find out, "on-the-fly", which process will be the next one to finish.

For step $k = 0, 1, \ldots, N-1$:

Find an unfinished process q that gives

$$t_{\mathrm{step}}^{k} = \min_i \frac{V_i^{\mathrm{on}} - V_i^{\mathrm{on,recv}} + V_i^{\mathrm{off}} - V_i^{\mathrm{off,recv}}}{BW_{\mathrm{MP}}^{\mathrm{mix}}(N-k,\theta_i)}; \tag{II.13}$$

Compute $t_q^{\mathrm{recv}} = \sum_{\ell=0}^{k} t_{\mathrm{step}}^{\ell}$; (process q finishes after step k) (II.14)

Compute $T_q = M_q^{\mathrm{in,on}}\cdot\tau^{\mathrm{on}} + M_q^{\mathrm{in,off}}\cdot\tau^{\mathrm{off}} +$
$$\max\left(t_q^{\mathrm{recv}}, t_{q,0}^{\mathrm{send}}, t_{q,1}^{\mathrm{send}}, \ldots, t_{q,M_q^{\mathrm{out}}-1}^{\mathrm{send}}\right); \tag{II.15}$$

For all unfinished processes:

$$V_i^{\mathrm{on,recv}} += t_{\mathrm{step}}^{k}\cdot\theta_i\cdot BW_{\mathrm{MP}}^{\mathrm{mix}}(N-k,\theta_i); \tag{II.16}$$
$$V_i^{\mathrm{off,recv}} += t_{\mathrm{step}}^{k}\cdot(1-\theta_i)\cdot BW_{\mathrm{MP}}^{\mathrm{mix}}(N-k,\theta_i). \tag{II.17}$$

In the above model, we have used $V_i^{\mathrm{on,recv}}$ and $V_i^{\mathrm{off,recv}}$ to record the accumulated volumes of intra- and inter-socket data received so far on process i.

The values of $V_i^{\text{on,recv}}$ and $V_i^{\text{off,recv}}$ are increased during each step using (II.16)-(II.17). After each step, a process q will finish, as expressed by (II.13)-(II.14). On each process, its individual ratio θ_i remains the same throughout the entire process. A source code that implements the above mixed-level model can be found in the appendix.

Verification examples

We illustrate the mixed intra/inter-socket model (II.12)-(II.17) with two simple examples, one using the dual-socket ThunderX2 machine and the other using the dual-socket Kunpeng machine (see Table II.1 in Section II.2.2). In the first experiment we use in total 48 processes, where 32 processes (16 on each socket) send and receive intra-socket messages and the remaining 16 processes (8 on each socket) send and receive inter-socket messages. The results are presented in Fig. II.5a, and the communication pattern is presented in Fig. II.5b. The plot in Fig. II.5a includes actual per-process time measurements as blue bars and the staircase model estimate as green bars. Notice that the mixed intra/inter-socket model correctly predicts that the off-socket processes complete after the on-socket processes. Using the error metric defined in (II.11), the total relative error is found at 6.6% for the staircase estimate.

In the second experiment, we use 64 processes spread over two Kunpeng sockets. The process connectivity is the same as shown in Fig. II.4b, but this time we deliberately "reshuffle" the process mapping (by the OpenMPI option `-map-by socket`) to incur a mixture of intra- and inter-socket communication. The per-process message sizes and types are displayed in Fig. II.5d. The actual time measurements and staircase model estimates are displayed in Fig. II.5c. The resulting total relative error is 3.6%.

II.4 Realistic tests

So far, we have mostly used synthetic experiments, with the only exception being those shown in Fig. II.4c-II.4f. In this section, we will apply the new performance models to realistic communication patterns.

II.4.1 Mixing intra-node and inter-node messages

The most general point-to-point MPI communication can simultaneously take place at all the levels: intra-socket, inter-socket and inter-node. Our approach is to use the one-level model (II.7)-(II.9) for estimating the inter-node cost per process as $T_i^{\text{inter-node}}$, whereas the mixed-level model (II.12)-(II.17) is used for estimating the mixed intra/inter-socket cost as $T_i^{\text{intra-node}}$. The total time per process is given by

$$T_i^{\text{total}} = T_i^{\text{inter-node}} + T_i^{\text{intra-node}}. \tag{II.18}$$

The above model of total time cost assumes that the intra-node and inter-node communication tasks cannot proceed simultaneously, whereas the intra-socket

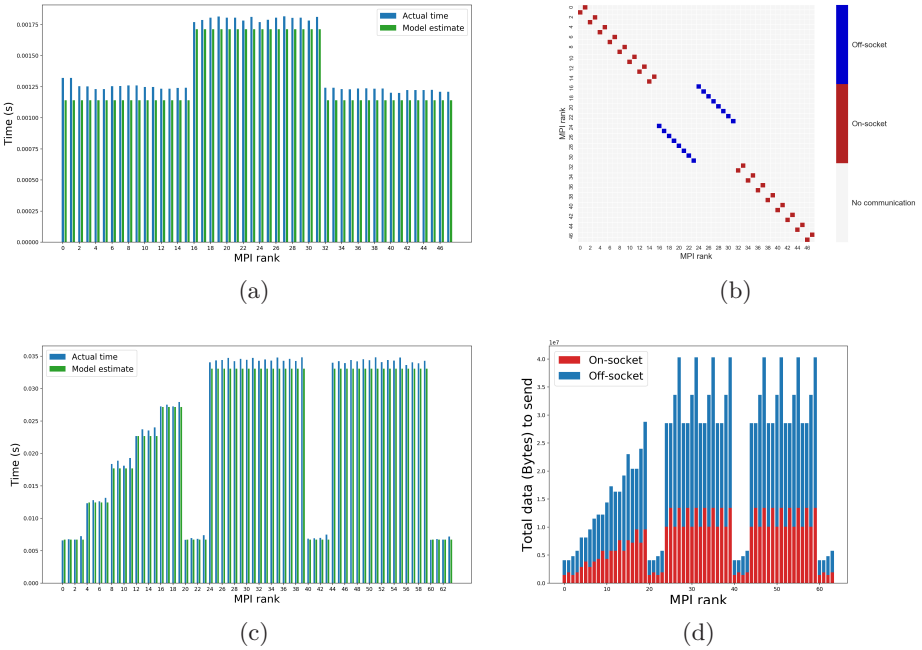


Figure II.5: Plot (a) shows the actual time (blue) and model estimate (green) for an experiment on the dual-socket ThunderX2 machine using 48 processes (evenly spread over two sockets), with the communication pattern shown in plot (b). Plot (c) shows the actual time and model estimate for an experiment on the dual-socket Kunpeng machine using 64 processes (evenly spread over two sockets). In this experiment each process sends and receives three messages of a varying size. The amount of on- and off-socket traffic each process receives is displayed in plot (d).

and inter-socket messages can compete for the same memory bandwidth (while dynamically affecting each other). We remark that summing up the inter-node costs with the intra-node costs is the same approach as adopted in [6].

II.4.2 Realistic communication patterns

For realistic communication patterns, we have adopted some examples of partitioning of realistic 3D unstructured meshes that have been used for reservoir simulations [19, 20, 26]. This is in connection with an MPI parallelization of a reservoir simulator, where Algorithm 2 needs to be repeatedly executed when numerically solving the involved partial differential equations. We refer the reader to [26] for the details. Given a decomposition of the mesh, based on a specified number of MPI processes, we can create the corresponding communication pattern of the irregular point-to-point MPI communication operations. Each

MPI message is classified as on-socket, off-socket or inter-node.

II.4.3 Small-scale experiments

To test the performance model (II.18) on realistic communication patterns, we compare the per-process execution times of Algorithm 2 with the model estimates when running different numbers of MPI processes on four computing platforms: ThunderX2, Kunpeng, Xeon-Gold and Epyc Rome (see Table II.1 in Section II.2.2), using up to four compute nodes. One realistic example of per-process communication volumes can be seen in Fig. II.6.

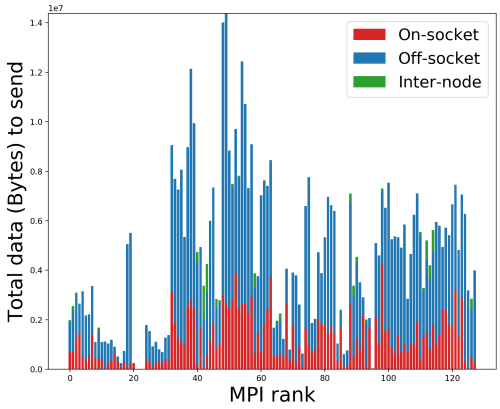


Figure II.6: One example of realistic per-process communication volumes when using 128 MPI processes spread over two compute nodes (with four sockets in total).

Table II.3 shows that the performance model (II.18) achieves good prediction results. The total relative error ranges from 7.5% to 18.9% on ThunderX2 and Kunpeng, from 12.2% to 15.6% on Xeon-Gold, and from 9.3% to 14.3% on Epyc Rome. A more detailed comparison for two ThunderX2 and two Kunpeng nodes is presented in Fig. II.7. Here, the per-process execution time and model estimate are displayed in bar plots. In Fig. II.7a, execution times and model estimates for 128 processes on two ThunderX2 nodes are displayed, while the plot in Fig. II.7b displays the timing results and model estimates for 256 processes on two Kunpeng nodes. We can observe good correspondence between the model estimates and execution times for most cases.

II.4.4 Large-scale experiments

We have also tested our model on the Betzy supercomputer [4], which is a BullSequana XH2000 system with in toal 1344 compute nodes. Each compute node has two 64-core AMD Epyc Rome CPUs for a total of 172,032 CPU cores. The system uses an Infiniband HDR 100 network connected in a Dragonfly+

Table II.3: Experiments with realistic communication patterns on ThunderX2, Kunpeng, Xeon-Gold and Epyc Rome. The results are obtained on one, two and four nodes, using up to 256 processes on ThunderX2, up to 512 processes on Kunpeng and Epyc Rome, and up to 208 processes on Xeon-Gold. The table displays the total number of messages ($\sum$M), max per-process messages, total communication volume (in MBs), the on-socket (intra-socket), off-socket (inter-socket) and inter-node percentages of the total volume, and the total relative prediction error.

ThunderX2								Kunpeng							
procs	$\sum$M	maxM	vol	on	off	inter	error	procs	$\sum$M	maxM	vol	on	off	inter	error
32	154	9	17	34%	66%	0%	0.142	32	154	9	17	34%	66%	0%	0.097
64	374	11	30	36%	64%	0%	0.157	64	374	11	30	36%	64%	0%	0.085
128	1180	22	68	27%	70%	3%	0.158	128	1180	22	68	27%	73%	0%	0.115
256	4022	33	248	35%	62%	3%	0.189	256	4022	33	248	36%	63%	1%	0.075
								512	14650	52	403	43%	56%	1%	0.110
Xeon-Gold								Epyc Rome							
procs	$\sum$M	maxM	vol	on	off	inter	error	procs	$\sum$M	maxM	vol	on	off	inter	error
26	132	10	13	24%	76%	0%	0.152	64	1394	46	153	37%	63%	0%	0.093
52	304	12	22	37%	63%	0%	0.149	128	2944	91	280	38%	62%	0%	0.143
104	760	14	41	29%	69%	2%	0.122	256	5118	104	544	36%	61%	3%	0.133
208	2446	24	123	25%	73%	2%	0.156	512	13552	104	1116	35%	57%	8%	0.141

topology [25], which consists of 14 groups each with 96 nodes. The Dragonfly+ topology improves scalability over the original Dragonfly, while maintaining the cost benefits when compared with the Fat-tree topology [15]. The Dragonfly+ topology also provides full bisection bandwidth for intra-group communications, but inter-group communications are oversubscribed and require the use of non-minimal routing to achieve good performance. This means that the Betzy system will see excellent inter-node communication for MPI processes running inside a single group, but performance may suffer with regards to both bandwidth and latency when the MPI processes are spread across two or more groups, depending on the overall network load.

The prediction accuracy of our model (II.18) has been tested on Betzy using communication patterns that include a larger proportion of inter-node messages arising from partitioning a large reservoir mesh that is ten times larger than that used for the experiments on ThunderX2, Kunpeng and Xeon-Gold in Section II.4.3. Fig. II.8 shows a breakdown of the per-node communication volumes for this large case when being partitioned into 1024 and 8192 subdomains, using 8 and 64 compute nodes on Betzy, respectively.

From Fig. II.8 we can observe that the communication patterns attained by partitioning the large mesh result in a majority of on-socket communication. However, we also notice that there is a significant amount of off-socket and inter-node communication. The model prediction accuracy for our large-scale Epyc Rome experiments is presented in Table II.4. In this table, we include results attained on 4 to 64 nodes. In the scenario of 64 nodes, the MPI processes are distributed (unevenly) across two Dragonfly+ groups in order to enforce non-uniform network connectivity and test performance accuracy when both inter- and intra-group communications are present. The total relative prediction error displayed in Table II.4 ranges from 12.2% to 20.9%, with the error being

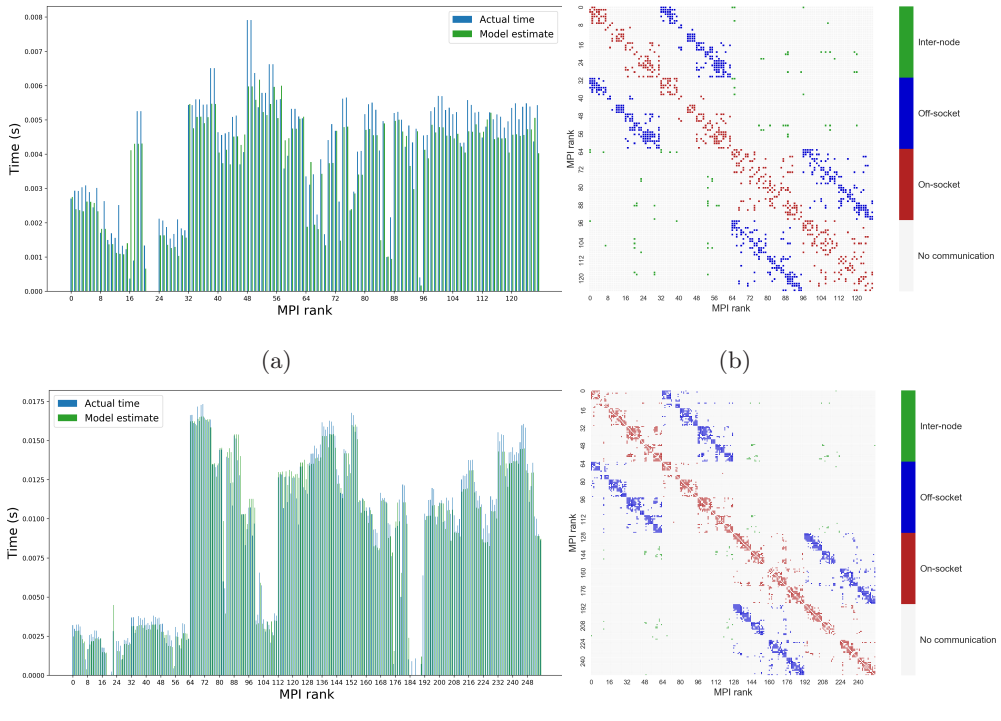


Figure II.7: The per-process execution times (blue bars) and model estimates (green bars) for two realistic communication patterns on, respectively, two ThunderX2 nodes (a)-(b) and two Kunpeng nodes (c)-(d).

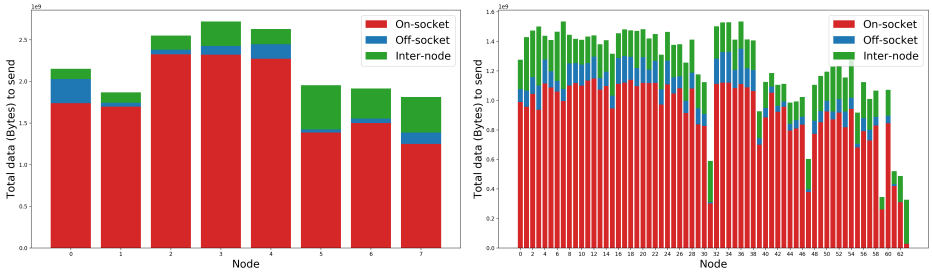


Figure II.8: Per-node communication volumes when partitioning a realistic reservoir mesh into 1024 and 8192 subdomains on 8 and 64 AMD Epyc Rome nodes.

16.2% for 8192 processes on 64 nodes.

Table II.4: Study of per-process MPI overhead with realistic communication patterns on a cluster of AMD Epyc Rome nodes. The rows show, respectively, the number of MPI processes (nodes), total number of messages, maximum per-process number of messages, total communication volume, maximum per-process communication volume, minimum per-process volume, overall intra-socket communication volume percentage, inter-socket volume percentage, inter-node volume percentage, and the total relative prediction error for the staircase estimate.

#procs (nodes)	512(4)	640(5)	768(6)	896(7)	1024(8)	2048(16)	4096(32)	8192(64)
#messages	13552	22598	29866	41844	50756	268036	1050652	2744632
Max messages	104	125	164	182	210	375	733	1235
Total vol (MB)	1116	1353	1524	1923	2098	4801	7541	9382
Max vol (MB)	6.68	8.15	7.44	6.62	5.96	4.15	2.82	1.86
Min vol (KB)	0.88	0.59	1.17	5.27	0.59	0.29	0.59	0.29
Intra-socket	83.9%	71.6%	72.4%	74.9%	82.3%	84.3%	81.9%	74.8%
Inter-socket	8.9%	7.7%	10.8%	7.0%	5.1%	3.9%	4.5%	8.0%
Inter-node	7.2%	20.7%	16.8%	18.1%	12.6%	11.8%	13.6%	17.2%
Prediction error	0.145	0.209	0.181	0.158	0.178	0.140	0.122	0.162

II.5 Related work

For a long time, the de facto standard for analyzing the communication performance of parallel systems was the postal model [2, 12]. This two-parameter model has the benefit of being very simple, but its performance estimates are only accurate for fix sized, single messages on massively parallel processing (MPP) systems. These traditional MPP systems were homogeneous and there was only inter-node communication (i.e., no intra-node communication). The limitation of single messages was addressed in the more recent max-rate model [11], which extends the postal model to support N concurrent messages from N send-receive process pairs. Improvements of the max-rate model were introduced in [5] to clearly distinguish between inter-node, intra-node and intra-socket messages and to estimate network contention. Nevertheless, the max-rate model and its successor still lack the support for variable sized messages, plus the weakness of adopting an over-simplified "roofline" model for quantifying the aggregate bandwidth that is available to multiple sender-receiver pairs. In comparison, our work in the present paper adopts accurate aggregate bandwidth values that are measured by a simple bi-directional, multi-pair micro-benchmark, which was extended from the `osu_bibw` benchmark of MVAPICH [18]. The main difference between our work and the max-rate models [5, 11] is the adoption of a *staircase* modeling to quantify the per-process cost for the general cases where different MPI processes can have different numbers of neighbors and the messages are of different sizes. Apart from modeling point-to-point communications on a

single interconnection level, our models can specifically handle the dynamic competition between concurrent intra-socket and inter-socket messages.

Apart from the postal model and the max-rate models, another body of work on communication modeling contains the LogP family of models. This originated in the LogP model which was first proposed by Culler et al. in [2]. The LogP model adds communication overhead as an additional parameter to better estimemate the cost of sending and receiving messages, but is still constrained to fix sized, single messages and limited in accuracy for long messages. The latter was improved in the LogGP model [1] which extends the LogP model with support for linear modeling of long messages. Several other refinements also exist, including LogGPC [17] which adds parameters for network contention, LogGPS [14] which estimates synchronisation costs in the communications library, and LogGPO [8] which characterizes the overlap between communication and computation. A comprehensive survey of the different models was provided in [23]. The main drawback of the LogP-family models is the adoption of hard-to-estimate parameters, such as network contention and synchronisation costs, in addition to the lack of support for variable sized messages and heterogeneous levels of latency and bandwidth. In comparison, instead of detailing the different contributing terms to the overall latency/overhead, which is the main modeling principle adopted by the LogP family, we let a micro-benchmark quantify the overall latency/overhead and focus instead on modeling the dynamic competition between variable sized, concurrent messages.

The particular modeling challenges due to the wide use of shared-memory nodes in modern compute clusters, together with the resulting concurrent messages, were investigated in the τ-Lop model [21] and its extension [22]. Our work follows the same notion of *concurrent transfers* from [21] and the reasoning about the resulting contention among multiple MPI processes for the same bandwidth. However, our "staircase" modeling strategy for bandwidth contention is completely different that used in [21].

II.6 Concluding remarks

The basic assumption for the work presented in the current paper is that concurrent MPI messages that are transmitted over the same connection of a communication network should *fairly* share the aggregate bandwidth available. Moreover, when the messages are of different sizes, the principle of fairness will lead to that the shortest message should complete first, followed by the second shortest one, and so on. Such a dynamically changing situation of competition will, at the same time, lead to a dynamic change of the available bandwidth, which may not follow a simple profile (II.3) as suggested by the max-rate model. This has inspired the "staircase" modeling strategy that is heavily used in our new performance models.

In reality, the actual completion sequence of the different messages may indeed not follow the fairness principle. Various stochastic features will lead to different sequences even when an MPI program is repeated several times on

the same parallel system while using the same configuration. We consider it impractical to properly model such stochasticity. Instead, we adopt an "idealized" expectation based on fairness. Despite this simplification, numerical experiments reported in Sections II.3 and II.4 have shown a consistently good agreement between the model predictions and the actual time measurements. In particular, a realistic case that involves 8192 MPI processes and 2.7 million concurrent messages (see Table II.4) has achieved an average overall accuracy of 83.8%.

Overall, our "staircase" modeling strategy is manageably simple. The models from Sections II.3.4 and II.3.5 can be efficiently implemented such as shown in the appendix. As preparation work before using our new models, the bi-directional, multi-pair micro-benchmark (Algorithm 1) needs to be executed for different MPI process counts and for three situations: within a CPU socket, between a pair of CPU sockets (with shared memory), and between a pair of nodes in a cluster. The measured values of τ and BW_{MP} can be tabulated once and for all. To save the preparation work, the τ and BW_{MP} values for the short and eager protocols can be skipped, if the message sizes of interest are known to lie in the rendezvous regime.

In this paper, the possible costs that are associated with packing and unpacking MPI messages are not included. Such extra costs are necessary if the outgoing messages are composed of data values that lie non-contiguously in the sender's memory, or the values of the incoming messages need to be placed in designated, non-contiguous locations in the receiver's memory. There exist models, such as $\log_n$P or $\log_3$P [7], that include *regularly strided* memory accesses, which are the simplest operations for packing and unpacking messages. As future work, we will extend our models to cover the packing and unpacking costs, and we will address these from the angle of memory bandwidth contention. Specifically, we will consider the situation where a preceding computation produces the values before they are collected from *irregularly* non-contiguous memory locations and packed as outgoing messages. Moreover, the different MPI processes on a same node may carry out the "packing step" at different paces (e.g., some processes have less data to pack and can thus initiate the MPI commands earlier), such that the message transfer and message packing by different MPI processes may compete for the same memory bandwidth. As another topic of future work, we will consider modeling overlaps between computation (including message packing and unpacking) and point-to-point MPI communication. Contention for the same memory bandwidth, see e.g. [9, 16], will also be central in this regard.

One of the main benefits of using our "staircase" modeling approach is the capability of pinpointing per-process time usage, thanks to an elaborate modeling of the bandwidth contention between messages of various sizes. Realistic experiments have seen actual time usages greatly varying from process to process, thus the fastest finishing MPI processes may have considerable idle time. We plan to incorporate such modeling into future mesh partitioning strategies, with the goal of producing better partitioning results where the fastest finishing processes do not have to wait long for the slowest processes to complete.

References

[1] Alexandrov, A. et al. “LogGP: Incorporating long messages into the LogP model—one step closer towards a realistic model for parallel computation”. In: *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*. 1995, pp. 95–105.

[2] Bar-Noy, A. and Kipnis, S. “Designing broadcasting algorithms in the postal model for message-passing systems”. In: *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*. 1992, pp. 13–22.

[3] Bastian, P. et al. “A generic grid interface for parallel and adaptive scientific computing. Part II: implementation and tests in DUNE”. In: *Computing* vol. 82, no. 2-3 (2008), pp. 121–138.

[4] *Betzy – The most powerful supercomputer in Norway*. https://documentation.sigma2.no/hpc_machines/betzy.html.

[5] Bienz, A., D. Gropp, W., and N. Olson, L. “Improving Performance Models for Irregular Point-to-Point Communication”. In: *Proceedings of the 25th European MPI Users’ Group Meeting*. 2018, pp. 1–8.

[6] Bienz, A., Gropp, W. D., and Olson, L. N. “Reducing communication in algebraic multigrid with multi-step node aware communication”. In: *The International Journal of High Performance Computing Applications* vol. 34, no. 5 (2020), pp. 547–561.

[7] Cameron, K. W., Ge, R., and Sun, X.-H. “$\log_n$P and $\log_3$P: Accurate Analytical Models of Point-to-Point Communication in Distributed Systems”. In: *IEEE Transactions on Computers* vol. 56, no. 3 (2007), pp. 314–327.

[8] Chen, W. et al. “LogGPO: An accurate communication model for performance prediction of MPI programs”. In: *Science in China Series F: Information Sciences* vol. 52, no. 10 (2009), pp. 1785–1791.

[9] Denis, A., Jeannot, E., and Swartvagher, P. “Interferences between Communications and Computations in Distributed HPC Systems”. In: *50th International Conference on Parallel Processing*. ICPP 2021. 2021.

[10] *DUNE Numerics*. https://dune-project.org. Last accessed: May 2022. 2021.

[11] Gropp, W., Olson, L. N., and Samfass, P. “Modeling MPI communication performance on SMP nodes: Is it time to retire the ping pong test”. In: *Proceedings of the 23rd European MPI Users’ Group Meeting*. 2016, pp. 41–50.

[12] Hockney, R. W. and Jesshope, C. R. *Parallel Computers Two: Architecture, Programming and Algorithms*. 2nd. IOP Publishing Ltd., 1988.

[13] Hockney, R. W. “The communication challenge for MPP: Intel Paragon and Meiko CS-2”. In: *Parallel Computing* vol. 20, no. 3 (1994), pp. 389–398.

[14] Ino, F., Fujimoto, N., and Hagihara, K. “LogGPS: A Parallel Computational Model for Synchronization Analysis”. In: *SIGPLAN Not.* vol. 36, no. 7 (June 2001), pp. 133–142.

[15] Kim, J. et al. “Technology-Driven, Highly-Scalable Dragonfly Topology”. In: *2008 International Symposium on Computer Architecture*. 2008, pp. 77–88.

[16] Langguth, J., Cai, X., and Sourouri, M. “Memory Bandwidth Contention: Communication vs Computation Tradeoffs in Supercomputers with Multicore Architectures”. In: *IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. 2018, pp. 497–506.

[17] Moritz, C. A. and Frank, M. I. “LoGPC: Modeling network contention in message-passing programs”. In: *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. 1998, pp. 254–263.

[18] *MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE*. https://mvapich.cse.ohio-state.edu/benchmarks/.

[19] *OPEN POROUS MEDIA*. https://opm-project.org. Last accessed: May 2022. 2022.

[20] Rasmussen, A. F. et al. “The Open Porous Media Flow reservoir simulator”. In: *Computers & Mathematics with Applications* vol. 81 (2021), pp. 159–185.

[21] Rico-Gallego, J.-A. and Díaz-Martín, J.-C. “τ-Lop: Modeling performance of shared memory MPI”. In: *Parallel Computing* vol. 46 (2015), pp. 14–31.

[22] Rico-Gallego, J.-A., Díaz-Martín, J.-C., and Lastovetsky, A. L. “Extending τ-Lop to model concurrent MPI communications in multicore clusters”. In: *Future Generation Computer Systems* vol. 61 (2016), pp. 66–82.

[23] Rico-Gallego, J.-A. et al. “A survey of communication performance models for high-performance computing”. In: *ACM Computing Surveys* vol. 51, no. 6 (2019). Article 126.

[24] Sander, O. *DUNE — The Distributed and Unified Numerics Environment*. Vol. 140. Lecture Notes in Computational Science and Engineering. Springer Cham, 2020.

[25] Shpiner, A. et al. “Dragonfly+: Low Cost Topology for Scaling Datacenters”. In: *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*. 2017, pp. 1–8.

[26] Thune, A., Cai, X., and Rustad, A. B. “On the impact of heterogeneity-aware mesh partitioning and non-contributing computation removal on parallel reservoir simulations”. In: *Journal of Mathematics in Industry* vol. 11 (2021).

Acknowledgments The work of the first author was partially supported by the Research Council of Norway under contract 237898. The work of the fourth author was partially supported by the European High-Performance Computing Joint Undertaking under grant agreement No. 956213 and the Research Council of Norway under contract 329017. The research presented in this paper benefited from the Experimental Infrastructure for the Exploration of Exascale Computing (eX3), which was financially supported by the Research Council of Norway under contract 270053, as well as resources provided by Sigma2 - the National Infrastructure for High Performance Computing and Data Storage in Norway.

Appendix II.A Example source code

The following C code implements a function that predicts the per-process time usages of many-pair, point-to-point MPI communication, by using the performance model of mixed intra- and inter-socket traffic as described in Section 3.5

The input arguments are the total number of MPI processes (`num_procs`), number of processes per group, two communication matrices (in the *compressed column sparse* format) that contain the source process identity and size of all incoming messages on, respectively, the intra- and inter-socket levels, plus two corresponding sets of latency and tabulated bandwidth values. The output argument, `predictions`, will contain the estimated per-process time usages on return.

```
void two_level_predict (int num_procs, int num_procs_pr_group,
                        int *jcols1, int *source_ids1, int *recv_sizes1,
                        int *jcols2, int *source_ids2, int *recv_sizes2,
                        float tau1, float tau2, float *bw_values1, float *bw_values2,
                        float *predictions  /* output */)
{
  int g, i, j, k;
  const int N = num_procs_pr_group;

  // allocation of temporary arrays
  float *msg_arrive_times1 = (float*)malloc(jcols1[num_procs]*sizeof(float));
  float *msg_arrive_times2 = (float*)malloc(jcols2[num_procs]*sizeof(float));
  float *proc_theta = (float*)malloc(N*sizeof(float));
  float *rest_V = (float*)malloc(N*sizeof(float));

  for (g=0; g<num_procs/N; g++) {  // go through the groups one by one
    int max_msgs_pr_proc = 0, proc_id, num_msgs, recv_v1, recv_v2;

    for (i=0; i<N; i++) {   // preparation work per process
      proc_id = g*N+i; recv_v1 = recv_v2 = 0;

      for (j=jcols1[proc_id]; j<jcols1[proc_id+1]; j++) // summing level-1 messages
        recv_v1 += recv_sizes1[j];
      for (j=jcols2[proc_id]; j<jcols2[proc_id+1]; j++) // summing level-2 messages
        recv_v2 += recv_sizes2[j];

      proc_theta[i] = (recv_v1==0&&recv_v2==0) ? 0 : 1.0*recv_v1/(recv_v1+recv_v2);
      rest_V[i] = recv_v1 + recv_v2;  // total receiving volume per process

      if ((num_msgs=jcols1[proc_id+1]-jcols1[proc_id]) > max_msgs_pr_proc)
        max_msgs_pr_proc = num_msgs;
      if ((num_msgs=jcols2[proc_id+1]-jcols2[proc_id]) > max_msgs_pr_proc)
        max_msgs_pr_proc = num_msgs;
    }

    char *finished = (char*)calloc(N,sizeof(char));  // all initial values 0
    float last_prediction = 0.;
```

```
  for (k=0; k<N; k++) {  // step-wise prediction of per-process total receive time
    float min_step = FLT_MAX, step, bw; int index_p = -1;
    for (i=0; i<N; i++) {  // find the process that will finish first in this step
      if (!finished[i]) {
        bw=(proc_theta[i]*bw_values1[N-k]+(1-proc_theta[i])*bw_values2[N-k])/(N-k);
        step = rest_V[i]/bw;
        if (step<min_step) {
          min_step = step; index_p = i;
        }
      }
    }

    for (i=0; i<N; i++)  // go through all the unfinished processes for this step
      if (!finished[i]) {
        bw=(proc_theta[i]*bw_values1[N-k]+(1-proc_theta[i])*bw_values2[N-k])/(N-k);
        rest_V[i] -= min_step*bw;  // update the rest receiving volume
      }

    last_prediction += min_step;
    predictions[g*N+index_p] = last_prediction;
    finished[index_p] = 1;
  }

  int *perm_indices = (int*)malloc(max_msgs_pr_proc*sizeof(int)); // help array

  // dissect per-process total time to pinpoint each message's arrival time
  for (i=0; i<N; i++) {
    proc_id = g*N+i;

    // ------- for level-1 incoming messages  -------
    num_msgs = jcols1[proc_id+1]-jcols1[proc_id];
    for (j=0, recv_v1=0; j<num_msgs; j++) {
      recv_v1 += recv_sizes1[jcols1[proc_id]+j]; perm_indices[j] = j;
    }

    // sort the incoming level-1 messages per process according to their sizes
    find_sorting_indices(&(recv_sizes1[jcols1[proc_id]]),perm_indices,0,num_msgs-1);

    float last_t = 0., factor = predictions[proc_id]/recv_v1; int last_s = 0;
    for (j=0; j<num_msgs; j++) {  // pinpoint arrival time per level-1 message
      k = jcols1[proc_id]+perm_indices[j];
      last_t += (num_msgs-j)*(recv_sizes1[k]-last_s)*factor;
      msg_arrive_times1[k] = last_t;
      last_s = recv_sizes1[k];
    }

    // ------- for level-2 incoming messages  -------
    num_msgs = jcols2[proc_id+1]-jcols2[proc_id];
    for (j=0, recv_v2=0; j<num_msgs; j++) {
      recv_v2 += recv_sizes2[jcols2[proc_id]+j]; perm_indices[j] = j;
    }

    // sort the incoming level-2 messages per process according to their sizes
    find_sorting_indices(&(recv_sizes2[jcols2[proc_id]]),perm_indices,0,num_msgs-1);

    last_t = 0.; factor = predictions[proc_id]/recv_v2; last_s = 0;
    for (j=0; j<num_msgs; j++) {  // pinpoint arrival time per level-2 message
      k = jcols2[proc_id]+perm_indices[j];
      last_t += (num_msgs-j)*(recv_sizes2[k]-last_s)*factor;
      msg_arrive_times2[k] = last_t;
      last_s = recv_sizes2[k];
    }
  }

  free (finished); free (perm_indices); // free storage of these two help arrays
}

// enlarge per-process predicted time if individual msg delivery time is larger
for (i=0; i<num_procs; i++)
  for (j=jcols1[i]; j<jcols1[i+1]; j++) {
    k = source_ids1[j];
    if (msg_arrive_times1[j] > predictions[k])
      predictions[k] = msg_arrive_times1[j];
  }
for (i=0; i<num_procs; i++)
  for (j=jcols2[i]; j<jcols2[i+1]; j++) {
    k = source_ids2[j];
    if (msg_arrive_times2[j] > predictions[k])
```

```
        predictions[k] = msg_arrive_times2[j];
    }

  // add the latency
  for (i=0; i<num_procs; i++)
    predictions[i] += (jcols1[i+1]-jcols1[i])*tau1 + (jcols2[i+1]-jcols2[i])*tau2;

  free(msg_arrive_times1);free(msg_arrive_times2);free(rest_V);free(proc_theta);
}
```

The above code consists of two main parts: Pre-calculation of the total volumes of intra- and inter-socket incoming traffic for each process and its individual θ_i value, according to (12); Estimation of the per-process time usage including a dissection of the delivery time for each message, according to (13)-(17). The function `find_sorting_indices` is used to find a permutation array that sorts an un-ordered vector.

Paper III

Distributed wells for enhancing the efficiency of parallel reservoir simulation

Andreas Thune, Markus Blatt, Xing Cai, Alf Birger Rustad

In preparation.

III

Distributed wells for enhancing the efficiency of parallel reservoir simulation

Andreas Thune Markus Blatt Xing Cai Alf Birger Rustad

April 25, 2023

Abstract

Parallel computing is needed to enable efficient simulations of multi-phase flow in high resolution petroleum reservoirs. Features of real-world reservoir models, such as highly heterogeneous geological properties and the presence of production and injection wells, must be taken into account in the domain decomposition strategy, which is the first step of parallelizing any reservoir simulator and has vital importance for achieving good parallel performance. Earlier research results show that the parallelized ILU and CPR preconditioners benefit from strongly coupled grid cells not being split between processes. This is has led to a prevailing practice that the production and injection wells in a 3D reservoir model, which each strongly connects multiple grid cells, are kept undivided during the domain decomposition, at the cost of additional challenges related to mesh partitioning. In this paper, we present an implementation of a reservoir simulator that allows the splitting of individual wells onto multiple processes, and a quantitative analysis of how distributed wells affect the partitioning quality and overall performance. We also demonstrate the parallel scaling capabilities of the ILU, AMG and CPR preconditioners when used in simulating two realistic reservoir models, a million-cell black-oil model, and a 9-million-cell CO_2 storage model. By analysing the computational components of these preconditioners, we explain their comparative performance. The new implementation and simulations are done using the OPM open-source reservoir simulator *Flow*.

1 Introduction

The purpose of this paper is to study various aspects of the parallel performance of reservoir simulation on large and realistic 3D reservoir models. That is, megacell models with one million or more active cells and complex geology and fluid physics. We are particularly interested in allowing distributing a production/injection well across multiple subdomains and how this impacts the resulting communication overhead and overall performance. To the best of our knowledge, this subject has not been studied in the existing literature of numerical reservoir simulation. Allowing the distribution of a single well across multiple subdomains simplifies mesh partitioning but complicates the reservoir simulator implementation significantly, especially when considering complex well models such as multi-segment wells in prediction mode that needs tailored solvers for the well equations. Hence, analyzing the performance benefits beforehand is important. We will also include a study on how the choice of linear solver preconditioner affects the scaling properties of the parallel simulator. This includes an ILU0-based implementation which is competitive with the nested factorization found in the most widely used proprietary reservoir simulator, and a CPR implementation similar to what is found in the latest generation

proprietary reservoir simulators. It is therefore hoped that our results have general interest across most relevant reservoir simulation scenarios.

Injection and production wells greatly impact the fluid flow in petroleum reservoirs, but on a much finer scale than the typical resolution of a reservoir mesh. Special well models are needed to accurately capture the wellbore-reservoir interaction. Moreover, the wells can penetrate multiple layers of the reservoir, and may therefore connect physically separate regions. Requiring that the grid cells perforated by a wellbore remain on the same partition in a parallel reservoir simulation complicates the domain decomposition step and can potentially be a hindrance to performance. On the other hand, distributing a well between multiple processes also introduces additional overhead.

1.1 Reservoir models

To demonstrate our new implementation and test the resulting parallel performance, we will consider two reservoir models developed by Equinor. Unfortunately, none of these reservoir models is openly available due to legal constraints. This means that our results are not readily reproducible by our peers. However, we have no other choice if realistic models are to be considered. While the openly available and commonly used SPE10 3D-model has just over one million cells, it is by no means representative of reservoir models typically found in industrial usage. Specifically, it is a box model with a Cartesian grid, rather than a realistic reservoir with complex geology. It has a simplified fluid model (two-phase immiscible), and a simplistic well set-up. Nor was the SPE10 case intended to be realistic, it was created for the purpose of studying upscaling techniques. Hence, we are not aware of any relevant open dataset for our study. On the other hand, our new and earlier implementations are fully available for testing both in compiled binaries and as open-source code for inspection and further improvement or study.

We have included one oil reservoir model with approximately one million active cells. This model is a realistic representation of a North Sea oil field. The fluid system is black oil, which means three phases (oil, water and gas) are present in the equations. Moreover, the phases are compressible and the two hydrocarbon phases (gas and oil) are miscible. There are more than a hundred production and injection wells, each using a standard well model (as opposed to a multi-segment model). Many of the wells are deviated (rather than vertical), introducing connections crossing large parts of the reservoir both vertically and laterally. In order to keep the computational time manageable for this study, only the first couple of years of production are included. Later in the field life, more wells are typically interplaying with more complex flow patterns, this may influence the results significantly, potentially making distributed wells more favourable.

Secondly, we have a CO_2 storage model [1] with nine million active cells including a well for CO_2 injection into the reservoir. The model represents Smeaheia, the planned location for CO2 storage on the Norwegian continental shelf. There is an openly available version of this model, but that version only has a few hundred thousand active cells, and is hence not well suited for our study. For this study we will use a simple fluid model though. Specifically, we set CO2 and water as immiscible, and ignore the temperature, otherwise full complexity is included. This simplification of the fluid model is justified by the need to validate the correctness of the simulation results with reference solutions, also ensuring that the numerical performance results are relevant and can be directly compared to alternative implementations in other reservoir simulators.

Because we want to validate our new distributed-well implementation, we will also include results from the small, realistic and openly available Norne model [2]. Norne is a black-oil model based on a real oil field in the Norwegian Sea. The unstructured mesh of the Norne reservoir con-

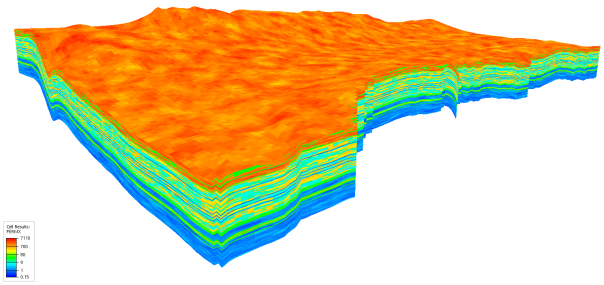


Figure 1: Permeability distribution in the CO_2 storage reservoir meshes (openly available model).

sists of 44420 grid cells, contains multiple faults and has a total of 36 active production/injection wells.

2 Simulator and mathematical model

In this section we give an overview of the *Flow* reservoir simulator and briefly present the standard well model for use in reservoir simulation [3]. We also describe the numerical solution strategy employed by *Flow* to solve the linear system of equations that arise from linearizing and discretizing the black-oil models [4, 5] or extensions like CO_2 [6] reservoir storage simulation.

2.1 OPM *Flow*

The implementation of distributed wells and parallel simulation experiments conducted in this paper use the open-source reservoir simulator *Flow* [7], provided by the Open Porous Media (OPM) initiative. *Flow* supports industry standard input and output formats, and offers fully-implicit discretizations of the black-oil model. In addition to the base black-oil simulator, *Flow* also has many extensions, such as CO_2-EOR [8] or CO_2 storage [9].

The linear algebra and corner-point grid implementations of *Flow* are based on the DUNE [10–13] software framework. Parallelization of *Flow* is enabled by both multi-threading and use of MPI processes. However, multi-threading only applies to parts of the simulator, namely matrix assembly and I/O. Parallelism is therefore mainly achieved by mesh partitioning that divides the reservoir grid cells evenly among a prescribed number of processes at the beginning of the simulation. MPI is then used to handle communication between the processes.

2.2 Standard well model

Production and injection wells connect the surface to the subsurface reservoirs, so understanding the impact of wells on the flow in the reservoir is a key aspect of reservoir simulation. Typically, the wells influence the flow near the wellbore on a finer scale than the resolution of the reservoir mesh allows. Additional equations are thus included to better capture the well-reservoir interaction.

The standard well model for single-phase flow, formulated in [14], expresses the flow rate of a well passing through a grid cell in terms of the difference between the pressure at the wellbore and the average grid cell pressure. The wellbore pressure in the grid cell that contains the

endpoint of the well is referred to as the bottom-hole pressure p_{bhp}, and the flow rate q_i in the well-connected cell i is defined in terms of bottom-hole pressure and cell pressure p_i:

$$q_i^r = J_i(p_i - (p_{bhp} + H_i)). \tag{1}$$

The factor J_i depends on the transmissibility and mobility, where the latter is calculated using the pressure-dependent fluid viscosity and formation volume factor properties, as well as relative permeability in the multi-phase case. In (1), H_i is the pressure difference between the bottom-hole depth and the wellbore pressure in the well-connected cell i. This is because in most cases the wellbore pressure along a 3D well is not constant, where the location of cell i relative to the well endpoint determines the actual value of H_i.

For wells that perforate multiple grid cells, we end up with a flow rate for each well-connected cell. These relate to each other through the inflow performance relationship presented in [15], using the total surface flow rate Q_t. By conservation, Q_t equals the sum of the flow rates q_i over all the well-connected cells:

$$Q_t - \sum_i q_i^r = 0. \tag{2}$$

The well-reservoir interaction for single-phase flow is modelled by the bottom-hole pressure p_{bhp} and total surface flow rate Q_t. However, the relationships in (1)-(2) combine to a single equation. The system is closed by introducing a well control, that governs one of the two parameters. The well control is either a desired upper surface flow rate, or a lower bottom-hole pressure threshold:

$$p_{bhp} - p_{bhp}^{target} = 0, \tag{3}$$

$$Q_t - Q_t^{target} = 0. \tag{4}$$

Eqs. (1)-(4) constitute a simplified description of the standard well model for single-phase flow. The case of multi-phase flow is extended by adding one variable and equation for each additional phase. The standard well model for the 3-phase case is described in [3]. The total flow rate is redefined and two new variables, weighted water and gas fraction of total well flow F_w and F_g, are introduced:

$$Q_t = \sum_\alpha g_\alpha Q_\alpha, \quad F_w = \frac{g_w Q_w}{Q_t}, \quad F_g = \frac{g_g Q_g}{Q_t}, \tag{5}$$

where Q_α and g_α are flow rate and weighting factor of phase component α.

2.3 Well equations in the linear system

When solving the black-oil model with Newton's method and implicit time integration, we need to solve a system of linear equations $MX = F$ for each Newton iteration. The number of unknowns per cell equals the number of fluid phases p in the model. In addition to the fluid phase unknowns, each active well adds $p + 1$ unknowns to the system. If we assume that we have N cells and L active wells in our model, the total number of unknowns is $pN + (p + 1)L$. We can separate the unknowns X into fluid phase unknowns $x \in \mathbb{R}^{pN}$ and the well unknowns $x_1^w, ..., x_L^w \in \mathbb{R}^{p+1}$, and then get a block matrix structure of the system $MX = F$ in the following

form:

$$M \begin{pmatrix} x \\ x_1^w \\ x_2^w \\ \vdots \\ x_L^w \end{pmatrix} = \begin{bmatrix} A & C_1 & C_2 & \cdots & C_L \\ B_1 & D_1 & 0 & \cdots & 0 \\ B_2 & 0 & D_2 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ B_L & 0 & 0 & \cdots & D_L \end{bmatrix} \begin{pmatrix} x \\ x_1^w \\ x_2^w \\ \vdots \\ x_L^w \end{pmatrix} = \begin{pmatrix} f \\ f_1^w \\ f_2^w \\ \vdots \\ f_L^w \end{pmatrix}. \tag{6}$$

Rather than solving the full system in (6), one can instead form the Schur complement and end up with a system of the dimension of the fluid phase part A of the system:

$$\left(A - \sum_{i=1}^{L} C_i D_i^{-1} B_i \right) x = f - \sum_{i=1}^{L} C_i D_i^{-1} f_i^w. \tag{7}$$

In *Flow*, the system based on the Schur complement $S = A - \sum_{i=1}^{L} C_i D_i^{-1} B_i$ is solved rather than the original formulation of the system (6). The often ill-conditioned linear system (7) is solved with iterative linear solvers, typically BiCGStab [16] or GMRES [17].

Preconditioning of the linear system (7) is necessary for finding a solution. *Flow* has an option to apply the preconditioner only to the fluid phase part of the system, meaning that we precondition A instead of S. There are three preconditioner options in *Flow*, incomplete LU factorization with zero fill-in (ILU0), algebraic multigrid (AMG) and constraint pressure residual (CPR). For AMG, *Flow* uses the DUNE implementation described in [18, 19]. This implementation builds the coarser levels by aggregating matrix entries based on a strength of connection criteria, and is well suited for problems with highly variable coefficients. In this paper, the AMG preconditioner is always used with an ILU0 smoother, which is applied in a single step pre and post solving the coarse system.

The CPR preconditioner can be thought of as a two-level method, where the fine level corresponds to the full system, and the coarse system corresponds to the pressure equation. In our simulations, we use the quasi-impes approach to form the pressure equation as described in [20]. AMG is applied to the coarse pressure equation system, and one step of ILU0 pre- and post-smoothing is applied to the entire coarse system. An illustration of the CPR two-level method as used in this paper is presented in Figure 2.

2.4 Parallelization of the linear solver

The parallelization of the *Flow* reservoir simulator is enabled by a division of the reservoir grid cells between a number of MPI processes. The division of cells is done only once, at the very beginning of the simulation, using a graph partitioning algorithm. Each process becomes the owner of a subset of the mesh cells, and is responsible for assembling the part of the linear system associated with these cells. However, a non-overlapping cell division makes local matrix assembly on each individual process incomplete without inter-process communication. Therefore, the *Flow* implementation includes a layer of overlapping "ghost" cells to each subdomain, to allow for local matrix assembly without the need for communication. In addition to avoiding communication, inclusion of ghost cells means that the sequential discretization and matrix assembly can be re-used for parallel simulations without modification.

The parallel iterative Krylov solver is implemented by parallelizing four computational kernels: vector addition, inner product, sparse matrix-vector multiplication and system preconditioning. In contrast to matrix assembly, solving the linear system in parallel requires inter-process

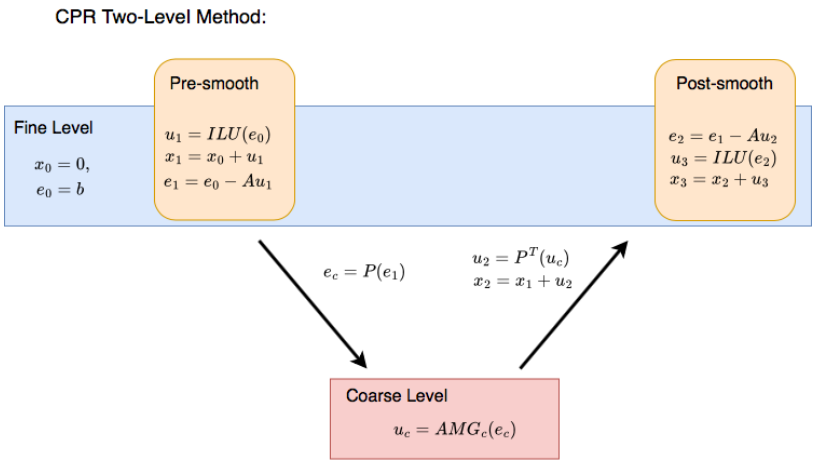


Figure 2: Two-level method for solving $Ax = b$, with $x_0 = 0$ as initial guess, AMG as coarse solver and ILU0 as pre- and post-smoother. The quasi-impes level transfer operators P and P^T moves a vector from fine to coarse level and coarse to fine level.

communication. The parallel inner product is enabled by a collective communication operation. Parallel sparse matrix-vector multiplication is performed by each process executing its local matrix-vector multiplication, after the vector entries associated with ghost cells are received on each process with point-to-point communication.

Parallelizing the preconditioning operation can be challenging. ILU factorization for example, is inherently sequential, although some parallelism can be achieved through graph colouring techniques. An alternative approach, employed by *Flow*, is to use a Block-Jacobi preconditioner, where the blocks correspond to process subdomains, and then apply ILU0 only to each local system. The main benefit of the Block-Jacobi preconditioner is its high degree of parallelism, resulting from a total absence of data-dependence between the blocks. Parallelization of the more complicated AMG and CPR preconditioners is similarly enabled by utilizing Block-Jacobi based smoothers and restricting agglomeration to Jacobi blocks. This inter-block data independence can be observed in Figure 3, where the Block-Jacobi idea is illustrated.

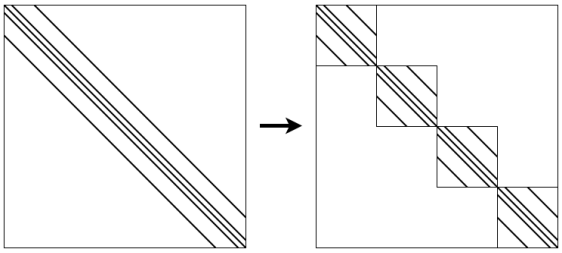


Figure 3: Illustration of how to create a block diagonal matrix based on subdomains (right) from an original matrix (left).

3 Distributed-well implementation

In this section we give an overview of the distributed-well implementation in OPM's *Flow* reservoir simulator.

Let $I \subset \mathbb{N}$ be an index set that contains exactly one index per grid cell and let each grid cell be identified by a unique index. As outlined in Section 2.4, the cells are divided between the available P processes. Consequently, let I_p, $p \in \mathcal{P} := \{1, \ldots, P\}$, be a non-overlapping partitioning of the index set I, such that $\bigcup_{p=1}^{P} I_p = I$ and $I_p \cap I_q = \emptyset$ for $p \neq q$.

For a particular well i, let the set $W_i \subset I$ be the indices associated with cells that the well i perforates. Then we say the well is"distributed" among multiple processes, if there exists no process p such that $W_i \subset I_p$ holds. We define the index set $(W_i)_p := W_i \cap I_p$ as the indices of the local cells on process p that the well perforates. Furthermore we denote with $\mathcal{P}_i = \{p \in \mathcal{P} \mid (W_i)_p \neq \emptyset\}$ the set of MPI processes that possess cells that well i perforates.

3.1 Parallel well Schur complement

The operations concerning the wells in (7) can be split into applying the linear operators B_i and C_i and solving the tiny linear system $D_i x_i = h_i$.

The matrix B_i has a block structure, it has one block row and its number of block columns equals the number of cells in the computational grid. The number of rows per block equals the number of well equations of well i, whereas the number of columns per block equals the number of fluid phases. The only non-zero blocks of B_i correspond to the cells that are perforated by well i. Let $(B_i)_p$ be the submatrix of B_i with only nonzeros for columns with indices in $(W_i)_p$ and $(x_i)_p$ a sparse vector with only nonzeros for indices in I_p, then we can compute

$$y_i = B_i x_i = \sum_{p=1}^{P} (B_i)_p (x_i)_p \,.$$

Here, each process that has one or several non-zero blocks of B_i computes the local matrix vector product first and the result is summed over all processes using communication.

The block sparsity of C_i is the same as that of the transpose of B_i. Assuming that each process knows the correct value z_i for well i, it can calculate the correct result for its local cells without communication using

$$(v_i)_p = (C_i)_p z_i \,.$$

The above notation $(C_i)_p$ means the submatrix of C_i with nonzeros only in rows with index in $(W_i)_p$. Note that ghost values for v_i need to updated using communication afterwards for the rest of the simulator.

To be able to compute correct values when solving

$$D_i z_i = y_i$$

each process also needs to know the correct values of the small matrix D_i. Note that the values of D_i depend on data of all perforations of the well. Hence its computation needs communication between all processes with perforations of this well.

Note that even if the perforated cells of the well reside on a different process it suffices to know the correct value of z_i to compute $(C_i)_k z_i$ for $k \in I_p$ and the process only computes for cells stored locally. Hence this is a local computation that does not require communication.

3.2 Parallel well setup

As mentioned above we want to achieve that each process which stores the perforated cells of well i also has the correct values of matrix D_i, as well as all other global properties of the well, such as its residual, flow rates etc. Then for matrix B_i it suffices to store only nonzero blocks whose column index is in $W_p \subset I_p$ on process p. For matrices C_i it suffices to store only the nonzeros with row index in $W_p \subset I_p$ on process p. The local values of C_i and B_i can be calculated without any communication from the well and local properties attached to the cells.

To have the correct values of D_i and other global properties stored on every process that stores the cells perforated by the well, an additional step including communication is needed. First each process with perforations of well i calculates its local contributions to D_i or other properties. Then we compute the global sum over these processes to get the correct full values.

The perforations of a standard well are ordered as specified in the input file. For some computations, like computing the pressure difference, this ordering needs to be adhered to. Therefore it is possible to access values of perforations that are above or below the current one. Note that this requires communication as those values might reside on other processes. Therefore we have developed functionality to create vectors with values above/below the perforation using communication to facilitate getting this information.

Note that some computations during the well setup need to happen serially (in a specific order of the well perforations). Therefore there is also an utility function that gathers all the needed data for all perforations of a well on all processes in $\mathcal{P}_i$ to create a vector with entries for all perforations. After that is created each process can do the same computation in the correct order no matter how the well is distributed and copy the new values back to the local representation of that vector without further communication.

In addition one is able to compute a partial sum governed by the order of the well perforations even for arbitrarily distributed wells. This summation gathers values from all perforations of a well i on the processes in $\mathcal{P}_i$. Then each process computes the partial sum on this using the correct ordering.

3.3 Custom MPI communicator for each distributed well

As seen above we need communication when setting up the well equations and solving the well Schur complement (the right hand side of (7)). An important aspect is that this communication for well i is limited to only a subset of processes $\mathcal{P}_i \subset \mathcal{P}$ of all MPI processes. This means that the communication needed to compute a well is independent from those of other wells as long as two wells do not perforate the same cells. The latter is a rather rare case in reservoir simulation and would usually only happen for a few wells in the model. If this is the case some of the communication might need to wait for others to complete. Apart from this rare case it should be possible for the communication to happen concurrently without inference.

To facilitate this we set up a custom MPI communicator for each well i that includes only the processes with rank in $\mathcal{P}_i$. This communicator is used whenever communication is needed for a distributed well. This way MPI will ensure that such communication will not interfere with communication for other wells or communication done by the rest of the communicator. This custom communicator is also mandatory to be able to use MPI's collective communication for computing global properties of a well.

4 Mesh partitioning and distributed wells

Wells connect different parts of the reservoir together, which can complicate the mesh partitioning step of parallel reservoir simulation. In this section, we present some strategies for dealing with the well connections during the mesh partitioning step. We also study how the wells impact the quality of the partitioning result.

The first step of parallelizing a reservoir simulation is to partition the reservoir mesh into disjoint subdomains. Typically, the aim is to create a partitioning of the mesh that minimize inter-process communication, while also ensuring good load balancing. The very heterogeneous permeability present in real-world reservoirs means that some grid cells are more strongly coupled than the others. It is therefore often beneficial for the linear solver performance to ensure that the partitioner does not separate such strongly-coupled cells. Unfortunately, the solver convergence efficiency can come in conflict with the goal of communication minimization. The inclusion of wells in the reservoir complicates matters further, by often connecting multiple mesh cells that are physically far apart. The standard approach for creating a disjoint division of the unstructured reservoir mesh is to apply graph partitioning on a weighted graph representation of the mesh. Here, the reservoir mesh is translated into a graph by turning each cell into a graph vertex. If two grid cells i and j share an interface, or if i and j are perforated by a common well, this connection is translated into an edge between vertices i and j in the graph.

A weighted graph $G = (V, E, \sigma, \omega)$ is composed of a set of vertices V, a set of edges $E \subset V \times V$, and two weight functions $\sigma : V \to \mathbb{R}$ and $\omega : E \to \mathbb{R}$. A P-way partitioning of G is an optimization problem defined as follows: Partition the vertex set V into P subsets $V_1, V_2, \ldots, V_P$ of approximately equal size, while minimizing the summed weight of all the "cut edges": $e = (v_i, v_j) \in E$ connecting vertices belonging to different vertex subsets. Suppose $\mathcal{C}$ denotes the "cut set" of a partitioned G, containing all the cut edges. The graph partitioning problem can be formulated more precisely as a constrained optimization problem, where the objective function J is the sum of the weights of all members of $\mathcal{C}$, also called *edge-cut*:

$$\min \; J(\mathcal{C}) = \sum_{e \in \mathcal{C}} \omega(e), \tag{8}$$

$$\text{subject to} \quad \frac{\max_p \sum_{v \in V_p} \sigma(v)}{\frac{1}{P} \sum_{v \in V} \sigma(v)} < \epsilon, \tag{9}$$

where $\epsilon \geq 1$ is the imbalance tolerance of the load balancing constraint.

When all the edge weights are set to 1, the edge-cut $J(\mathcal{C})$ is an approximation of the total communication volume, and minimizing $J(\mathcal{C})$ will reduce the communication overhead. As demonstrated in [21–23], using a non-uniform edge-weight function ω can improve the convergence of parallel linear solvers when dealing with highly heterogeneous linear systems. In the *Flow* reservoir simulator, we consider face- and well-edges separately in the edge-weight function, which looks as follows:

$$\omega(e) = \begin{cases} T_{ij} & e \text{ is a face-edge,} \\ K & e \text{ is a well-edge.} \end{cases} \tag{10}$$

The face-edge weight T_{ij} is either uniform, or based on cell interface transmissibility. Uniform weights would result in less communication overhead, while pure transmissibility edge-weights yield better linear solver convergence. With a reservoir simulator that allows for distributed wells, we have three options for assigning the K value for the well-edges. The first option is to set $K = \infty$, or in practice a really large number, in order to avoid wells being split between partitions. A second option is to set $K = 1$ in combination with uniform face-edge weights, acknowledging that splitting wells results in increased communication overhead. The last option

is to set $K = 0$, thereby completely ignoring well induced cell connections. When the simulator implementation handles the well equations separately and not included in the system matrix, as is the case in *Flow*, wells do not impact the convergence of the linear solver. $K = 0$ is therefore the appropriate well-edge weight when using transmissibility face-edge weights.

To investigate the impact of well-edge weight on the partition quality, we will consider four simple metrics: total communication volume, maximum communication volume per process, total number of partition neighbours and maximum number of neighbours per process. We define the per-process communication volume of the i-th disjoint subset V_i of a partitioning of V as:

$$CV(V_i) = \sum_{v \in V_i} (\lambda(v) - 1). \tag{11}$$

The function $\lambda(v)$ equals the number of subsets that v and all nodes with an edge connection to v belong to. Total and max communication volume is then simply the sum and maximum of all the per-process communication volumes $CV(V_i)$. In Figure 4, we present these metrics when performing P-way partitioning of the black-oil case reservoir mesh (see Section 1.1) with transmissibility face-edge weights and $K = 0$ or $K = \infty$ well-edge weights.

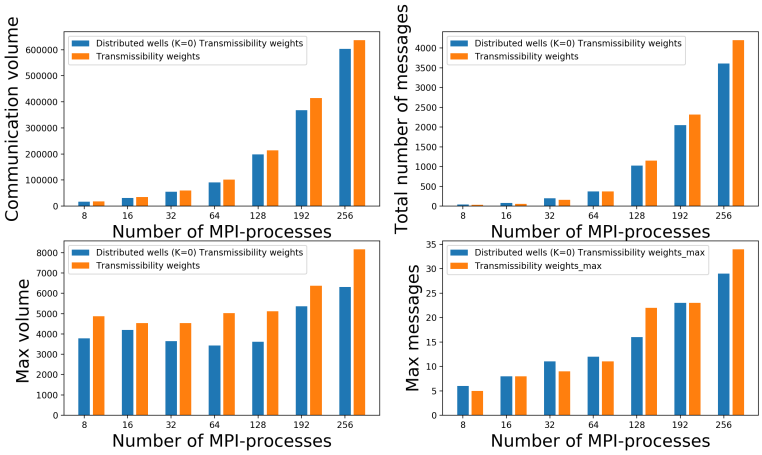


Figure 4: Communication volume (top left), total number of messages (top right), max per-process communication volume (bottom left) and max per-process messages (bottom right) when partitioning the black-oil case reservoir mesh with transmissibility face-edge weight and either $K = 0$ (blue bar) or $K = \infty$ (orange bar).

In Figure 4 we observe that using $K = 0$ well-edge weights results in significantly improved partition quality over $K = \infty$ well-edge weights when partitioning the black-oil reservoir case. For the total and max per-process communication volume metrics, $K = 0$ produces better results than $K = \infty$ for all number of MPI processes. Results for the total and max messages metrics are less conclusive, as $K = \infty$ produces fewer messages in some cases. However, $K = 0$ seems to outperform $K = \infty$ in the message count when the number of MPI processes is large.

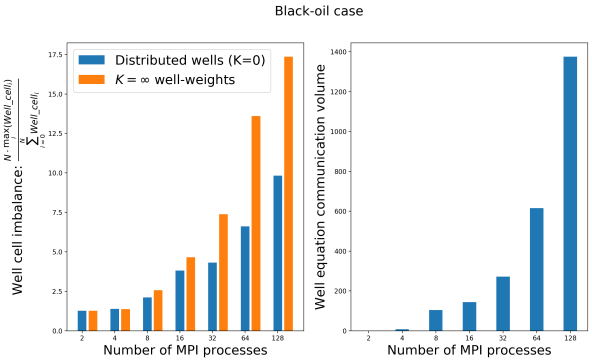


Figure 5: Imbalance of well connected cells when using $K = 0$ and $K = \infty$ well-edge weights in the partitioning scheme (left) and total well equation communication volume (right).

4.1 Well communication overhead

Another point of interest is how the distributed-well implementation impacts the overall runtime. When mesh partitioning produces individual wells being distributed on multiple processes, the parallel implementation needs inter-process communication for well handling and thus induces extra parallel overhead. On the other hand, allowing distributed wells means that well-connected cells can be more evenly distributed among the processes. Figure 5 shows two plots created using the partitioning results of the black-oil case. The first plot compares the imbalance of well-connected cells, when using $K = 0$ and $K = \infty$ well-edge weights. The second plot presents the total communication volume associated with the well equations for different numbers of MPI processes.

In the left plot of Figure 5, we observe that using $K = 0$ well-edge weights yields a better distribution of well-connected cells among the processes, especially for number of processes larger than 32. This means that the chance of well-induced load imbalance is reduced. The second plot of Figure 5 presents the total communication volume associated with evaluating the well contribution of the Schur complement $(\sum_{i=1}^{L} C_i D_i^{-1} B_i)x$ in (7). The need for communication increases with number of MPI processes, but the total well communication volume is negligible compared to the communication volume associated with the fluid equations presented in Figure 4. For a 256-process case, the fluid system communication volume is around 600000, while the well-related volume is around 1500.

4.2 Validation on the Norne black-oil case

To validate the distributed-well implementation, we have conducted a simple experiment on the open Norne reservoir simulation case [2]. We simulate this benchmark twice with *Flow*, both using 16 MPI processes. In the first run, we ensure that individual wells are not split, while in the second simulation we allow distributed wells. To compare the two simulations we consider the well bottom-hole pressures (BHP) of the active wells of the Norne model. In Figure 6 the average absolute difference in the well BHP between the two simulations is presented. Figure 6 also includes a plot presenting the number of subdomains each well is split in, when we allow for distributed wells in our simulation.

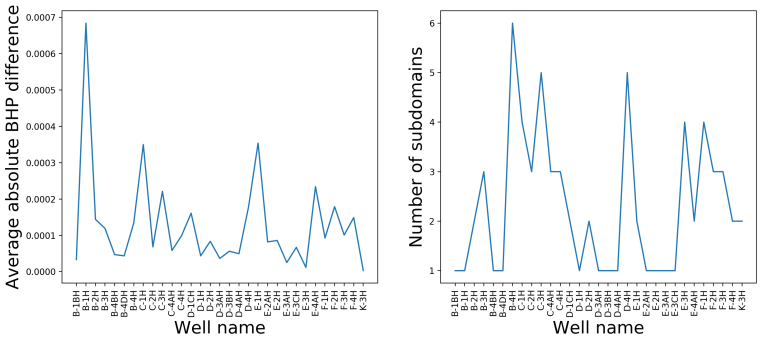
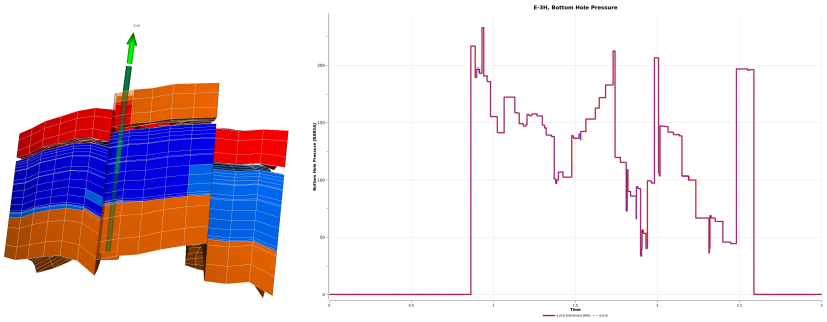


Figure 6: Average absolute difference in well BHP (left) and number of subdomains containing parts of each well (right) for simulations of Norne with 16 processes.

The average absolute difference displayed in Figure 6 varies between $2.82 \cdot 10^{-6}$ and $6.6 \cdot 10^{-4}$, considerably below the linear solver tolerance of 0.01 used in the simulations. Notice also that when we allow for distributed wells in our simulation, the partitioning of the Norne mesh results in the majority of the wells being distributed among two or more process subdomains.

A more detailed look at well E-3H is presented in Figure 7. The plot in Figure 7a shows well E-3H in a slice of the Norne mesh. Cell colors represent different partitions, and we observe that the well perforates cells belonging to four different subdomains. In Figure 7b the BHP curves of the two simulations are plotted against time, where a good agreement between the two simulations can be observed.



(a) (b)

Figure 7: Figure (a) is an illustration of a well that perforates cells on multiple subdomains in the Norne reservoir mesh. Cell colors represent different subdomains of the partitioning. The plot in (b) presents bottom-hole pressure during the simulation when allowing and disallowing distributed wells for the well in (a).

4.3 Simulation results

To illustrate the impact of distributed wells on a parallel reservoir simulation, we have conducted experiments on the black-oil case and the CO_2 case, comparing simulations with $K = 0$ and $K = \infty$. We run *Flow* using 8 to 256 MPI processes on one and two Epyc Milan 7763 CPUs for the black-oil case, and 16 to 512 processes on one to four Epyc Milan 7763 nodes for the CO_2 case. The total simulation execution times for the different numbers of MPI processes are reported in Figure 8.

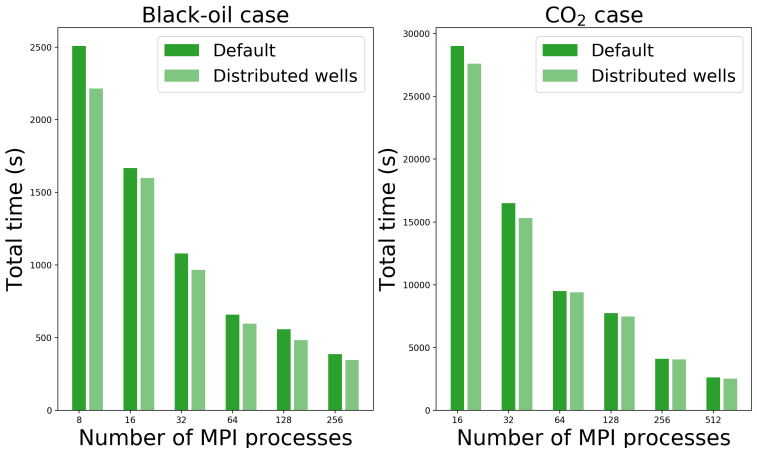


Figure 8: *Flow* execution time of black-oil and CO_2 case for simulation using either $K = \infty$ or $K = 0$ well-edge weights.

In the left plot of Figure 8, we notice that allowing distributed wells ($K = 0$) yields improved performance over discouraging distributed wells ($K = \infty$) for all the MPI-process counts for the black-oil case. However, the difference in the execution time is quite modest (between 4% and 15%), and we do not observe that the impact increases with the number of MPI processes. In the results from the CO_2 case displayed in the right plot of Figure 8, we observe similar results, achieving a slight reduction in execution time for all numbers of MPI processes. To explain the reason for the difference in performance we look at more details from the black-oil simulation in Table 1.

Table 1 contains statistics from *Flow* simulations on the black-oil case. The information displayed in the table includes the number of cells in the largest partition (Max-cells), the total number of linear iterations (Iterations), the total number of Newton iterations (Newton Iter) and the total simulation execution time (Total time).

In the Max-cells row of Table 1 that displays the number of cells of the largest partition after ghost cells are added, we observe that the distributed-well simulations ($K = 0$) yield a lower max-cells number for all numbers of MPI processes. This is not an unexpected result, because the partitioner does not take wells into account, and might therefore result in fewer ghost cells. In fact, the number of ghost cells added to a subdomain of the partitioned reservoir mesh equals the subdomains communication volume defined in (11). As observed in Figure 4, partitioning with $K = 0$ results in lower maximal communication volume than $K = \infty$.

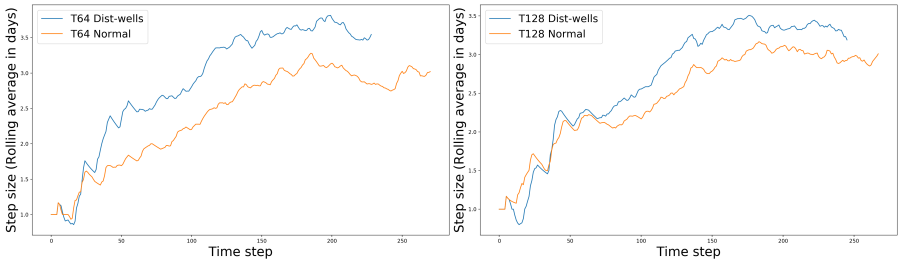
Regarding the total number of linear iterations, the two well-edge weighting schemes produce

Table 1: Detailed performance information from *Flow* simulations on the black-oil case.

	#procs	8	16	32	64	128	256
Max-cells	Default	142322	73811	40442	22643	15189	10346
	Dist-wells	140400	72819	37741	20963	12099	10254
	$\frac{\text{Default}}{\text{Dist-wells}}$	1.01	1.01	1.07	1.08	1.26	1.01
Iterations	Default	18918	17361	17944	17985	18582	19383
	Dist-wells	18307	17939	18117	18949	19290	20359
	$\frac{\text{Default}}{\text{Dist-wells}}$	1.01	0.97	0.99	0.95	0.96	0.95
Newton Iter	Default	2307	2018	2271	2166	2156	1842
	Dist-wells	1987	1970	1996	1902	1952	1650
	$\frac{\text{Default}}{\text{Dist-wells}}$	1.16	1.02	1.14	1.13	1.10	1.12
Total time	Default	2508.44	1667.79	1080.07	658.03	557.09	385.91
	Dist-wells	2214.92	1599.91	966.5	597.1	482.38	345.43
	$\frac{\text{Default}}{\text{Dist-wells}}$	1.13	1.04	1.12	1.10	1.15	1.12

very similar numbers. For 16 processes $K = 0$ gives a lower iteration count, and for 32, 64, 128 and 256 processes $K = \infty$ gives the better result. For Newton iterations the picture is more clear, and we observe that $K = 0$ yields a lower Newton iteration count for all MPI-process counts. The number of Newton iterations has an important impact on the system assembly performance, and seems to be the factor that most likely explains the execution-time difference between simulations with $K = 0$ and $K = \infty$.

The reason for the difference in Newton iterations between $K = 0$ and $K = \infty$ simulations, as observed in Table 1, is that the adaptive time steps are on average larger when partitioning with $K = 0$. This can be observed in Figure 9, where 100-day time step rolling average is displayed for both well-edge weight strategies for 64 and 128 MPI processes.



(a) 64 MPI-processes

(b) 128 MPI-processes

Figure 9: 100-day time step rolling average with *Flow* on black-oil case using 64 (a) and 128 (b) MPI-processes. The blue lines represent simulations using $K = 0$ for well-edge weights, while the orange lines comes from simulations with $K = \infty$ for well-edge weights.

In Figure 9 we notice that the simulations with $K = 0$ have used larger time steps on average than simulations using $K = \infty$ well-edge weights. This again explains the lower Newton iteration count observed for the distributed-well simulations in Table 1. The reason for the larger time

steps in the $K = 0$ case is however not immediately obvious.

Flow allows the user to supply hardcoded time steps. We can therefore look at how the simulator behaves when using time steps generated by the $K = 0$ runs in the $K = \infty$ simulations, as well as the reverse, $K = \infty$ time steps in $K = 0$ simulations. Executing this experiment should demonstrate that the time steps are the reason for the lower Newton iteration count observed in the $K = 0$ simulations. The total Newton iterations required to complete *Flow* simulations of the black-oil case using $K = 0$, $K = \infty$ and $K = \infty$ with $K = 0$ time steps are displayed in Table 2 for 8, 16, 32, 64 and 128 MPI-processes.

Table 2: Total *Flow* Newton iterations when applied to black-oil case using $K = \infty$ (Default), $K = 0$ (Dist-wells) and $K = \infty$ with $K = 0$ time steps (time step).

	$K = \infty$ partitioning		$K = 0$ partitioning	
#procs	Normal	Hardcoded	Normal	Hardcoded
8	2307	2099	1987	2126
16	2018	2031	1970	1947
32	2271	2148	1996	2046
64	2166	2117	1902	2039
128	2156	2049	1952	2075

In Table 2 we observe that using $K = 0$ time steps in $K = \infty$ simulations yields fewer Newton iterations than the original $K = \infty$ simulations for all the displayed numbers of processes, except 16. Similarly, hardcoded $K = \infty$ time steps yield more Newton iterations than the original time steps, when partitioning with $K = 0$ well-edge weights (again the case of16 processes is the exception). The 16-process simulations have the smallest gap between the default and distributed-well simulations.

5 Parallel preconditioner performance

In this section we compare the parallel performance of different preconditioner options in the *Flow* reservoir simulator when applied to the 1-million cell black-oil and 9-million cell CO_2 cases that are described in Section 1.1. (All the numerical experiments presented in Section 4 have used the parallel block-Jacobi preconditioner with ILU0 as the subdomain solver.) We will first look at the performance of the solver on one linear system each from the black-oil and CO_2 cases, and then the impact of preconditioner choice on the full simulations. We will consider three preconditioner options provided by *Flow*, namely block-Jacobi with subdomain ILU0, AMG and CPR. Both the stand-alone AMG and the two-level CPR-AMG pressure preconditioners use ILU0 smoothers, and perform a single pre- and post-smooth step, as shown in Figure 2. More details can be found in Section 2.4.

In the two-level CPR and stand-alone AMG preconditioners a hierarchy of coarser linear systems are created. To compare the computational cost of AMG and CPR, we look at the size of the coarse systems in the hierarchy. We consider the AMG hierarchies extracted from the three-phase black-oil and two-phase CO_2 cases. The size of the fine and coarse systems in the stand-alone AMG preconditioner and CPR's AMG pressure solver is presented in Table 3. For the black-oil case, both the AMG hierarchies consist of 7 levels, and on the coarsest level, the AMG and CPR-AMG linear systems have 8190 and 2448 degrees of freedoms (DoFs), respectively. These are very small numbers compared to the size of the finest systems, and therefore marginally

Table 3: Numbers of DoFs in the CPR and AMG hierarchies. For CPR total, fine level, pressure equation fine AMG level and coarser level DoFs are shown. For AMG total, fine level and coarser level DoFs are displayed.

Case	Black-oil		CO_2	
PreCon	AMG	CPR	AMG	CPR
Fine CPR	N/A	3278169	N/A	18183632
Level 0	3278169	1092723	18183632	9091816
Level 1	838926	293321	4955196	2477598
Level 2	221298	89241	1562772	781386
Level 3	63258	29043	470368	235184
Level 4	22200	8938	115938	57969
Level 5+	19452	6742	40880	20440
Total	4443303	4798177	25328786	30848025

contribute to the total computational cost of AMG and CPR. In Table 3, we only display the size of the 5 finest levels of the AMG hierarchies and the sum of the sizes of the remaining coarse systems.

In Table 3, we observe that CPR yields a slightly larger total number of DoFs in the hierarchy of linear systems than AMG. For both preconditioners and both cases, it is the finest level that makes up the largest portion of the total number of DoFs. However, when constructing CPR for the CO_2 system, the difference between the fine-level and coarse-level DoFs is significantly lower. We use the same smoother and the same number of pre- and post-smoothing steps in AMG and CPR, and therefore expect each CPR preconditioning operation to be slightly more computationally expensive than AMG.

The last row in Table 3, i.e., the total number of DoFs, can be used to estimate the relative computational cost of applying AMG and CPR. Specifically, we want to estimate the cost of AMG and CPR with respect to block-Jacobi-ILU0. The majority of the computational time of applying AMG and CPR is made up of smoother operations. In this paper, we always apply the smoother twice on each levek, as one pre-smoothing and one post-smoothing step. The smoother operation consists of several computations, with the two most costly being the smoother preconditioner (ILU0 in our case) and one SpMV multiplication.

Let T_{AMG} denote the computational cost of applying AMG, and let T_{ILU0_ℓ} and T_{SpMV_ℓ} be the computational costs of executing one ILU0 backward/forward sweep and one SpMV operation on hierarchy level ℓ. For our AMG setup, T_{AMG} can then be expressed as:

$$T_{\text{AMG}} \approx \sum_{\ell=0}^{\ell_{\max}-1} 2(T_{\text{ILU0}_\ell} + T_{\text{SpMV}_\ell}), \tag{12}$$

where $\ell_{\max}$ represents the coarsest level in the hierarchy. On that level the system is solved with a direct solver and the smoother operations are not used. To relate the above expression to a cost estimate for ILU0, we make two assumptions. First, we assume that $T_{\text{ILU0}_\ell} \approx \frac{N_\ell}{N_0} T_{\text{ILU0}_0}$ and $T_{\text{SpMV}_\ell} \approx \frac{N_\ell}{N_0} T_{\text{SpMV}_0}$. Second, $T_{\text{ILU0}_0} \approx T_{\text{SpMV}_0}$ is assumed. Here, N_ℓ denotes the number of DoFs in the system on level ℓ in the AMG hierarchy. Inserting these assumptions into (12),

we get:

$$T_{\text{AMG}} \approx 2\frac{\sum_{\ell=0}^{\ell_{\max}-1} N_\ell}{N_0}\left(T_{\text{ILU0}_0} + T_{\text{SpMV}_0}\right) \approx 4\frac{N_{tot}}{N_0}T_{\text{ILU0}_0}. \tag{13}$$

Here, we define $N_{tot} = \sum_{\ell=0}^{\ell_{\max}-1} N_\ell$. By using (13) and the last row in Table 3, we can estimate that the computational cost of applying AMG each time is at least 5.4 and 5.6 times more expensive than ILU0 for the black-oil and CO_2 models, respectively. The same estimates for CPR are 5.8 and 6.8. In addition to the two smoothing steps on each level, the direct solver at the coarsest level, as well as the level transfer step, contribute to the computational cost of AMG and CPR. To get an accurate picture of the relative computational cost of ILU0, AMG and CPR, we compare their measured time of applying $y = M^{-1}x$ for an increasing number of processes. The measurements are presented in Figure 10.

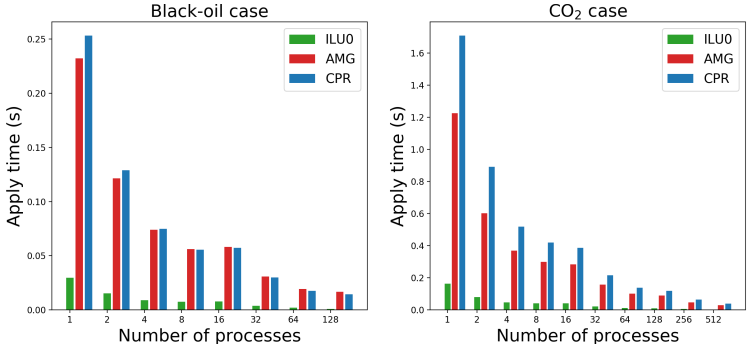


Figure 10: Execution time of ILU0, AMG and CPR preconditioners applied to a black-oil case system (left) and a CO_2 system (right) for increasing number of MPI processes.

The results in Figure 10 show that CPR, as expected, is a slightly more expensive operation than AMG, and that the relative difference is larger for the CO_2 case than the black-oil case. AMG and CPR are both significantly more expensive than ILU0. AMG is between 7.4 and 8.1 times slower than ILU0 in both cases, while CPR is 7.5 to 8.5 times slower than ILU0 for the black-oil case, and 9.7 to 10.7 times slower for the CO_2 case system. This is larger than the estimate in (13), which does not consider level transfer cost, direct solve on coarsest level or that SpMV can be slightly more expensive than ILU0.

Setting up the linear systems down through the levels of the AMG and CPR hierarchies is relatively expensive compared to the set-up time for ILU0. On every level of the AMG hierarchy, except the coarsest, an agglomeration step is performed so one can set up the system on the coarser level. The set-up step also includes a direct solver LU-decomposition on the coarsest level, as well as constructing the smoother on the finer levels of the hierarchy. In parallel, new communication patterns must be created between processes on each level. To illustrate the set-up cost, we include plots of preconditioner set-up time in Figure 11.

The equations in the black-oil and CO_2 storage models are time-dependent and non-linear. To evolve our simulation from one time-step to the next, we need to perform multiple Newton iterations, where each iteration involves solving a system of linear equations. Exploiting that the non-zero pattern remains constant throughout the simulation, we can reuse parts of the AMG and CPR set-up to save time. When reusing the set-up, updating AMG and CPR only consists

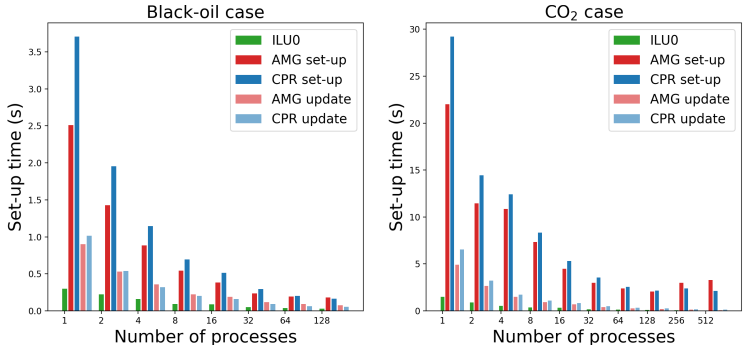


Figure 11: Time taken to construct the ILU0, AMG and CPR preconditioners for the black-oil and CO_2 systems.

of constructing the smoothers and a coarsest level LU-decomposition. Preconditioner update time is included for AMG and CPR in the plots in Figure 11.

In the plots displayed in Figure 11, we observe that the CPR and AMG set-up time is significantly more than the ILU0 set-up time. For both models, the CPR set-up is more than 10 times slower than ILU0 for all MPI process counts. Moreover, we also observe that the set-up time scales poorly when using more than 128 processes for the CO_2 case. However, the execution time of updating AMG and CPR, by reusing an old set-up, is a much faster operation, and for CPR, is only 3 to 4 times slower than the ILU0 set-up. The update operation also scales when using more than 128 processes.

The relative difference in the preconditioner execution time and set-up time observed in Figures 10 and 11, means that CPR and AMG only compare favourable to ILU0 if they result in significantly fewer linear solver iterations. The computational cost of the iterative linear solvers, such as CG, GMRES or BiCGStab, mainly consists of SpMV operations and applying the preconditioning operations. In one BiCGStab iteration, SpMV and preconditioner are both performed and applied twice. Using the expression in (13), we can estimate the relative cost of an AMG preconditioned BiCGStab iteration with respect to an ILU0 preconditioned BiCGStab iteration. We derive this estimate by dividing the approximate cost of an AMG preconditioned BiCGStab iteration $2(T_{\text{AMG}} + T_{\text{SpMV}})$ by the approximate cost of an ILU0 preconditioned BiCGStab iteration $2(T_{\text{ILU0}} + T_{\text{SpMV}})$:

$$\frac{2(T_{\text{AMG}} + T_{\text{SpMV}})}{2(T_{\text{ILU0}} + T_{\text{SpMV}})} \approx \frac{4\frac{N_{tot}}{N_0} + 1}{2} \tag{14}$$

The above expression states that an AMG preconditioned BiCGStab iteration is approximately $\frac{4\frac{N_{tot}}{N_0}+1}{2}$ times more costly than an ILU0 preconditioned BiCGStab iteration. Based on the measurements in Figure 10, we therefore expect an AMG preconditioned solver to be faster than an ILU0 preconditioned solver, if the latter requires at least 5-6 times more iterations to converge. We consider two linear systems extracted respectively from the black-oil and CO_2 cases. The two systems were extracted from sequential simulations of the black-oil and CO_2 cases, and were chosen because the ILU0 preconditioned solver needed many iterations to solve them. The change in the relative residual as a function of time when using an ILU0, AMG or CPR preconditioned solver to solve the two systems with one and 128 MPI processes is presented

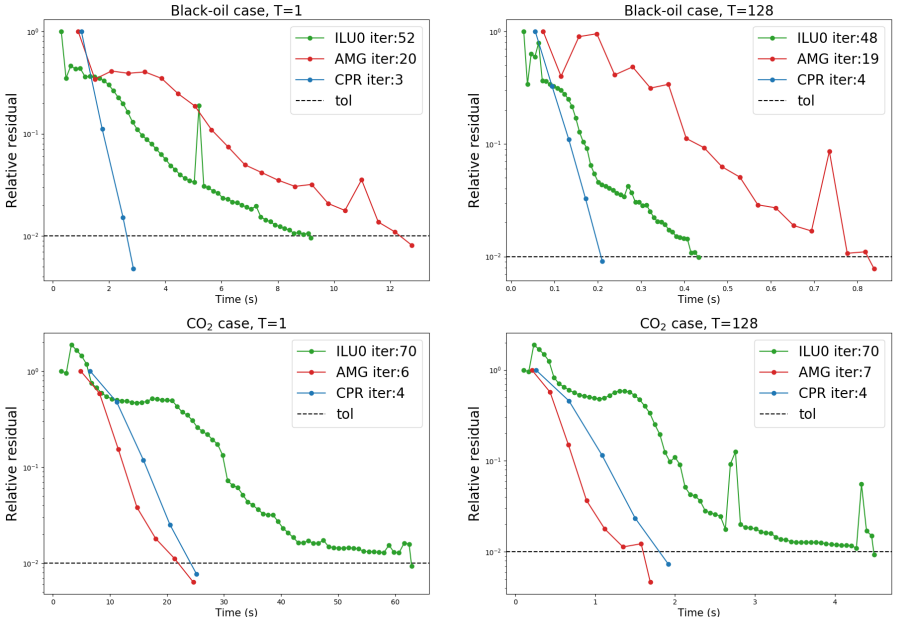


Figure 12: Relative residual reduction against time for systems extracted from the black-oil and CO_2 cases. Each dot represents a BiCGStab iteration.

in Figure 12. The plots are dotted, and each dot represent a BiCGStab iteration. The update time is added to the solver time for a fair comparison between the solvers. We consider the systems solved when the residual is reduced by a factor of 0.01.

The measurements from the black-oil and CO_2 cases, showcased in Figure reffig:residual, present different pictures of the preconditioner performance. For the black-oil case, CPR outperforms ILU0 whereas ILU0 outperforms AMG. For the CO_2 case, AMG achieves a residual reduction below the given tolerance first, but the CPR performance is not far behind AMG. The relative preconditioner performance is more or less the same when using 1 and 128 MPI processes. The requency of dots, observed for both bases, show that the ILU0 preconditioned solver requires more iterations than AMG and CPR to reach the convergence tolerance. However, as the results in Figures 10 and 11 show, the distance between dots are smaller for ILU0, and a lower update time works in its favour. Notice also that although all the three preconditioners reach convergence, AMG and CPR achieve a larger reduction in the residual beyond the tolerance.

5.1 Full simulation results

We now consider the performance of full simulations of the black-oil and CO_2 model cases. We compare the execution times of parallel *Flow* simulations when using ILU0, AMG and CPR preconditioned linear solvers. All experiments are conducted on a small cluster of dual-EPYC Milan 7763 64-core CPUs. In all our experiments we only carry out the preconditioner set-up once for AMG and CPR, and then reuse the initial CPR pressure system set-up and AMG

hierarchies through the whole simulation.

5.1.1 Black-oil model

The execution time and total number of linear iterations for simulations of the black-oil case are displayed in Figure 13. Simulations are executed with 8, 16, 32, 64, 128, 256 and 512 MPI processes.

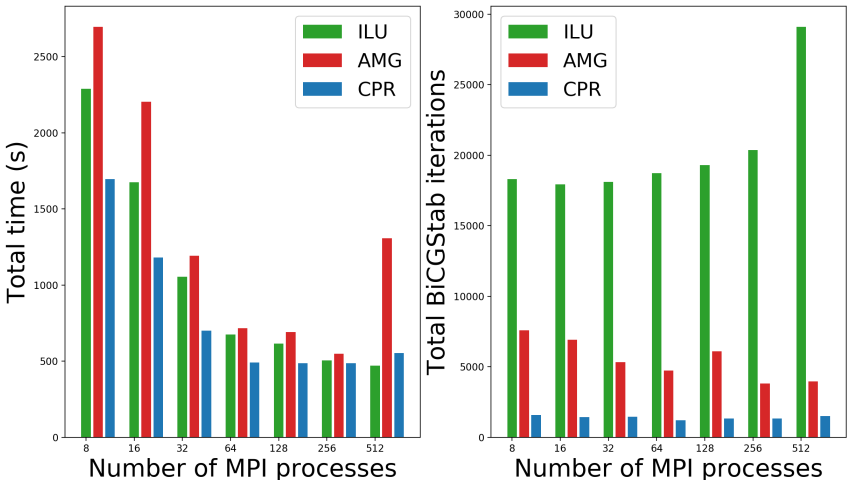


Figure 13: Execution time and total number of linear iterations for parallel simulations of the black-oil case.

The results displayed in Figure 13 demonstrate that using CPR yields better overall performance than ILU0 when simulating the black-oil case for the number of processes lower than 256. CPR results in lower total number of linear iterations for all MPI process counts. The ILU0 preconditioned simulations require around 10 times more linear iterations to complete than the CPR preconditioned simulations. Although simulations with the AMG preconditioner need fewer linear iterations than the ILU0 simulations, ILU0 yields faster execution time for all process counts. The only exception concerns the 512-process simulations, where the AMG needs 2.5-3.5 fewer linear iterations than the ILU0 simulations. This advantage of fewer linear iterations is not sufficient to offset the larger preconditioner cost, and explains why the ILU0 simulation outperforms AMG.

Note that the scaling past 128 processes is quite poor for all the choices of preconditioners, but particularly for AMG and CPR, where we observe an increase in the execution time when using 512 processes. For AMG and CPR, the particularly bad performance observed for 512 processes is caused by the relatively poor results of the AMG coarsening algorithm. The coarsest systems in the AMG and CPR hierarchies have 62511 and 24288 DoFs, compared to around 7000 and 3000 DoFs when using fewer processes. The increase in the execution time from 256 to 512 processes is caused by the direct solver of the coarsest system.

A detailed overview of where time was spent in the simulation of the black-oil case is presented in Table 4, where we display how much time *Flow* spends on assembling the linear systems and

Table 4: Total time spent on the system assemblies and linear solvers when simulating the black-oil case with an ILU0, AMG and CPR preconditioned solver. The table also presents Newton iterations, linear iterations and average single system assembly time.

PreCon		8	16	32	64	128	256	512
ILU0	Single assemble	0.412	0.21	0.114	0.61	0.042	0.031	0.022
	Newton iterations	1987	1970	1996	1902	1952	1650	1442
	Assembly time	989.57	526.77	300.09	161.97	117.83	106.24	95.58
	Linear iterations	18307	17939	18117	18735	19290	20359	29110
	Solver time	878.28	815.21	455.65	266.99	207.63	107.56	70.7
AMG	Single assemble	0.427	0.22	0.127	0.071	0.043	0.032	0.022
	Newton iterations	2359	1947	1861	1859	1656	1558	1380
	Assembly time	1075.54	478.11	265.66	162.24	101.14	94.74	84.6
	Linear iterations	7604	6930	5331	4739	4368	3832	3969
	Solver time	1557.16	1230.93	558.55	356.15	275.22	183.97	883.1
CPR	Single assemble	0.425	0.22	01.27	0.07	0.045	0.031	0.021
	Newton iterations	1415	1465	1369	1273	1480	1291	1073
	Assembly time	729.11	405.10	220.43	119.45	94.35	88.99	70.75
	Linear iterations	1580	1429	1454	1225	1350	1818	1522
	Solver time	634.72	499.7	257.35	151.05	141.06	117.68	207.27

how much is on solving the systems, when using ILU0, AMG or CPR preconditioning. We report the average time spent on assembling each system, the number of Newton iterations, total assembly time, number of linear iterations and total solver time.

In Table 4 we notice that the time needed to assemble a single system is roughly the same for all the choices of preconditioners. This is expected since the preconditioner does not affect the system assembly. However, the AMG and CPR preconditioned simulations require lower total assembly time because of fewer Newton iterations needed. As can be observed in Figure 12, the AMG and CPR preconditioned solvers achieve higher reductions in the relative residual beyond the tolerance. This leads to faster convergence for the Newton solver.

The total number of linear iterations increases with the number of processes for ILU0, but not for AMG and CPR. Despite this, the time spent on solving the linear systems continues to decrease for the simulations with ILU0 preconditioned solver, while we observe poor scaling in the solver time above 128 processes for AMG and CPR. The lack of reduction in the solver time between 8 and 16 processes, observed for all preconditioners, can be explained by the mapping of MPI processes in relation to the hardware memory channels. Increasing from 8 to 16 processes, the memory bandwidth available is however unchanged. Since the linear solver is bound by the memory bandwidth, little to no improvement in the solver time is achieved.

5.1.2 CO_2 storage model

A similar comparison for the CO_2 case is displayed in Figure 14, where we present the results for simulations executed with ILU0, AMG and CPR preconditioning. The plots in Figure 14 display the simulation execution time and total number of linear iterations for 16, 32, 64, 128, 256 and 512 MPI processes.

The results presented in Figure 14 show that AMG preconditioning outperforms ILU0 and

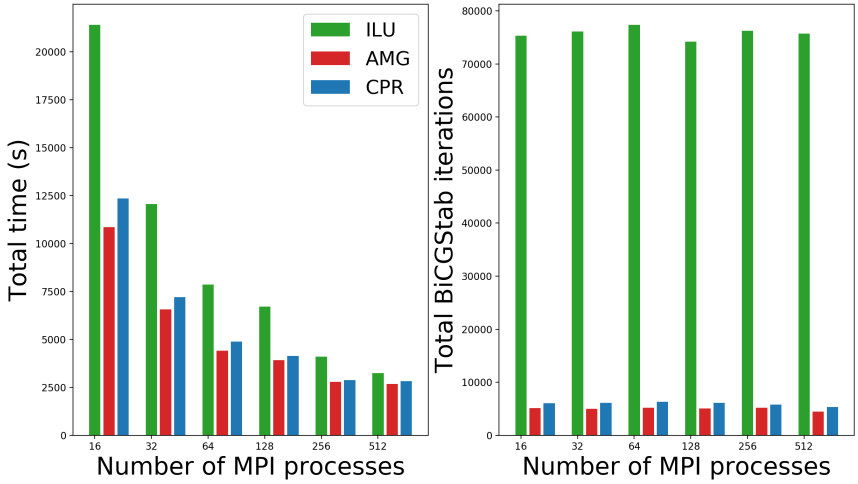


Figure 14: Simulation execution time and total number of linear iterations for simulation of the CO_2 case using ILU0, AMG and CPR preconditioned linear solvers.

CPR preconditioning for all number of processes when simulating the CO_2 case. We observe that both AMG and CPR need significantly fewer linear iterations than ILU0 to complete the simulations, a factor of around 14 to 16 for all the process numbers. The execution time for the AMG preconditioned simulation, when using 16 processes, is about 50% lower than the ILU0 preconditioned simulation. The comparative improvement in the execution time is reduced for larger numbers of processes, we only observe 10% lower simulation time for 512 MPI processes.

A detailed overview of where time was spent in the simulation of the CO_2 case is presented in Table 5. Here, we display how much time *Flow* has spent on assembling the linear systems and how much has been spent on solving the systems, when using the three preconditioners. We report the average time usage per system assembly, Newton iterations, total assembly time, number of linear iterations and total solver time.

The performance metrics presented in Table 5 show that the time spent on the linear solver determines the overall performance. The AMG preconditioned simulation outperforms CPR and ILU0 because of the faster convergence, which is reflected by a lower iteration count, as observed in Figure 14.

5.2 Comparing the performance of *Flow* and *Eclipse* 300 on the CO_2 storage model

We compare the parallel performance of *Flow* on the CO_2 model with the commercial alternative *Eclipse* 300. These simulations were conducted on a server with dual socket Intel E5-2687W CPUs. For *Flow*, we used the AMG preconditioner in the linear solver, while *Eclipse* 300 uses nested factorization. Figure 16 displays the execution time of *Flow* and *Eclipse* 300 when applied to the 9-million cell CO_2 case.

Figure 16a displays the execution time of *Eclipse* 300 for 1, 2, 4, 8 and 16 MPI processes, while Figure 16b displays the same for *Flow* together with the 16-process *Eclipse* 300 time usage.

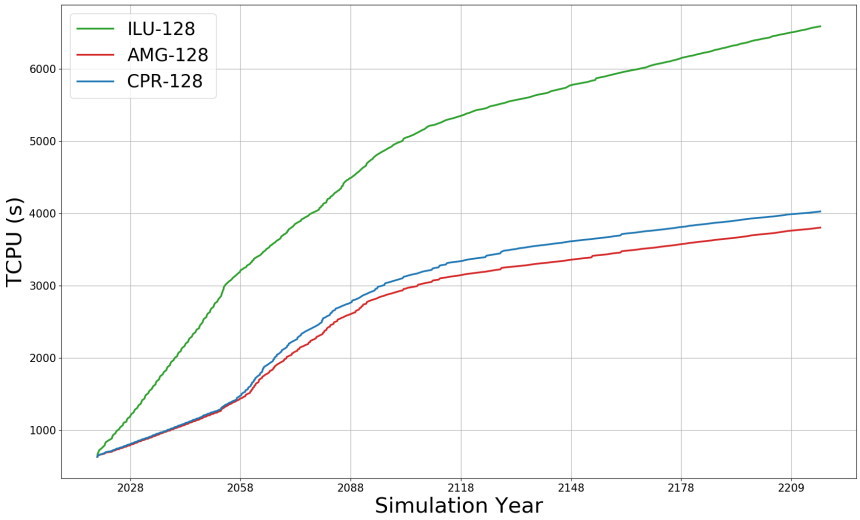
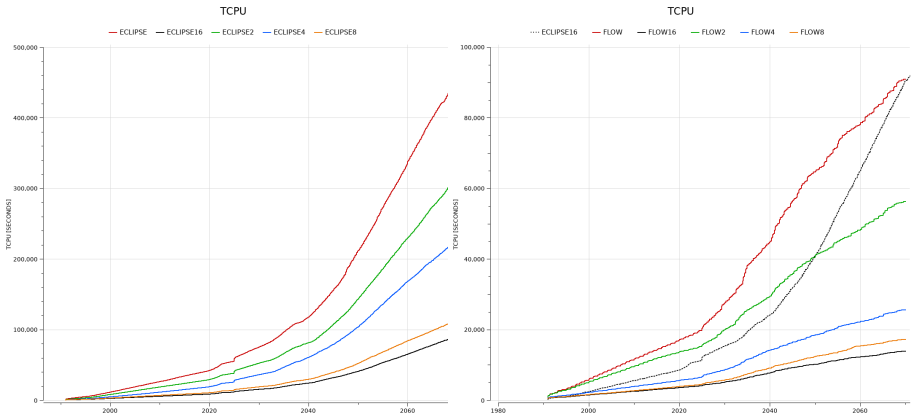


Figure 15: Reported CPU time at each time-step of the CO_2 case simulation when using a ILU0, AMG and CPR preconditioned solver and 128 MPI-processes.

Table 5: Total time spent on the system assemblies and linear solvers when simulating the CO_2 case with an ILU0, AMG and CPR preconditioned solver. The table also presents Newton iterations, linear iterations and average single system assembly time.

PreCon		16	32	64	128	256	512
ILU0	Single assemble	0.44	0.25	0.15	0.12	0.061	0.051
	Newton iterations	2986	2892	2859	2965	3071	2906
	Assembly time	1537.3	848.8	504.9	401.5	216.4	171.9
	Linear iterations	75301	76088	77378	74177	76238	75727
	Solver time	16781.3	8885.3	5597.9	4698.01	2326.96	1177.7
AMG	Single assemble	0.44	0.25	0.15	0.11	0.062	0.048
	Newton iterations	2764	2831	2875	2943	2999	2838
	Assembly time	1411.3	830.2	504.6	378.6	212.5	157.8
	Linear iterations	5113	5031	5190	5105	5218	4494
	Solver time	6224.81	3431.48	2169.14	1927.73	1073.49	651.68
CPR	Single assemble	0.44	0.26	0.15	0.11	0.065	0.048
	Newton iterations	3000	3003	2999	2953	2833	2842
	Assembly time	1515.5	882.1	531.7	361.3	212.8	157.8
	Linear iterations	6092	6123	6319	6145	5774	5322
	Solver time	7441.5	4048.5	2605.0	2219.1	1163.0	788.5



(a) (b)

Figure 16: Simulation execution time of the CO_2 model using *Flow* and *Eclipse* 300 for 1, 2, 4, 8 and 16 MPI processes. The plot in (a) displays only the *Eclipse* 300 results, while the plot in (b) shows the *Flow* timings plus the *Eclipse* 300 timing using 16 MPI processes.

Both simulators achieve good parallel scaling up to 8 processes. However, the performance of *Flow* is much better than that of *Eclipse* 300. Specifically, the execution time of a single-process simulation in *Flow* is equivalent to that of a 16-process simulation in *Eclipse* 300.

6 Conclusion

The work presented in this paper touches on several aspects of the parallel performance of reservoir simulations. In particular, the first major aspect of the work addresses how to allow individual wells to be distributed across multiple subdomains in a parallel simulator and the resulting performance impact. The distributed-well implementation enabled in the *Flow* reservoir simulator is presented, as well as the details about the related Schur complement and how parallel well setup is handled.

With a distributed-well implementation we can simplify the mesh partitioning step, because the cells perforated by a well no longer need to be coupled together using infinity-weighted edges in the partitioning graph. Through experiments on a million cell black-oil model and a 9-million cell CO_2 storage model, we demonstrate a positive impact of distributed wells. We consider both the communication overhead and overall performance. For the black-oil model, allowing distributed wells results in a significant reduction in the communication volume. Although additional well-related communication overhead is introduced, it is compensated by the better partitioning quality. We also observe improved load balancing of well-perforated cells among the subdomains.

For the overall performance, we observe a reduction in the execution time for simulations that allow distributed wells. This improvement is larger for the black-oil model, which contains more active wells than the CO_2 model. The reduced simulation time can not be explained by the reduction in communication volume alone. Instead, we argue that the improved performance is caused by distributed wells somehow resulting in more efficient time stepping.

The second major aspect of the paper is a study on how the choice of preconditioner for the linear solver impacts parallel reservoir simulation performance. We consider three options, ILU0, AMG and CPR, and compare their performance for simulations of the black-oil and CO_2 storage models. Estimates of the relative computational costs of AMG and CPR compared to ILU0 are used to explain the performance of the simulations of the two models. We observe that the optimal choice of preconditioner differs between the two cases, with CPR producing the fastest simulations for the black-oil model, while the AMG preconditioner outperforming both ILU0 and CPR on the CO_2 model.

References

[1] T. H. Sandve, A. B. Rustad, A. Thune, B. Nazarian, S. E. Gasda, and A. F. Rasmussen, "Simulators for the gigaton storage challenge. A benchmark study on the regional Smeaheia model.," in *EAGE GeoTech 2022 Sixth EAGE Workshop on CO2 Geological Storage*, vol. 2022, pp. 1–5, European Association of Geoscientists & Engineers, 2022.

[2] "Norne reservoir simulation benchmark." https://github.com/OPM/opm-data, 2019.

[3] J. A. Holmes, "Enhancements to the strongly coupled, fully implicit well model: wellbore crossflow modeling and collective well control," in *SPE Reservoir Simulation Symposium*, OnePetro, 1983.

[4] Z. Chen, G. Huan, and Y. Ma, *Computational Methods for Multiphase Flows in Porous Media*, vol. 2. Philadelphia: SIAM, 2006.

[5] K. Aziz and A. Settari, *Petroleum Reservoir Simulation*. London: Applied Science Publishers, 1979.

[6] C. A. Chase and M. R. Todd, "Numerical simulation of CO2 flood performance (includes associated papers 13950 and 13964)," *Society of Petroleum Engineers Journal*, vol. 24, no. 06, pp. 597–605, 1984.

[7] A. F. Rasmussen, T. H. Sandve, K. Bao, A. Lauser, J. Hove, B. Skaflestad, R. Klöfkorn, M. Blatt, A. B. Rustad, O. Sævareid, K.-A. Lie, and A. Thune, "The Open Porous Media Flow reservoir simulator," *Computers & Mathematics with Applications*, vol. 81, pp. 159–185, 2021.

[8] T. H. Sandve, A. F. Rasmussen, and A. B. Rustad, "Open reservoir simulator for CO2 storage and CO2-EOR," in *14th Greenhouse Gas Control Technologies Conference*, 2018.

[9] T. H. Sandve, S. E. Gasda, A. F. Rasmussen, and A. B. Rustad, "Convective dissolution in field scale CO2 storage simulations using the OPM Flow simulator," in *TCCS–11. CO2 Capture, Transport and Storage. Trondheim 22nd–23rd June 2021 Short Papers from the 11th International Trondheim CCS Conference*, SINTEF Academic Press, 2021.

[10] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, M. Ohlberger, and O. Sander, "A generic grid interface for parallel and adaptive scientific computing. Part I: abstract framework," *Computing*, vol. 82, no. 2, pp. 103–119, 2008.

[11] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, R. Kornhuber, M. Ohlberger, and O. Sander, "A generic grid interface for parallel and adaptive scientific computing. Part II: implementation and tests in DUNE," *Computing*, vol. 82, no. 2-3, pp. 121–138, 2008.

[12] M. Blatt and P. Bastian, "The Iterative Solver Template Library," in *Applied Parallel Computing. State of the Art in Scientific Computing* (B. Kågström, E. Elmroth, J. Dongarra, and J. Waśniewski, eds.), vol. 4699 of *Lecture Notes in Computer Science*, pp. 666–675, Springer, 2007.

[13] M. Blatt and P. Bastian, "On the Generic Parallelisation of Iterative Solvers for the Finite Element Method," *Int. J. Comput. Sci. Engrg.*, vol. 4, no. 1, pp. 56–69, 2008.

[14] D. W. Peaceman, "Interpretation of well-block pressures in numerical reservoir simulation (includes associated paper 6988)," *Society of Petroleum Engineers Journal*, vol. 18, no. 03, pp. 183–194, 1978.

[15] A. S. Williamson and J. E. Chappelear, "Representing wells in numerical reservoir simulation: Part 1 – Theory," *Society of Petroleum Engineers Journal*, vol. 21, no. 03, pp. 323–338, 1981.

[16] H. A. van der Vorst, "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems," *SIAM Journal on scientific and Statistical Computing*, vol. 13, no. 2, pp. 631–644, 1992.

[17] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 3, pp. 856–869, 1986.

[18] M. Blatt, O. Ippisch, and P. Bastian, "A massively parallel algebraic multigrid preconditioner based on aggregation for elliptic problems with heterogeneous coefficients," *arXiv preprint arXiv:1209.0960*, 2012.

[19] M. Blatt, *A parallel algebraic multigrid method for elliptic problems with highly discontinuous coefficients*. PhD thesis, Universtität Heidelberg, 2010.

[20] R. Scheichl, R. Masson, and J. Wendebourg, "Decoupling and block preconditioning for sedimentary basin simulations," *Computational Geosciences*, vol. 7, no. 4, pp. 295–318, 2003.

[21] A. Thune, X. Cai, and A. B. Rustad, "On the impact of heterogeneity-aware mesh partitioning and non-contributing computation removal on parallel reservoir simulations," *Journal of Mathematics in Industry*, vol. 11, no. 1, pp. 1–23, 2021.

[22] J. K. Cullum, K. Johnson, and M. Tůma, "Effects of problem decomposition (partitioning) on the rate of convergence of parallel numerical algorithms," *Numerical Linear Algebra with Applications*, vol. 10, no. 5-6, pp. 445–465, 2003.

[23] E. Vecharynski, Y. Saad, and M. Sosonkina, "Graph partitioning using matrix values for preconditioning symmetric positive definite systems," *SIAM Journal on Scientific Computing*, vol. 36, no. 1, pp. A63–A87, 2014.

Appendices

Appendix A

The Open Porous Media Flow reservoir simulator

Atgeirr Flø Rasmussen, Tor Harald Sandve, Kai Bao, Andreas Lauser, Joakim Hove, Bård Skaflestad, Robert Klöfkorn, , Markus Blatt, Alf Birger Rustad, Ove Sævareid, Knut-Andreas Lie, Andreas Thune

Computers and Mathematics with Applications 81 (2021) 159–185

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

The Open Porous Media Flow reservoir simulator

Atgeirr Flø Rasmussen [a], Tor Harald Sandve [b], Kai Bao [a], Andreas Lauser [c], Joakim Hove [d], Bård Skaflestad [a], Robert Klöfkorn [b,*], Markus Blatt [d], Alf Birger Rustad [e], Ove Sævareid [b], Knut-Andreas Lie [a], Andreas Thune [f]

[a] SINTEF Digital, Norway
[b] NORCE Norwegian Research Centre AS, Norway
[c] Poware Software Solutions, Germany
[d] OPM-OP AS, Norway
[e] Equinor ASA, Norway
[f] Simula Research Laboratory, Norway

ARTICLE INFO

Article history:
Available online 11 June 2020

ABSTRACT

The Open Porous Media (OPM) initiative is a community effort that encourages open innovation and reproducible research for simulation of porous media processes. OPM coordinates collaborative software development, maintains and distributes open-source software and open data sets, and seeks to ensure that these are available under a free license in a long-term perspective.

In this paper, we present OPM Flow, which is a reservoir simulator developed for industrial use, as well as some of the individual components used to make OPM Flow. The descriptions apply to the 2019.10 release of OPM.

1. Introduction

The Open Porous Media (OPM) initiative was started in 2009 to encourage open innovation and reproducible research on modeling and simulation of porous media processes. OPM was initially founded as a collaboration between groups at Equinor (formerly Statoil), SINTEF, the University of Stuttgart, and the University of Bergen, but over time, several other groups and individuals have joined and contributed. What today forms the OPM suite of software, has mainly been developed by SINTEF, NORCE (formerly IRIS), Equinor, Ceetron Solutions, Poware Software Solutions, and Dr. Blatt HPC-Simulation-Software & Services.

The initial vision was to create long-lasting, efficient, and well-maintained, open-source software for simulating flow and transport in porous media. The scope has later been extended to also provide open data sets, thus making it easier to benchmark, compare, and test different mathematical models, computational methods, and software implementations.

OPM has so far primarily focussed on tools for reservoir simulation, and consists today of the OPM Flow reservoir simulator, upscaling tools, and a selection of supporting and experimental software pieces. The corresponding source code is divided into six modules, as shown in Fig. 1, and is organized as a number of git repositories hosted on github.com/OPM. The data repositories opm-data and opm-tests are also hosted there, and contain example cases and data used for integration testing. Herein, we only give an in-depth description of OPM Flow, which in turn uses or builds on existing frameworks and libraries such as Dune [1], DuMuX [2], Zoltan [3], and Boost to reduce implementation and maintenance

* Corresponding author.
E-mail address: robert.kloefkorn@norceresearch.no (R. Klöfkorn).

https://doi.org/10.1016/j.camwa.2020.05.014

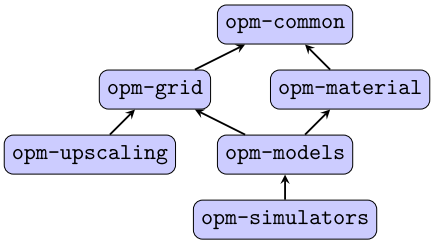


Fig. 1. Current module structure as of the 2019.10 release. Arrows indicate inter-module dependencies: a module depends on another module if it has an arrow pointing towards it. Since the OPM software suite is in active development, the module organization is likely to keep changing in the future.

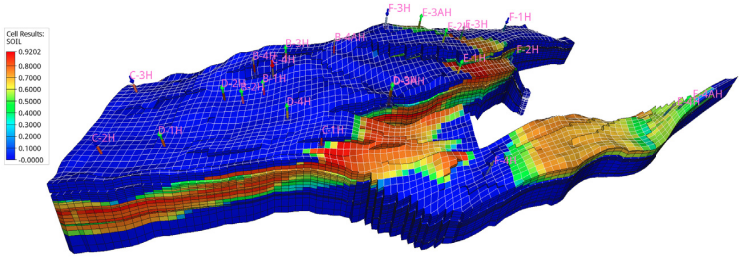


Fig. 2. The Norne reservoir showing oil saturation and wells after approximately 3.5 years of production. The z-direction has been exaggerated 5 times.

cost and improve software quality. The Matlab Reservoir Simulation Toolbox [4–6] has also been a significant source of ideas and concepts. In addition to the simulator tools, OPM contains the ResInsight postprocessor, which has been used for all the 3D visualization herein, such as Fig. 2. Apart from computational routines underlying flow diagnostics [7], ResInsight is independent of the other OPM modules.

All software in OPM is distributed under the GNU General Public License (GPL), version 3, with the option of using newer versions of the license. Datasets are distributed under the Open Database License, version 1.0, with individual content under the Database Contents License, version 1.0.

Code development in OPM follows an open model: All source code contributions are made using the GitHub pull request mechanism. Anyone can create such a request, which asks the maintainers to merge some change (bugfix, improvement, or new feature) into the master branch of one of the module repositories. Each request is reviewed and typically merged by the maintainer after some discussion and possible revisions of the proposed changes. OPM generally accepts contributions of many kinds, not just software improvements and additions, but also use-case reports, documentation, and bug reports.

2. OPM Flow

OPM Flow is developed to serve two main purposes: (1) as an open-source alternative and drop-in replacement for contemporary commercial simulators to assist oil companies in planning and managing (conventional) hydrocarbon assets, and (2) as a research and prototyping platform that enables its users to implement, test, and validate new models and computational methods in a realistic and industry-relevant setting. To this end, OPM Flow aims to represent reservoir geology, fluid behavior, and description of wells and production facilities as in commercial simulators and hence offers standard fully-implicit discretizations of black-oil type models and supports industry-standard input and output formats. The simulator is implemented using automatic differentiation (AD) [8] to avoid error-prone derivation and coding of analytical Jacobians for the residual equations, which also makes it easier to extend the simulator with new model equations, e.g., for CO_2 injection or enhanced oil recovery (EOR). Equally important, using AD makes it relatively simple to extend the simulator to compute sensitivities and gradients with respect to various types of model parameters, using e.g., an adjoint method [9].

This paper does not aim to explain how to install or run OPM Flow. For that we refer to the web site, opm-project.org, or the OPM Flow manual [10].

2.1. The black-oil model

The black-oil equations constitute the most widely used flow model in simulation of hydrocarbon reservoirs. The model is based on the premise that we have three different fluid phases (aqueous, oleic, and gaseous) and three (pseudo)components: water, oil, and gas. Oil and gas are not single hydrocarbon species, but represent all hydrocarbon species that exist in liquid and vapor form at surface conditions. Mixing is allowed, in the sense that both oil and gas can be found in the oleic phase, in the gaseous phase, or in both. The amounts of dissolved gas in the oleic phase and vaporized oil in the gaseous phase must be kept track of. Below, we will use subscripts w, o, g to indicate quantities related to each of the phases or components.

2.1.1. Model equations

The black-oil equations can be deduced from conservation of mass for each component with suitable closure relationships such as Darcy's law and initial and boundary conditions. The equations are discretized in space with an upwind finite-volume scheme using a two-point flux approximation and in time using an implicit (backward) Euler scheme. The resulting equations are solved simultaneously in a fully implicit formulation by a Newton-type linearization with a properly preconditioned, iterative linear solver. We give both the continuous and discrete equations.

Continuous equations. The conservation laws form a system of partial differential equations, one for each pseudocomponent α:

$$\frac{\partial}{\partial t}\left(\phi_{\text{ref}} A_\alpha\right) + \nabla \cdot \mathbf{u}_\alpha + q_\alpha = 0, \tag{1}$$

where the accumulation terms and fluxes are given by

$$A_w = m_\phi b_w s_w, \qquad \mathbf{u}_w = b_w \mathbf{v}_w, \tag{2a}$$

$$A_o = m_\phi (b_o s_o + r_{og} b_g s_g), \qquad \mathbf{u}_o = b_o \mathbf{v}_o + r_{og} b_g \mathbf{v}_g, \tag{2b}$$

$$A_g = m_\phi (b_g s_g + r_{go} b_o s_o), \qquad \mathbf{u}_g = b_g \mathbf{v}_g + r_{go} b_o \mathbf{v}_o. \tag{2c}$$

and the phase fluxes are given by Darcy's law:

$$\mathbf{v}_\alpha = -\lambda_\alpha \mathbf{K}(\nabla p_\alpha - \rho_\alpha \mathbf{g}). \tag{3}$$

In addition, the following closure relations should hold:

$$s_w + s_o + s_g = 1, \tag{4}$$

$$p_{c,ow} = p_o - p_w, \tag{5}$$

$$p_{c,og} = p_o - p_g; \tag{6}$$

see Appendix A.1 for definitions of the symbols used.

Discrete equations. In the following, let subscript i denote a discrete quantity defined in cell i and subscript ij denote a discrete quantity defined at the connection between two cells i and j. Two cells share a connection if they are adjacent geometrically in the computational grid or share an explicit non-neighbor connection (NNC; provided by the user). For example $v_{o,ij}$ is the oil flux from cell i to cell j. For oriented quantities such as fluxes, the orientation is taken to be from cell i to cell j, and the quantity is skew-symmetric ($v_{o,ij} = -v_{o,ji}$), whereas non-oriented quantities such as the transmissibility T_{ij} are symmetric ($T_{ij} = T_{ji}$). Quantities with superscript 0 are taken at the start of the discrete time step, other quantities are at the end of the time step. Superscripts or subscripts applied to an expression in parenthesis apply to each element in the expression.

The discretized equations and residuals are, for each pseudo-component α and cell i:

$$R_{\alpha,i} = \frac{\phi_{\text{ref},i} V_i}{\Delta t}\left(A_{\alpha,i} - A^0_{\alpha,i}\right) + \sum_{j \in C(i)} u_{\alpha,ij} + q_{\alpha,i} = 0, \tag{7}$$

where A_α are as in (2) and u_α are fluxes defined similar to the velocities $\mathbf{u}_\alpha$:

$$u_w = b_w v_w, \tag{8a}$$

$$u_o = b_o v_o + r_{og} b_g v_g, \tag{8b}$$

$$u_g = b_g v_g + r_{go} b_o v_o. \tag{8c}$$

The relations (4)–(6) hold for each cell i. The fluxes are given for each connection ij by:

$$(b_\alpha v_\alpha)_{ij} = (b_\alpha \lambda_\alpha m_T)_{U(\alpha,ij)} T_{ij} \Delta\Phi_{\alpha,ij} \tag{9}$$

$$(r_{\beta\alpha} b_\alpha v_\alpha)_{ij} = (r_{\beta\alpha} b_\alpha \lambda_\alpha m_T)_{U(\alpha,ij)} T_{ij} \Delta\Phi_{\alpha,ij} \tag{10}$$

$$\Delta\Phi_{\alpha,ij} = p_{\alpha,i} - p_{\alpha,j} - g\rho_{\alpha,ij}(z_i - z_j) \tag{11}$$

$$\rho_{\alpha,ij} = (\rho_{\alpha,i} + \rho_{\alpha,j})/2 \tag{12}$$

$$U(\alpha, ij) = \begin{cases} i, & \Delta\Phi_{\alpha,ij} \geq 0, \\ j, & \Delta\Phi_{\alpha,ij} < 0. \end{cases} \tag{13}$$

See Appendix A.2 for definitions of the symbols used above.

2.1.2. Choice of primary variables

For a grid with n cells, the material balances (7) give us $3n$ equations. (When only two phases are defined, either oil/water or oil/gas, the system is simplified and we solve only two mass balance equations, giving $2n$ equations.) In addition, we have equations and unknowns from the well model, which will be described in Section 2.2. An important question is which quantities we should use as primary variables, i.e., the unknowns we solve for. There is some flexibility, as we have relations connecting many of the quantities.

Our basic choice for non-miscible flow is to use the pressure of one of the phases (which we take to be oil pressure p_o), s_w, and s_g as our primary variables. The pressure p_o will then behave mostly in an elliptic fashion, whereas the saturations s_w and s_g are more hyperbolic.

For miscible flow, it is a little more complicated: Since the gaseous phase may disappear if all the gas dissolves into the oleic phase, and similarly the oleic phase may disappear if all the oil vaporizes into the gaseous phase, we cannot always use s_g for our third variable. Instead we use the third variable to track the composition of the phase that has *not* disappeared. We therefore use either s_g, r_{go}, or r_{og} as our third primary variable, depending on the fluid state, and we call that variable x:

$$x = \begin{cases} s_g, & \text{all three phases present,} \\ r_{go}, & \text{no gaseous phase,} \\ r_{og}, & \text{no oleic phase.} \end{cases} \tag{14}$$

This choice is made separately for each cell i depending on its state.

2.1.3. Initial and boundary conditions

Initial conditions are defined by the initial values of pressure p, saturations s_α, and, if applicable, the mixing ratios r_{go} and r_{og}. These values can be specified by the user directly in the input data, or computed from an equilibration procedure that ensures the initial state is in vertical equilibrium. This procedure takes as input the fluid pressure at a reference depth, the depths of the water–oil and oil–gas contacts, and the capillary pressures at those contact depths. It then solves the following ordinary differential equations:

$$\frac{\mathrm{d}p_\alpha}{\mathrm{d}z} = \rho_\alpha(z, p_\alpha)\, g. \tag{15}$$

The fluid density typically depends on the pressure, and possibly also on its composition; see Section 2.1.6 for details. The equations are solved numerically using a 4th-order Runge–Kutta method, with the order of the phases decided by the location of the datum depth: First, we solve for the phase in whose zone the datum is located, using the datum pressure to fixate the pressure solution. We then solve for phases corresponding to the zones above and below. To fixate the phase pressures, we use the already computed phase pressure(s) evaluated at the zone contact depth and the input capillary pressures.

Good estimates of the initial water saturation of the reservoir are often known from seismic and well logs. This data can be used to improve the initial state. The equilibration procedure is the same as described above, but the capillary pressure is scaled to match the initial water saturation. Each cell will thus have its own scaling of the capillary pressure that needs to be taken into account during the simulations.

OPM Flow has so far mainly been developed to simulate reservoirs having no fluid communication with the surrounding rock; the default boundary conditions are therefore no-flow Neumann conditions. In that case, the only way the reservoir communicates with the outside is through the coupled well models, see Section 2.2. In recent versions, support for more general boundary conditions has been added, as well as support for some aquifer models that allow the reservoir domain to be in communication with surrounding water-carrying rock.

2.1.4. Rock properties

Rock *porosity*, denoted ϕ, is the void fraction of the bulk volume that is able to store and transmit fluids. It usually depends on pressure, which we model as $\phi = m_\phi(p)\phi_{\text{ref}}$, where the multiplier m_ϕ is a function of pressure and ϕ_{ref} is a reference porosity. This model does not account for the possibility of irreversible changes to the rock, such as fracturing. We model porosity as cell-by-cell piecewise constant.

The rock *permeability* (or absolute permeability), denoted $\mathbf{K}$, measures the ability of a porous medium to transmit a single fluid phase at certain pressure conditions; see Darcy's law (3). For an isotropic porous medium the permeability is a scalar, but in general $\mathbf{K}$ is a tensor, indicating that the medium's resistance to flow depends on the flow direction.

A typical example includes layered rocks in subsurface reservoirs, which are typically more permeable in the directions tangential to the layer than in the normal direction. We model permeability as a cell-by-cell piecewise constant tensor. The permeability is not used directly in the discrete formulation above, but enters the equation through the transmissibility factor; see Appendix B for details.

2.1.5. Relative permeability and capillary pressure

The rock's ability to transmit fluids is reduced when more than one fluid phase is present. This is modeled by introducing a saturation-dependent factor $k_{r,\alpha}$ called *relative permeability*. (In (3), we use the *mobility* $\lambda_\alpha = k_{r,\alpha}/\mu_\alpha$, where μ_α denotes the viscosity.) For a three-phase system, one could imagine that relative permeabilities were functions of two independent arguments. A more common and natural approach is to assemble the three-phase properties stepwise from "two-phase" relations. The water relative permeability is then assumed to be a function of water saturation only, $k_{r,w}(s_w)$, and its domain is confined by the connate water saturation s_{wco} and maximum saturation $s_{w,max}$. We complement the water/oil sub-system by a relative permeability for oil $k_{r,ow}(s_o)$, with oil saturations bounded by $s_o \in [1-s_{w,max},\, 1-s_{wco}]$. Similarly for the gas phase, we have a relative permeability $k_{r,g}(s_g)$, and endpoints $s_{g,min}$ and $s_{g,max}$ respectively. Whereas the connate water s_{wco} is a significant parameter that signals presence of water also in the gas cap, it is usual to assume $s_{g,min} = 0$. Thus, the oil relative permeability for the gas/oil/connate-water subsystem, $k_{r,og}(s_o)$, has a domain bounded by $s_o \in [1 - s_{g,max},\, 1 - s_{wco}]$.

Both sub-systems are also equipped with a capillary pressure relation, and assuming normal wettability conditions (water wetting relative oil) $p_{c,ow}(s_w)$ will be a decreasing function of s_w while (oil wetting relative gas) $p_{c,og}(s_g)$ is an increasing function of s_g.

A simple and robust approach for combining $k_{r,ow}$ and $k_{r,og}$ into a true three-phase relative permeability $k_{r,o}$ is to assume local segregation in each cell and use a volume-weighted average between the water and gas zones

$$k_{r,o}(s_o) = \frac{(s_w - s_{wco})k_{r,ow}(s_o) + s_g k_{r,og}(s_o)}{s_w - s_{wco} + s_g}. \tag{16}$$

For each property, an arbitrary number of alternative tables can be defined and assigned cellwise throughout the grid. The functions may also be modified by the introduction of cellwise transformations, known as endpoint scaling, that provide local adaptivity.

OPM Flow provides some support for hysteresis effects [11] that typically will arise from reversal of saturation changes. For relative permeabilities, the possibility is limited to the non-wetting phases, i.e., the oil phase in the water/oil sub-system and the gas in the gas/oil system. In addition to the usual (drainage) relative permeability $k_{r,n}(s_n)$, a second (imbibition) curve $k^{imb}_{r,n}(s_n)$ must be supplied. Consider a drainage process that is reversed after reaching some saturation s^{hst}_n. Following Carlson's method [12], a so-called scanning curve is obtained by finding an appropriate saturation translation Δ^{hst}_n from the relation $k^{imb}_{r,n}(s^{hst}_n + \Delta^{hst}_n) = k_{r,n}(s^{hst}_n)$. The imbibition process will then proceed according to the relative permeability curve $k^{imb}_{r,n}(s_n + \Delta^{hst}_n)$.

2.1.6. Pressure–volume–temperature (PVT) relationships

Pseudocomponents, or components for short, essentially correspond to the identifiable phases at standard surface conditions, and each component is assigned a surface density, $\rho_{S,\alpha}$. At reservoir conditions, the fluid volume will consist of certain mix of components, referred to as the composition. In the standard black-oil formulation, pressure is essentially the single intensive quantity, thus implicitly assuming steady-state heat distribution. (This can to some extent be generalized by partitioning the reservoir into multiple PVT regions, each with a separate set of properties.)

Water has a simple PVT behavior, with a one-to-one correspondence between the water component and the aqueous phase in terms of the water shrinkage factor $b_w = \rho_w/\rho_{S,w}$ that relates the density ρ_w at reservoir conditions to the density $\rho_{S,w}$ at surface conditions. The factor is a function of water pressure, and it is parameterized in terms of a constant compressibility, a reference pressure, and a corresponding reference factor. So $b_w = b_w(p_w; c_w, p_{w,ref}, b_{w,ref})$. Water viscosity, μ_w, is considered pressure dependent, and the fraction μ_w/b_w is also specified as a constant "compressibility" quantity.

The oleic phase behavior can be specified at various levels of complexity, and the simplest formulation available is similar to that of water. There is also a slightly more flexible variant available, in which b_o and μ_o are given as tabulated functions of the oil pressure p_o, thus relaxing the constant compressibility form. These versions do not permit any dissolved gas and are consequently referred to as "dead oil" formulations.

For a system with gas dissolved into the oleic phase, we must specify the dissolution capacity of the oleic phase, represented as the maximum gas–oil ratio r_s, as a function of pressure. For each value of r_s, the corresponding pressure is referred to as the bubble point pressure p_{bpp}. Thus, the actual ratio of dissolved gas at given reservoir conditions obeys $r_{go} \le r_s(p_o)$ with equality if $p_o \le p_{bpp}$. Shrinkage factors b_o and viscosities μ_o must be supplied accordingly, and in particular, properties must be defined also for undersaturated conditions, i.e., pressures above p_{bpp}. A simulator must be able to handle the change from having no free gas at pressures above p_{bpp} to having a gas phase present at lower pressures. An example is given in Section 3.4; see Fig. 14. Oil containing dissolved gas is known as "live oil".

The properties of the gaseous phase mirror those of the oleic, and the gas equivalent to dead oil is called "dry gas", referring to the exclusion of vaporized oil from the gaseous phase. This opposed to "wet gas", which in analogy to the

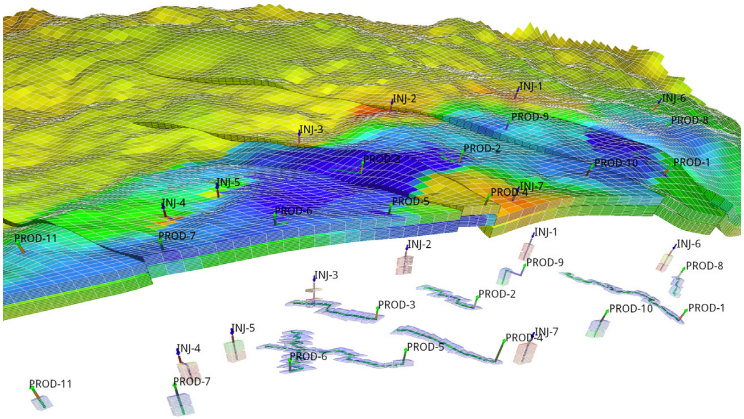


Fig. 3. Case from the OLYMPUS Optimization Challenge [15]. Top: pressure at the end of the simulation time. Bottom: wells and the cells they are connected to.

live-oil model, requires specification of a maximum vaporized oil–gas ratio r_v that controls the amount of oil in the gaseous phase. OPM Flow can also have a fourth phase to support solvent modeling, and these PVT properties are defined similar to those of dry gas.

OPM Flow provides a mechanism to contain or tune the behavior when combining live-oil and wet-gas properties. The values of both r_s and r_v may be scaled back by some positive powers of the fraction $(s_o/s_{o,max})$, where the denominator refers to the cellwise historical maximum of the oil saturation s_o.

2.2. Well models

OPM provides two different well models. The 'standard' well model describes the flow conditions within each well with a single set of primary variables [13]. This works adequately in describing the majority of wells in reservoir simulation, as a result, it has been used as the default well model. More advanced multi-segment well models [14] discretize the wellbore with 'segments' to provide more detailed, flexible, and accurate wellbore simulation, especially for multi-lateral wells and long horizontal wells. In either case, wells interact with reservoir formation through connections that represent the flow paths between the wellbore and single grid blocks. The calculation of inflow rates through connections plays an important role within well modeling. Fig. 3 shows both vertical and mostly horizontal wells in a realistic reservoir model.

2.2.1. Standard well model

For the standard well model, a single set of primary variables is introduced for each well. For a three-phase black oil system, there are four primary variables: Q_t is the weighted total flow rate, the weighted fractions of water F_w and gas F_g describe the fluid composition within the wellbore [13], and the fourth is the bottom-hole pressure p_{bhp}, i.e., the pressure in the wellbore at the datum depth. The following equations relate the primary variables to the flow rates:

$$Q_t = \sum_{\alpha\in\{o,g,w\}} g_\alpha Q_\alpha, \quad F_w = \frac{g_w Q_w}{Q_t}, \quad F_g = \frac{g_g Q_g}{Q_t} \tag{17}$$

Here, Q_α is the flow rate of component α under surface conditions and g_α is a weighting factor. For gas, this factor is typically chosen to be a small value like 0.01 to avoid gas fractions close to unity [13].

Volumetric inflow rates at reservoir conditions are calculated as

$$q^r_{\alpha,j} = T_{w,j} M_{\alpha,j}[p_j - (p_{bhp,w} + h_{w,j})], \tag{18}$$

where $q^r_{\alpha,j}$ is the flow rate of phase α through connection j. The other symbols are described in Appendix A.3. The rate is negative for flow from wellbore to reservoir, and positive for flow into the opposite direction. The pressure differences $h_{w,j}$ between connection j and the datum point are computed explicitly based on the fluid composition in the wellbore at the start of the timestep.

To keep the system closed, we introduce conservation equations for each component,

$$R_{\alpha,w} = \frac{A_{\alpha,w} - A^0_{\alpha,w}}{\Delta t} + Q_\alpha - \sum_{j\in C(w)} q_{\alpha,j} = 0, \tag{19}$$

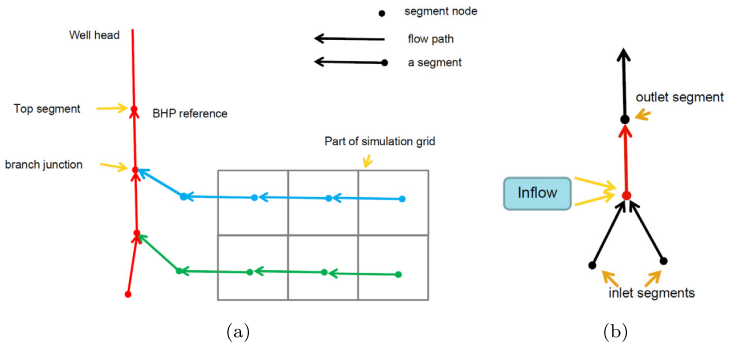


Fig. 4. An illustration of the multi-segment well and the segment structure.

that are solved in a fully implicit and coupled way with the black-oil equations (7). Here, $C(w)$ is the set of connections of the well w, $q_{\alpha,j}$ is the flow rate of phase α through connection j under surface condition, which can be calculated from $q^r_{\alpha,j}$ defined in (18); the relation is similar to the one between component and mass fluxes (2). The storage term $A_{\alpha,w}$ describes the amount of component α in the wellbore (small volume), and it is introduced for better stability of the well solution.

In addition, we need equations that model how the wells are controlled. The well control equation for a prescribed bhp target reads

$$R_{c,w} = p_{bhp,w} - p^{target}_{bhp,w} = 0, \tag{20}$$

whereas for a rate-controlled well we have

$$R_{c,w} = Q_\alpha - Q^{target}_\alpha = 0. \tag{21}$$

where Q^{target}_α is the desired surface-volume rate of the controlled component α, typically oil rate for a production well.

Under some conditions, fluids from the reservoir formation can enter the wellbore through some connections and get reinjected into the formation through other connections of the same well. This phenomenon is called cross-flow, and is modeled by the standard well model. Since the model assumes that the fluid composition is uniform throughout the wellbore (given by F_g and F_w), the same composition will be injected through all injecting connections. When this is not accurate enough, a multi-segment well model can be used.

2.2.2. Multi-segment well model

So-called multi-segment well models [14] were introduced to simulate more advanced wells, such as multilateral wells, horizontal wells, and inflow control devices. With a multi-segment model, the wellbore is divided into an arbitrary number of segments (Fig. 4). Including more segments can give a more accurate simulation, at higher computational cost. Each segment consists of two parts: a segment node and a flow path to the neighboring segment in the direction towards the well head, defined as the outlet segment (Fig. 4(b)). Most segments also have inlet segment neighbors in the direction away from the well head. In the well shown in Fig. 4(a), there are three branches, including one main branch (in red color) and two lateral branches (in green and blue colors). For multilateral wells, a segment node must be placed at the branch junction. Segments at branch junction points have additional inlet segments and segments at the end of branches do not have inlet segments. The top segment of each well does not have an outlet segment. The pressure at this segment is the bottom-hole pressure of the well, whereas the component rates equal the component rates for the well as a whole.

Each segment has the node pressure p as a primary variable in addition to the primary variables Q_t, F_w, and F_g introduced for the standard well model (see (17)). Likewise, the component equations (19) are extended to include additional terms $Q_{\alpha,i}$ that represent the flow rate from the inlet segments $I(n)$ of segment n as well as the inflow from the reservoir through the segment's connections $C(n)$.

$$R_{\alpha,n} = \frac{A_{\alpha,n} - A^0_{\alpha,n}}{\Delta t} - \sum_{i \in I(n)} Q_{\alpha,i} - \sum_{j \in C(n)} q_{\alpha,j} + Q_{\alpha,n} = 0. \tag{22}$$

Each segment can thus receive inflow from more than one connection or no connection through the segment node, whereas each connection can only contribute to one segment. In the standard well model, the connection is located at the centroid of the grid block. In the multi-segment well model, the connection and segment node can be located at any place in the grid block. The inflow calculation of the connection in the multi-segment well model is as follows:

$$q^r_{\alpha,j} = T_{w,j} M_{\alpha,j} (p_j + H_{cj} - p_n - H_{nc}). \tag{23}$$

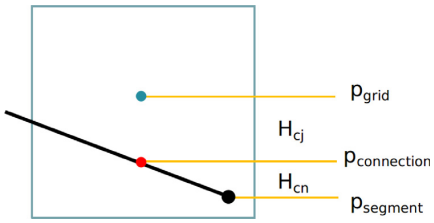


Fig. 5. Hydrostatic pressure drops used in the multi-segment well model.

Here, p_j is the pressure at the cell center of the grid block that contains connection j; H_{cj} is the hydrostatic pressure difference between the cell center and the connection; p_n is the pressure of segment n; and H_{nc} is the hydrostatic pressure difference between the connection and the segment (Fig. 5).

The last equation for each segment describes the pressure relationships between the segment n and its outlet segment m:

$$R_{p,n} = p_n - p_m - H_h - H_f - H_a = 0. \tag{24}$$

Here, the H terms represent the hydrostatic, frictional, and acceleration pressure drops between the segments.

The top segment does not have an outlet segment, and hence the pressure equation is replaced by a well control equation, which is the same as for the standard well model, (20) or (21).

As with the standard well model, the well equations are solved with the reservoir mass conservation equations in a coupled, fully implicit way, see Section 2.3.2.

2.3. Solution strategy

Altogether, the reservoir and well equations described thus far define a large system of (fully implicit) nonlinear equations, which we can be written on compact residual form as $R(y) = 0$, where y is the vector of primary variables. This system is solved using a Newton–Raphson type method. Let $y_n = (p_o, s_w, x)$ be the primary variables after n Newton iterations. Given an initial state y_0, the solution of $R(y) = 0$ can be found by iteratively solving

$$J(y_n)(y_{n+1} - y_n) = -R(y_n) \tag{25}$$

until $R(y_n)$ is less then some prescribed tolerance. Creating the Jacobian matrix $J(y_n)$ and solving this linearized problem is the core computational task of the simulator, and its performance depends largely on how this part of the code is programmed. The linearization and the solution procedure for (25) is explained in detail in the following subsections, but first some words about convergence of the nonlinear solver.

2.3.1. Nonlinear solver issues

The solution at the end of the previous time step is used as initial guess, therefore the Newton–Raphson method is expected to converge rapidly if a sufficiently small time step is used. For large time steps and/or large changes in the solution, typically caused by changing well controls, the Newton–Raphson method may converge poorly or not at all. One simple way of improving the convergence is to restrict the change allowed per iteration and prevent the solution from jumping far across the boundary between saturated and undersaturated states in a single iteration. This so-called Appleyard chop technique [16] is implemented in OPM Flow, and is often sufficient for the method to converge. Still, time steps may need to be cut to make the solver converge for some difficult cases.

Note also that since the third primary variable x may change interpretation from one iteration to the next, we are not strictly using the Newton–Raphson method as such, but a close relative. Changing primary variable may disturb the convergence, and we therefore add a small threshold to the variable-switching logic in (14) to prevent oscillation between states. Lack of smoothness of the residuals can also come from other sources. For example, the functions for relative permeability and capillary pressure can be almost discontinuous in some cases, they can also have hysteretic behavior.

The third point that affects the convergence of the nonlinear solver is the accuracy of the linear solvers. Solving the linear system to a loose accuracy may lead to poor convergence of the nonlinear solver. The tuning of the time-stepping procedure, and the tolerance used in the linear and nonlinear solvers are highly related, and optimal choices are case- and problem-dependent.

From the user side, there are several command-line options that can be used to control the solver. In particular, `--tolerance-mb` controls the mass balance error as a reservoir-average saturation error, and `--tolerance-cnv` controls the maximal local residual error. For more information about runtime options and other aspects of running OPM Flow, see Section 2.2 of the OPM Flow manual [10]. Running OPM Flow with the option `--help` will print a list of available parameters.

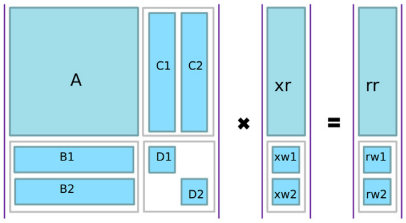


Fig. 6. Structure of the matrix system for a case with two wells. Here, A contains the reservoir flow equations, D_1 and D_2 contain the linearized well equations for each of the two wells, whereas the B and C matrices are coupling entries between the reservoir and well models. Note that the size of A is typically orders of magnitude larger than the other submatrices..

2.3.2. Solving the system of linear equations

The overall structure of the linearized system of equations is shown in Fig. 6. The well equations are added to the global system using a Schur complement approach:

$$(A - \sum_w C_w D_w^{-1} B_w) x_r = R_r - \sum_w C_w D_w^{-1} R_w, \tag{26}$$

$$x_w = D_w^{-1}(R_w - B_w x_r). \tag{27}$$

Eq. (25) is always taken to be this modified system. The linear system solved is therefore always of the same size as the reservoir part of the system.

Depending on fluid properties and other reservoir-specific parameters, the system of equations (7) could exhibit elliptic or degenerate parabolic behavior or could even be hyperbolic for specific setups. Therefore, solving the linearized system (25) stemming from these equations is challenging, since it is likely non-symmetric and ill-conditioned. Instead of solving equation (25) directly, the solution is approximated by an iterative linear solver. Two variants are available, a stabilized Bi-conjugate Gradient method (BiCG-stab) [17] and a restarted Generalized Minimal Residual (GMRes) [18] solver. By default, BiCG-stab is used. In addition, several options are available for preconditioning of the linear system, the default choice being the incomplete lower triangular–upper triangular factorization with level of fill in 0 (ILU0). More sophisticated choices are based on an algebraic multigrid (AMG) method for preconditioning, such as a recently added Constraint Pressure Residual (CPR) preconditioner (originally introduced in [19]) using the approach of [20].

Implementation of the AMG and the CPR preconditioner is further discussed in Section 2.5 and the solver itself in [21]. The OPM Flow manual [10] gives an overview of parameters for choosing solvers and preconditioners, as well as other related parameters. The solvers and preconditioners are implemented in the Dune module dune-istl and described in [22,23]. In the current version, OPM Flow assembles the Jacobian matrix into the block compressed row storage (`BCRSMatrix`) data structures provided by dune-istl. Each matrix entry is of the type `FieldMatrix`, which is a dense matrix with fixed row and column length provided by dune-common. Blocked matrix storage enables better performance on modern CPUs compared to the standard approach that stores each matrix entry separately. Likewise, the unknown Δy and the right hand side $-R(y_n)$ are stored in a `BlockVector` from the dune-istl package.

The interface between OPM Flow and linear solver packages has been designed in a flexible way so that besides dune-istl other linear algebra packages could be used. For example, an experimental implementation using the PETSc library [24] has been successfully tested. We compared dune-istl from OPM Flow's default setup with solvers from PETSc on the Norne case and found that the PETSc solvers were between 1% and 2% faster. However, thread parallel matrix assembly is much faster with dune-istl than with PETSc. The reason is that whereas dune-istl enables direct access to matrix data structures, PETSc only offers procedural access. Since PETSc would introduce another hard library dependency to the project, our overall evaluation was that using dune-istl is more advantageous. On the other hand, having developed a coupling to PETSc solvers enables the project to periodically check for improved alternatives as those may become available. We have also tested OPM Flow with the AMGCL [25] library. In particular, we have compared the CPR preconditioner from AMGCL and the one based on dune-istl and found that the performance is similar. A detailed study of linear solver performance is outside the scope of this paper.

2.4. Automatic differentiation in OPM Flow

The coefficients of the linearized systems required by the nonlinear Newton solver (e.g., $J(y_n)$ in Eq. (25)) have traditionally been computed by evaluating closed-form expressions obtained by differentiating the discretized flow equations analytically. Differentiating these flow equations manually and programming the resulting formulas is both time-consuming and error-prone. The problem is particularly pronounced when extending an existing simulator with new functional relationships. If a quantity is set to depend on a different or new (primary) variable, the changes in derivatives will propagate through the chain rule to the Jacobians of all quantities that depend on the modified quantity. Thus, even

very simple extensions in functional dependencies may cause large modifications throughout the simulator code. As stated in [26]: "The tedious and error-prone process of constructing the Jacobian matrix is a major reason why it is difficult to accommodate additional capabilities into existing simulators".

Automatic differentiation (AD) is a way to mitigate this problem; see for example [27] or [8] for an introduction. There are many examples of AD tools used for scientific computing, such as Sacado [28], ADOL-C [29] or ADETL [30]; the latter used for the AD-GPRS [26,31] reservoir simulator.

The AD mechanism implemented in OPM works by treating each quantity in the program not as just a single value, but as a value paired with values for the "relevant" partial derivatives. Whenever the value of a new quantity is computed, a combination of the chain rule and standard differentiation rules for elementary unary and binary operations is used to compute the correct value of all the corresponding partial derivatives. This is called the forward approach to AD, in contrast to backward AD, which stores the adjoints of each quantity.

When it was decided to use AD in OPM, we had already gained significant experience from developing a new object-oriented framework based on automatic differentiation in MRST [5]. As part of this work, we evaluated and tested different pre-existing libraries, but found none that both provided the desired flexibility and performance and were made available under a compatible open-source license. Experience from MRST [5] and [8] indicated that implementing a custom library would represent a limited effort. This proved to be true: the resulting AD template class in OPM consists of approximately 650 lines of code, including comments. Using this library also avoids adding another external dependency to the project.

Basing OPM on AD has in our opinion turned out to be a very good decision. It has eliminated a lot of mundane work in developing new models and functionality, and has also opened a future road to a simulator that will be largely differentiable.

An alternative to AD is using numerical differentiation of the residuals to approximate the Jacobian. This has been applied successfully, for example in DuMuX [2]. Since the resulting Jacobian is an approximation, this can lead to a slight increase in Newton iterations. AD Jacobians will be as accurate as hand-coded ones, and you do not have to choose an epsilon for the finite differences. It is possible to compile large parts of OPM to use numerical differentiation instead of AD by setting some options at compile time. Comparing the results indicates that the AD-using code is a little faster overall.

The cost of AD for the user can be lower performance compared to a well-tuned hand-written implementation. We have made numerous improvements to our automatic differentiation implementation to reduce this overhead, but it is clear that by investing sufficient effort into a hand-written code, one can possibly get better performance. Writing such optimal Jacobian code for arbitrary extensions is not a trivial task, and due to the increased chance for errors is not guaranteed to give any speed-up at all.

2.4.1. General approach

The OPM implementation of AD introduces a class that mimics the behavior of the built-in floating point types of C++ (`double` and `float`) as closely as possible. The AD object contains a value and a fixed number of derivatives and defines all basic arithmetic operators as well as common mathematical functions. Also, to allow easy comparisons with code that uses standard floating-point objects, all functions that work on AD-objects can also be used with objects of built-in floating-point types. In OPM an implementation class called `Evaluation` is provided for this purpose:

```
// ValueT is typically double or float
template <class ValueT, int numDerivs>
class Evaluation;
```

This class overloads all arithmetic operators so that objects of type `Evaluation` can be used in the same way as plain `double` or `float` types. An example code snippet for the implementation of the multiplication operator looks as follows:

C++ code

```
// Multiplication operator
Evaluation& operator*=(const Evaluation& other)
{
    // The values are multiplied, whereas the derivatives
    // follow the product rule, i.e.,~(u*v)' = (u'u + u'v).
    const ValueType u = this->value();
    const ValueType v = other.value();
    // Value, usually position 0
    data_[valuepos_] *= v ;
    //  Derivatives usually start at position 1
    for (int i = dstart_; i < dend_; ++i) {
        data_[i] = data_[i] * v + other.data_[i] * u;
    }
    return *this;
}
```

For improved performance the implementation of the `Evaluation` class is specialized for values with up to 12 partial derivatives, whereas a default implementation based on `for` loops is used for higher numbers of derivatives. The opm-material module provides a python script implementing a simple code generation algorithm for the specialized `Evaluation classes` for any number, if necessary. Typical three-phase black-oil equations use three partial derivatives, corresponding to the number of primary variables per cell.

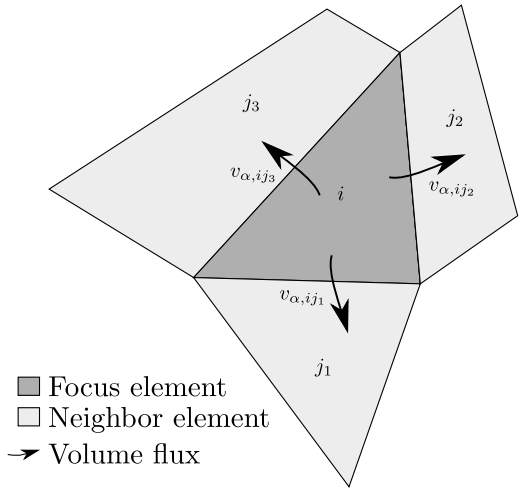


Fig. 7. Localized linearization of element i and its neighbors $j_{(\cdot)}$. Equations (7) are evaluated locally with derivatives of the primary variables of element i.

2.4.2. Application-specific localization

For performance reasons, the implementation of automatic differentiation in OPM only supports carrying a fixed number of derivatives, which needs to be specified at compile-time. Yet, the number of unknowns for the discretized conservation equations (7) is only known at run-time because it depends on the number of grid cells.

This problem is solved using a localized linearization scheme as illustrated by Fig. 7: The residuals of the mass conservation equations are computed for a single element, which is called the *focus* element. In addition to the values of the residuals of the focus element, the partial derivatives of the residuals of all elements in the local neighborhood are calculated with regard to the fixed number of primary variables of the focus element. For more details on this approach, confer [32].

The two-point flux-approximation (TPFA) scheme used ensures that the residuals of any cell only depend on the variables of the cell itself and its immediate neighbors, which lets the localized linearization scheme be applied.

Once the local residual vector and its dense Jacobian matrix have been computed, they are transferred to a variable-sized vector for the residual and a variable-sized sparse matrix for its Jacobian matrix representing the linearized system of equations for the whole spatial domain. This vector and the matrix are subsequently passed to the linear solver.

2.5. Parallelization

Parallelization of OPM Flow follows a hybrid approach with a combination of multi-threading and MPI. The assembly of the Jacobian matrix is multi-threaded through OpenMP, and controllable through the OpenMP environment variables. The assembly part is less dependent on memory bandwidth, and this enables us to exploit more CPUs on a multicore machine than the scaling possible for the linear solve part. In particular, it allows for improved performance on CPUs with hyper-threading. Writing binary results to file is also threaded to insulate the reservoir simulation from slow disk systems, as further commented on in Section 2.6. The `CpGrid` class implementing stratigraphic grids is based on the Dune grid interface and OPM Flow therefore naturally inherits the MPI parallel interface of the Dune modules dune-grid and dune-istl (see [1,23,33]). This keeps the implementation effort as small as possible. Our main goal is to reuse sequential algorithms whenever possible and only introduce parallelism specific code where absolutely necessary. In particular, the discretization, and algorithms for adaptive time stepping and the Newton–Raphson nonlinear solver are the same for sequential and parallel runs. For the latter, only the convergence criterion will be switched to a parallel one as it needs some global communication.

Standardized parallel file formats for oil reservoir data are currently lacking and the ECLIPSE file format is inherently sequential in nature (see Section 2.6). At the beginning of a simulation, a process therefore needs to read the complete input file and set up a grid for the entire domain. This is a potential bottleneck for scalability to large number of cores, which has not yet been reached. Using the entire grid read initially, the transmissibilities representing cell–cell connections are computed. A high value means that a change of pressure or saturation in a cell will influence the properties of the neighboring cell strongly. Therefore, the current domain decomposition in OPM Flow is based on keeping cells connected through high transmissibilities on the same process when partitioning. This is done to ease the coarsening

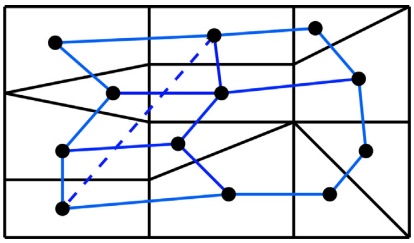


Fig. 8. Graph of a grid with one well, where the black dots are the vertices associated with cells in the grid and the solid lines the edges associated with the cell faces. The dashed blue line is an edge introduced by a well connecting two non-neighboring cells.

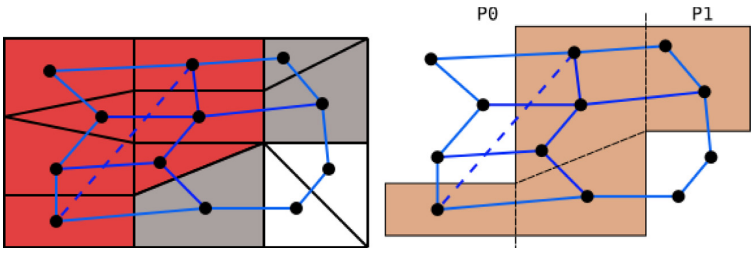


Fig. 9. (a) Interior (red) and ghost (gray) cells stored on a process, and (b) grid cells involved in communication. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

phase of the algebraic multigrid solver and improve convergence of other linear solvers. The implementation currently does not allow one well to be distributed across multiple subdomains.

At the start of the simulation, the global grid is partitioned using the graph partitioning software Zoltan [34,35], which was chosen because of its flexible license and because of its interfaces to other partitioning software libraries. The partitioning is computed from the grid graph defined with cells as vertices and edges either representing a face between two cells or a connection between two cells via a well. Fig. 8 shows a conceptual illustration. An edge associated with a cell face is assigned a scaled value of the transmissibility whereas any edge associated with a well connection gets the maximum weight available to ensure that all cells connected by a well end up in the same subdomain. This will give partitions in which domain boundaries are at faces having low transmissibility. The motivation for using transmissibility as edge weights is to improve the convergence of the parallel linear solver. Because of this particular choice of edge weights, the edge-cut will no longer be an approximation of the total communication volume. Hence, the improved convergence comes at the cost of increased parallel overhead.

To ensure validity of the partitioning, a post-processing step is introduced after the initial load balancing to enforce that all cells connected by a well are really contained in one subdomain, since our edge weights are merely a strong suggestion to the load balancer. As a result, the calculation of the well model can be done sequentially on the process that stores the well without additional communication.

The flux calculation, which is a central part of any cell-centered finite-volume scheme, needs to access information from neighboring cells. OPM Flow therefore not only stores the grid cells assigned to one process, but also a halo layer of so-called ghost cells at the process boundary. During the simulation, the values attached to these ghost cell are updated using point-to-point communication. Fig. 9(a) illustrates the local grid stored on one process.

This decomposition is used to set up a distributed view of the `CpGrid` class. It implements the parallel DUNE grid interface [1,33] allowing values attached to interior cells to be easily communicated to other processes, where these cells belong to the ghost layer. For the small example in Fig. 9(b), only the brown cells will be involved in the communication.

As a result, the sequential discretization algorithms can be reused almost unchanged, since all necessary data (neighboring and wells) is available on one process and only one communication step during each linearization step of the nonlinear solver is needed to update the ghost layer.

The iterative solvers in dune-istl are parallelization agnostic and can be turned into parallel ones by simply using parallel versions of the preconditioner, scalar product, and linear operator. All preconditioners, except AMG, apply the sequential operator on the degrees of freedom attached to the interior cells only and update the rest using one step of interior-ghost communication. When used in AMG as smoothers, these parallel preconditioners are called hybrid smoothers (see [36]). This approach works very well in the AMG solver provided by dune-istl [21], as it uses data agglomeration onto fewer processes whenever the number of unknowns per process drops below a prescribed threshold.

Thus on coarser levels fewer processes are involved in the computation until on the coarsest level only one process computes and a direct coarse solver is used. This remedies the missing smoothing across the processor partition boundaries on the finer levels and keeps the ratio between time used for computation between communication steps and time needed for communication high. In addition, a block ILU0 (one block for each process) or a CPR preconditioner is provided. Both the existing parallel scalar product and linear operator from dune-istl are reused without modifications.

2.6. I/O in OPM Flow

To be industrially relevant, OPM Flow must read and write files in standard formats well known to the industry. OPM Flow implements support for the input format used by the ECLIPSE simulator from Schlumberger and also writes output files compatible with the output from said simulator, because these are the dominant file formats supported by most pre- and post-processing tools.

2.6.1. Reading input files

The input for a simulation case consists of many separate data items. The computational grid, petrophysical properties, fluid properties, and well scheduling and behavior must all be specified, along with other options controlling for example output.

It is common for commercial simulators to do this in the form of an input file set, often called a *deck* (named for the decks of punched cards used in the early age of simulation). An input deck consists of many keywords that set properties or trigger functionality in the simulator, and may be distributed across several files (using an INCLUDE keyword). The deck is organized into sections dealing with separate parts, and is similar to a programming language that manipulates the reservoir simulator like a state machine. The sections are:

`RUNSPEC`	Dimensions, phases present.
`GRID`	Grid topology and geometry, faults, petrophysical properties.
`EDIT`	Modifying inputs specified in the GRID section, multipliers.
`PROPS`	Fluid properties.
`REGIONS`	Define regions for properties and output.
`SOLUTION`	Initial solution.
`SUMMARY`	Choose variables for time-series output.
`SCHEDULE`	Parameters and controls for wells and well groups, time stepping.

The input format is well documented in the OPM Flow manual [10]. The sections are described in Chapter 3.4 of the manual, and each supported keyword is described in the following chapters. The list of supported keywords is extensive, and the parsing performed in OPM Flow has been tested on a selection of different input decks from industry. For obvious reasons, OPM Flow does not yet support all keywords and features available in the ECLIPSE simulator. Still, OPM Flow is fast reaching a point where the functionality available covers the most common use cases, and is wide enough that it can be feasible to modify unsupported cases to adapt their reservoir description to what is currently supported. We therefore believe the current parsing framework is sufficient to make OPM Flow an attractive alternative in the industry.

Parsing in OPM Flow is a two-stage process. In the first stage, basic string parsing is performed, comments are stripped, included files are read in, default values and multipliers are resolved, and numerical items are converted to the correct type. This produces an instance of class `Opm::Deck`, which is essentially a container of `Opm::DeckKeyword` objects. The second stage in the parsing process consists of creating an instance of type `Opm::EclipseState`, which is an object at a much higher semantic level, in which the interaction between keywords has been taken into account, e.g., all the `EDIT` section modifiers have been applied to produce consistent geological input fields and the various well-related keywords from the `SCHEDULE` section have been assembled into well objects.

The keywords recognized by OPM Flow are specified with a simple JSON schema. A new keyword can be supported by adding a JSON specification and registering it with the build system, then OPM Flow will be able to parse the new keyword and create `Opm::DeckKeyword` instances to represent it. Taking a new keyword into account in the `Opm::EclipseState` class requires changes to the code depending on the semantics of the new keyword. To complete the addition of a new feature, the numerical code can then be extended to use the new information available from the `Opm::EclipseState` object.

2.6.2. Writing output files

OPM Flow can output a range of different file types. Summary files contain time-series for well or reservoir data. Restart files, along with INIT and EGRID files, can be used for restarting a simulation from a report step or checkpoint, as well as for visualization of the reservoir and the solution variables. RFT files are used by engineers to get more information about reservoir behavior.

The files are ECLIPSE-compatible in the sense that they are correctly rendered in independent third-party viewers like Petrel, S3Graf, and RMS, in addition to OPM ResInsight. These binary output files are also compatible to the point that Schlumberger's simulator ECLIPSE can restart a simulation from these files.

The input keywords RPTRST, RPTSOL, and RPTSCHED are used to configure which properties should be written to the restart files, and also for which report steps output should be enabled, most of this output configuration is supported in OPM Flow.

The SUMMARY section is used to configure which result vectors should be added to the summary output. OPM Flow will use the configuration inferred from this section when creating summary output. The list of available summary vectors is extensive and OPM Flow does not support them all, but a wide range of field, group, well, region, and block vectors are supported. In addition, OPM Flow has its own unique PRT file, which has a report formatted just like ECLIPSE for compatibility. However, everything else in the PRT file is unique for OPM Flow. The same goes for the terminal output, which is not the same as ECLIPSE. The ambition is to make output more useful for the end user.

VTK-format output files for result fields can also be written for visualization with, e.g., Paraview. This feature can be activated with a command-line option, see the manual [10] for this and other output-related options.

2.6.3. Parallel file I/O

The ECLIPSE file format is inherently sequential, but since this file format is a de-facto standard in the industry, the parallel simulation program also needs to handle the format. Whereas initial reading of the deck is less performance critical, since it is only done once, the reoccurring writing of simulation data during the simulation poses a potential performance bottleneck. To absorb the penalties imposed by slow file systems, such as network file systems, the actual output is queued in a separate output thread that no longer needs to by synchronized with the simulation. To cater to the sequential file format at hand, a dedicated I/O rank collects all data from all processes and then the sequential output routines can be used as before. Again, this is a potential show stopper for scaling to large number of cores; this has not been an issue yet, but may need to be addressed in future releases of the software. To avoid performances penalties, the communication between the I/O rank and all other ranks is done asynchronously, such that each simulation core only places an MPI_Isend and continues with simulation, whereas the I/O rank only places MPI_Irecv in combination with MPI_Iprobe and MPI_Waitall.

2.7. Extended models

Whereas the family of black-oil models presented thus far is sufficient for many real-world cases, it lacks essential features for describing chemically enhanced oil recovery. This section briefly outlines two extended models implemented in OPM Flow, a solvent model and a polymer model, that add one or a few extra components and model their effect on the fluid flow.

2.7.1. Solvent

Miscible gas injection is a common EOR technique. The injected gas mixes with the hydrocarbons and enhances the sweep efficiency by mobilizing the residual oil in the reservoir. An important application for the solvent module in OPM is to explore the combined goal of CO_2 storage with EOR. The CO_2 is miscible with oil at pressures above a certain threshold, and injecting CO_2 is thus a commonly used approach to enhanced recovery.

The solvent module extends the black-oil model with a fourth component representing the injected gas. In addition to the extra conservation equation required for the solvent, the solvent modifies the relative permeability, the capillary pressure, the residual saturations, the viscosity, and the density of the hydrocarbons. The modeling strategy we use is to compute the properties for the fully miscible and the immiscible case and interpolate between these limits using a miscibility function M depending on pressure and solvent saturation.

Large contrasts in viscosity and density between the injected gas and the oil lead to fingering effects. Resolving individual fingers requires very fine grid resolution and is not feasible for field-scale simulations. The Todd–Longstaff model [37] introduces an empirical parameter ω to formulate effective parameters that take into account the effect of fingers on the fine-scale dispersion between the miscible components to overcome this obstacle. The solvent module in OPM follows the implementation suggested in [37,38], the most important details are included here.

The conservation equation for the solvent component, s, is

$$\frac{\partial}{\partial t}\left(\phi_{\text{ref}} A_s\right) + \nabla \cdot \mathbf{u}_s + q_s = 0, \tag{28}$$

where the accumulation term and the flux are given by

$$A_s = m_\phi b_s s_s, \qquad \mathbf{u}_s = -b_s \lambda_s \mathbf{K}(\nabla p_g - \rho_s \mathbf{g}). \tag{29}$$

In addition, the solvent saturation is added to Eq. (4), giving:

$$s_w + s_o + s_g + s_s = 1. \tag{30}$$

The relative permeability of the solvent and the gas component is given as a fraction of the total (gas + solvent) relative permeability:

$$k_{r,s} = \frac{s_s}{s_g + s_s} k_{r,gt}, \qquad k_{r,g} = \frac{s_g}{s_g + s_s} k_{r,gt}. \tag{31}$$

One can change the gas and solvent fractions multiplying $k_{r,gt}$ with simple functions of the same fractions to allow a more flexible formulation. The total relative permeability is given as an interpolation between the fully miscible relative permeability and the immiscible one:

$$k_{r,gt} = M\frac{s_g + s_s}{s_g + s_s + s_o}k_{r,n}(s_n) + (1 - M)k_{r,g}(s_g + s_s), \tag{32}$$

$$k_{r,o} = M\frac{s_o}{s_g + s_s + s_o}k_{r,n}(s_n) + (1 - M)k_{r,o}(s_w, s_g), \tag{33}$$

where M is a user-defined parameter that varies from zero to one and may depend on both the solvent fraction and the oil pressure. Water-blocking effects are incorporated in the model by further scaling the normalized saturations by effective mobile saturations that increase with increasing water saturation; see [38] for details.

The Todd–Longstaff model formulates the effective viscosities for solvent, gas, and oil as a product of the pure viscosities and the fully mixed viscosity

$$\mu_{\text{eff},o} = \mu_o^{1-\omega}\mu_{m,os}^{\omega}, \quad \mu_{\text{eff},g} = \mu_g^{1-\omega}\mu_{m,gs}^{\omega}, \quad \mu_{\text{eff},s} = \mu_s^{1-\omega}\mu_{m,gos}^{\omega}, \tag{34}$$

where the fully mixed viscosities are defined using standard $\frac{1}{4}$-power fluid mixing rule [37]. Pressure effects on the mixing are modeled by replacing the single parameter ω with an effective mixing parameter

$$\omega_{\text{eff}} = \omega\omega_p. \tag{35}$$

where ω_p is a function of pressure [38,39]. For effective density calculations confer [38]. The solvent model is used to simulate the SPE 5 benchmark in Section 3.3.

2.7.2. Polymer

Polymer flooding is another widely used EOR technique. The major mechanism of polymer flooding is to improve the sweep efficiency by increasing the viscosity of the drive fluid to improve the mobility ratio. The polymer model in OPM assumes that the polymer component is only transported within the water phase and only affects the properties of this phase. The model also includes the effects of dead pore space, polymer adsorption in the rock, and permeability reduction effect. Shear-thinning/thickening non-Newtonian fluid rheology is modeled by a logarithm look-up calculation method based on the water velocity or shear rate.

The polymer model is developed by adding the following polymer transport equation to the black-oil model

$$R_{p,i} = \frac{V_i}{\Delta t}(A_{p,i} - A_{p,i}^0) + \sum_{j\in C(i)} F_{p,ij} + q_{w,i}c_{w,i} = 0, \tag{36}$$

where

$$A_{p,i} = \phi_{\text{ref},i}m_\phi b_r b_w s_w(1 - s_{dpv})c + \rho_r C^a(1 - \phi_{\text{ref},i}) \tag{37}$$

$$F_p = b_w v_p c \tag{38}$$

$$(b_w v_p)_{ij} = \left(\frac{m_T k_{r,w}}{\mu_{\text{eff},p}R_k}\right)_{U(w,ij)} T_{ij}\Delta H_{w,ij}, \tag{39}$$

$$(b_w v_w)_{ij} = \left(\frac{m_T k_{r,w}}{\mu_{\text{eff},w}R_k}\right)_{U(w,ij)} T_{ij}\Delta H_{w,ij}. \tag{40}$$

Here, c is polymer concentration, b_r is rock expansion factor, s_{dpv} is the dead pore space, C^a is the polymer adsorption concentration, v_p is polymer flux rate, and $\mu_{\text{eff},p}$ is the effective polymer viscosity, which is a function of the polymer concentration c and pressure of the water phase. The viscosity of water is changed as a result of dissolution of polymer, and we calculate the water flux rate based on effective water viscosity $\mu_{\text{eff},w}$, which is also a function of polymer concentration. The polymer concentration $c_{w,i}$ used to calculate the polymer rate through well connections in grid block i is set as the injection concentration for injectors, whereas $c_{w,i} = c_i\mu_{\text{eff},w}/\mu_{\text{eff},p}$ for producers, where c_i is the polymer concentration in grid block i. The effective viscosities are calculated using the Todd–Longstaff mixing model [37] to handle sub-grid fingering effects, similar to the solvent model of Section 2.7.1.

3. Numerical examples and results

This section demonstrates OPM Flow through a series of numerical examples. Sections 3.1 to 3.3 report small synthetic cases exercising particular parts of the simulator's fluid behavior. Section 3.4 reports a more realistic case to highlight the treatment of wells. The Norne case in Section 3.5 is a real field case and demonstrates that OPM Flow can be applied to complex industrial field cases. The polymer example in Section 3.6 shows some of the EOR capabilities of OPM Flow.

We conclude the section with some notes on performance. Complete input decks used for the examples are available from github.com/OPM, and can be run with default options.

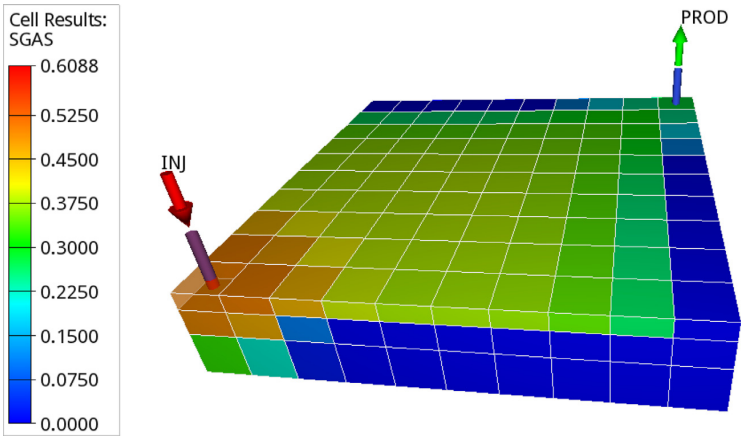


Fig. 10. Gas saturation of the SPE 1 benchmark case after a period of gas injection. The z-axis has been exaggerated 20 times.

3.1. SPE 1 benchmark

The Society of Petroleum Engineers (SPE) has run a series of comparative solutions projects with the aim of comparing and benchmarking different simulators or algorithms. The first such project (nicknamed SPE 1), was introduced for benchmarking three-dimensional black oil simulation [40]. Two examples were suggested, and here we use the second case to demonstrate OPM Flow. The grid consists of $10 \times 10 \times 3$ cells covering a computational domain of 10 000 ft $\times$ 10 000 ft $\times$ 100 ft. The thickness of each layer is 20 ft, 30 ft, and 50 ft respectively. Two wells are located in two opposite corners. The first well injects gas from the top layer at a rate of 100 MMscf/day with a BHP limit of 9014 psia. The other well is set to produce oil from the bottom layer with a production target of 20 000 stb/day and BHP limit of 1000 psia. The reservoir is initially undersaturated. The participants were asked to report the oil production over time and the gas–oil ratio (GOR) over time. Fig. 10 shows a view of the case after a few years of injection, whereas Fig. 11 verifies that there is excellent agreement in the well responses computed by OPM Flow and the commercial simulator ECLIPSE 100.

3.2. SPE 3 benchmark

The third comparative solution project [41] was introduced to study gas cycling of retrograde condensate reservoirs using compositional modeling. Based on the data initially provided as the second case in [41], we designed a benchmark test case for the OPM Flow simulator.

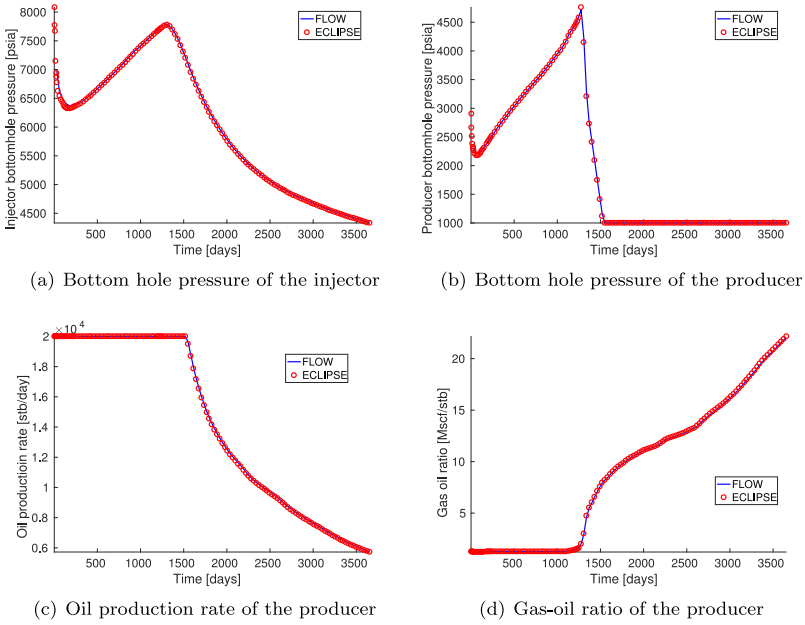
Black-oil properties are generated from compositional data using PVTsim Nova. The dimension of the grid is $9 \times 9 \times 4$ with 293.3 ft $\times$ 293.3 ft cell sizes in the horizontal direction and the thickness of each layer being 30 ft, 30 ft, 50 ft and 50 ft, respectively. The horizontal permeability for each layer is 130 md, 40 md, 20 md, and 150 md respectively, whereas the vertical permeabilities are 13 md, 4 md, 2 md, and 15 md.

An injector is located in the corner (cell column (1, 1)) and perforates the top two layers, whereas a producer perforates the bottom two layers of cell column (7, 7). The simulation time is 14 years in total. The gas injection rate is 5700 Mscf/day for the first four years, and 3700 Mscf/day for the next five years. No gas is injected the last five years. The BHP limit is a constant 4000 psia. The production well operates with a constant gas production limit 5200 Mscf/day and a constant BHP limit of 500 psia. Fig. 12 confirms good agreement for bottom-hole pressures and rates obtained by OPM Flow and ECLIPSE.

3.3. SPE 5 benchmark

The fifth comparative solution project [42] is designed to compare four-component miscible simulators with fully compositional simulators for three different miscible gas injection scenarios. Herein, we compare the solvent simulator in OPM Flow with the solvent simulator in ECLIPSE for the three cases introduced in the SPE 5 paper. The same grid with dimension $7 \times 7 \times 3$ and the same fluid properties are used for all the cases. A water–alternating-gas (WAG) injector is located in Cell (1, 1, 1). By changing the alternation sequence and rates of water and gas, different miscibility conditions are obtained in the three different scenarios. A producer located in Cell (7, 7, 3) is controlled by the same oil production rate in all three cases; see [42] for details, or confer the input deck given in opm-data.

Fig. 13 shows that, except for some minor discrepancies in the gas production rate and the injector bottom-hole pressure, there is general agreement between the well responses computed by OPM Flow and those computed by the



(a) Bottom hole pressure of the injector (b) Bottom hole pressure of the producer

(c) Oil production rate of the producer (d) Gas-oil ratio of the producer

Fig. 11. Well responses computed for the SPE 1 benchmark, Case 2.

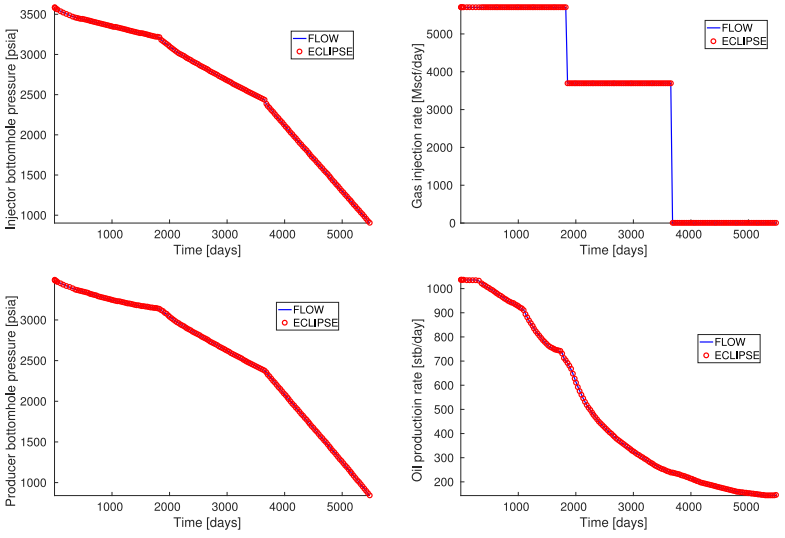


Fig. 12. Bottom-hole pressures and well rates of the SPE 3 benchmark, Case 2.

solvent module in ECLIPSE. A closer investigation reveals that the discrepancies are mainly caused by differences in how the density mixture of the Todd–Longstaff model is implemented. To the best of our knowledge, both simulators use the same model, but the complexity of the model may lead to differences in the actual implementation.

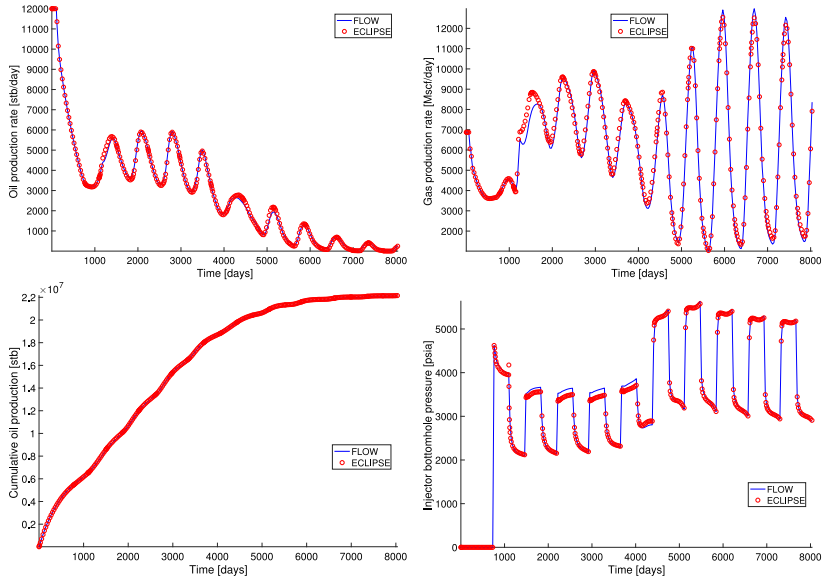


Fig. 13. Oil production rate, gas production, and cumulative oil production of the producer, as well as bottom-hole pressure of the injector for the SPE 5 benchmark, Case 1.

3.4. SPE 9 benchmark

The ninth comparative solution project [43] is designed to re-examine black-oil simulation. The model includes a rectilinear grid with 9000 grid cells and the dimension is 24 × 25 × 15 (Fig. 14). Originally, the grid was provided in conventional rectangular coordinates. In opm-data, we provide data sets also in the corner-point grid format. Cell (1, 1, 1) is at a depth of 9000 ft, and the remaining part of the grid is dipping in the x-direction with an angle of 10 degrees. The model has a layered porosity structure and highly heterogeneous permeability field. The well pattern consists of 25 rate-controlled producers and one injector in the corner of the grid.

The total simulation time is 900 days. The injector is controlled by water injection rate with target rate 500 stb/day and BHP limit 4000 psia. The producers are under oil rate control with BHP limit 1000 psia. The production rate target is 1500 stb/day for all the producers, except between 300 and 360 days, when the rate target is temporarily lowered to 100 stb/day.

This benchmark exercises in particular the well model of the simulator and its interaction with the reservoir. It also has nontrivial phase behavior, as initially there is no free gas in the reservoir, but as pressure is reduced below the bubble point, free gas appears near the top, as can be seen in Fig. 14. Fig. 15 shows that the well curves computed by OPM Flow and ECLIPSE are in excellent agreement.

3.5. Norne

Norne is a Norwegian Sea oilfield that has been in production since 1997. The operators of the field have chosen to share data, making it the only real field simulation case that was openly available until 2018, when the Volve field data were also made open. The Norne field was initially operated using alternating water and gas injection, and in total 36 wells have been opened, not all being active at the same time. The simulation model uses a corner-point grid with sloping pillars and several faults, so the grid has many non-logical-Cartesian connections and is therefore completely unstructured and challenging to process (see Fig. 16).

The Norne field model includes features such as hysteresis, end-point scaling, oil vaporization control, and multiple saturations regions. For a full description or to test the model, confer to github.com/OPM. Fig. 17 shows that there also in this case is excellent match between OPM Flow and ECLIPSE, despite the complexity of the model.

3.6. Polymer injection enhanced oil recovery (EOR) example

To demonstrate the use of the polymer functionality of OPM, we consider a simple 3D example taken from [44]. The grid consists of 2778 active cells and is generated with MRST [4]. The reservoir has physical extent of 1000 m × 675 m

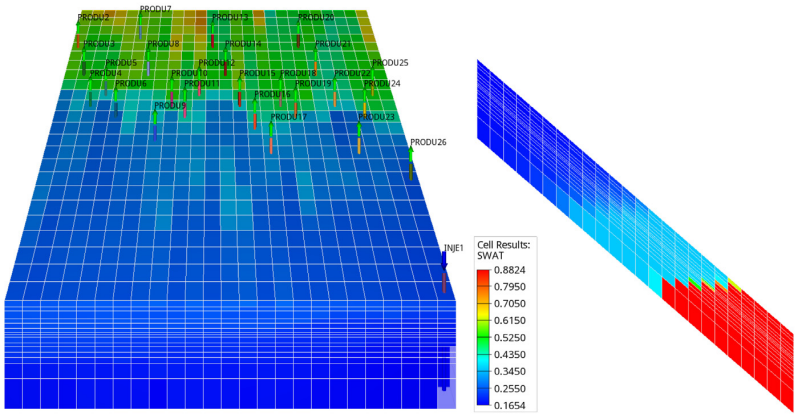


Fig. 14. Left: frontal view of the SPE 9 benchmark case showing free gas saturation at the end of the simulation, and all wells. Right: side view showing initial water saturation. The z-axis is exaggerated 5 times in both plots.

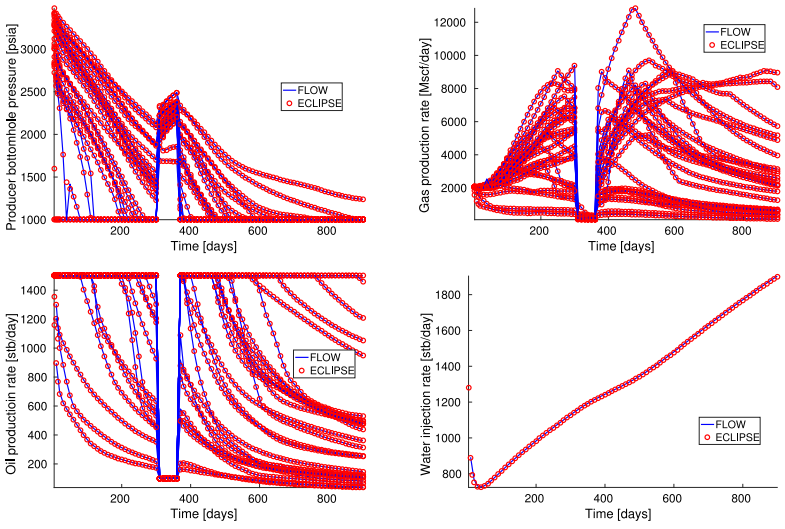


Fig. 15. Bottom hole pressures, gas production rates, and oil production rates for all producers in the SPE 9 benchmark. Water rate is reported for the injector.

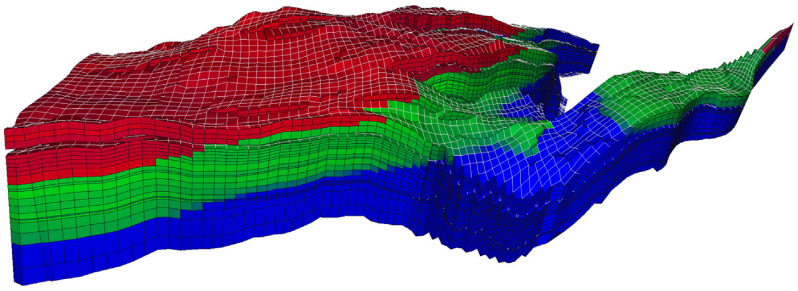


Fig. 16. Grid for the Norne field case, with initial fluid distribution. Red is gas, green is oil, and blue is water. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

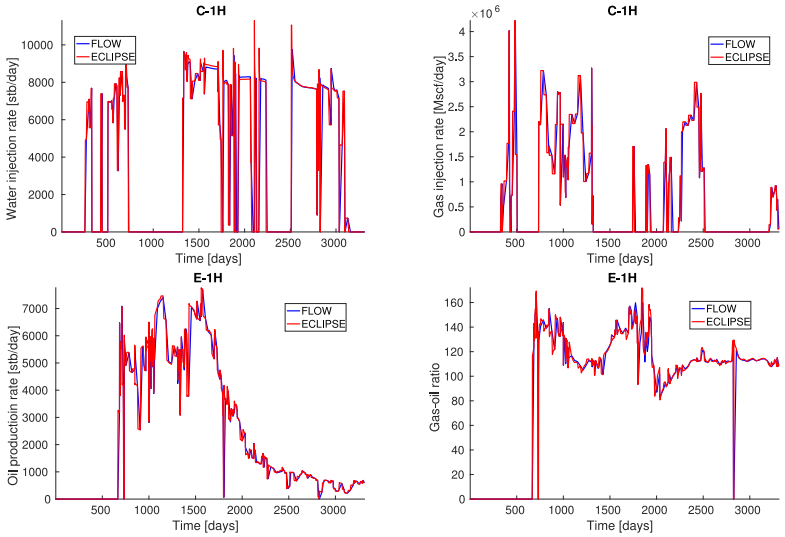


Fig. 17. Well responses for injector C-1H and producer E-1H of the Norne field.

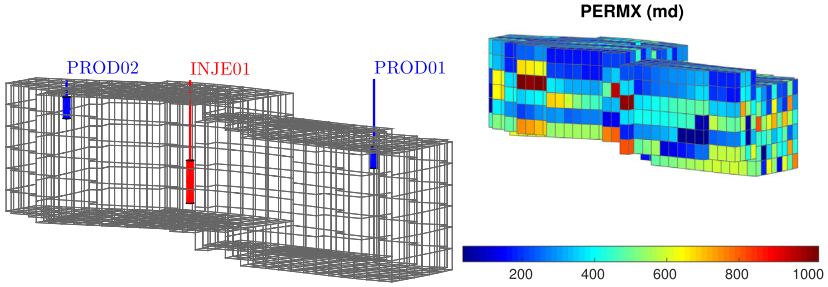


Fig. 18. Computing grid and horizontal permeability of the polymer example.

$\times$ 212 m and contains one injector and two producers (Fig. 18). The injector is under rate control with target rate 2500 m^3/day and bottom-hole pressure limit 290 bar. The producers are under bottom-hole pressure control with a bottom-hole pressure target of 230 bar.

The flooding process begins with a 560-day pure water injection, then a 400-day polymer injection with concentration 1.0 kg/m^3, followed by a 1530-day pure water injection. Four different configurations are compared. In the first, no polymer is injected. The second includes polymer, but does not account for any shear effect. The third models shear-thinning effects, whereas the fourth includes shear-thickening effects.

Fig. 19 reports well responses for all four configurations. The bottom-hole pressure (bhp) of the injector increases when the polymer injection process starts. When no-shear effect is involved, the bhp reaches the limit, and the injector switches to bhp control mode, which causes decreased injection rate. This is a natural consequence of reduced injectivity from the polymer injection. With shear-thinning effect, the bhp of the injector still increases but manages to stay below the limit. As a result, the target injection rate is maintained. With shear-thickening effect, the injection rate decreases further compared with the simulation that disregarded the shear effect. We emphasize that this example is designed for verification of the polymer functionality of OPM and not related to any real polymer flooding practice.

For comparison purposes, results from ECLIPSE are also included in Fig. 19 with thick dashed lines. Good agreement is observed between the simulators for all four configurations.

3.7. Numerical performance

Reservoir simulators are complex, and analyzing numerical performance is involved. Hence, we can only present a brief overview here. In terms of performance, a well-implemented reservoir simulator can be divided in two operations

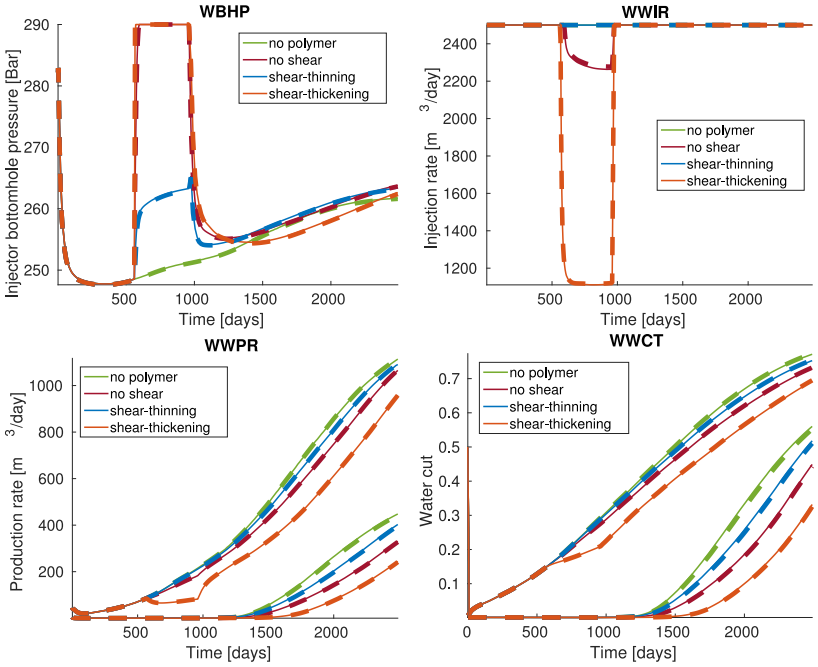


Fig. 19. Well responses for the polymer flooding case. The top plots show bottom-hole pressure (left) and water rate (right) for the injector. The bottom plots show water rates and water cuts for the two producers. Results from OPM Flow are shown with thin lines, ECLIPSE results with thick dashed lines.

of almost equal importance. The first operation is the assembly or linearization, i.e., calculating the Jacobian matrix. The second is the preconditioning and solution of the resulting linear system. For OPM Flow, the assembly part may take approximately half the simulation time with the linear solver accounting for the remaining half. However, this is strongly case-dependent, and the balance changes with the complexity of the fluids used, length of time steps, etc. Before it makes sense to compare OPM Flow computationally to other implementations, we need to point out a few prominent implementation choices. Use of automatic differentiation makes the calculation of the Jacobian less error prone, but requires more computations than a direct approach, even though implementation of automatic differentiation using SIMD instructions reduces parts of this overhead, as shown in [32].

For the linear solver part, it is the preconditioner that uses most of the computational time. Multiple preconditioners are available in OPM Flow. The default is incomplete LU-factorization with zero fill-in. This seems to be a good choice for small and medium sized problems. For larger cases, CPR preconditioning with AMG is preferable, and OPM Flow provides this as an option. For the linear solver itself, a biconjugate gradient method is used by default.

3.7.1. Serial performance

For very small cases like SPE1 and SPE3, the serial performance of OPM Flow is comparable to that of ECLIPSE 100. The runs typically finish faster with OPM Flow, since the simulator does not need to check for licenses. Since both cases finish in a couple seconds on a modern CPU, performance is typically not important, though. Moving over to SPE9, which is more complex, the picture changes in favor of ECLIPSE 100. Actually, ECLIPSE 100 is approximately twice as fast on SPE9 compared to OPM Flow. Still, the case finishes in around ten seconds on a modern CPU, so it has not been considered important for performance tuning in OPM Flow. So far, the numerical efforts in OPM Flow have been centered around two full-field simulation cases. The first is Norne, which is openly shared with the community. The Norne model is very different from the SPE cases, both in terms of being more complex, and in terms of run times. A serial run of the Norne case typically takes around ten minutes on a fast CPU. The run times of OPM Flow and ECLIPSE 100 are similar, with about ten percent run time advantage to ECLIPSE 100 when using OPM Flow from precompiled packages. Compiling OPM Flow from source with more recent compiler and more aggressive compiler switches, the tables turn in favor of OPM Flow.

In addition, a proprietary full-field model has been used to guide implementation, a model which is computationally more intensive than Norne. The model is unfortunately only available through a non-disclosure agreement, and hence cannot be shared openly. The model has more than 100 000 cells, dozens of injectors and producers that penetrate the reservoir vertically and horizontally, and complex geology with faults and thin layers. The run time is approximately four

Table 1
OPM Flow performance indicators for simulations of the Norne model. The columns display number of MPI-processes (procs), threads per process during assembly (thr(A)), compute nodes involved, assembly execution time (AT), linear solver execution time (LST), total execution time (TT), total number of linear iterations (iter), speedup and parallel efficiency (eff) with respect to serial simulation. The computations were performed on four dual AMD EPYC 7601 2.2 GHz 32-core processors using 1, 2, …, 64 MPI-processes. The AMD nodes have 8 memory lanes per socket and a 170.6 GB/s memory bandwidth, and the interconnect between the nodes is HDR InfiniBand (200 Gbits/s).

Procs	thr(A)	Nodes	AT (s)	LST (s)	TT (s)	Iter	Speedup	eff
1	1	1	351.49	300.59	691.26	25744	–	–
2	1	1	180.73	191.14	401.89	25331	1.72	0.86
4	1	1	95.71	112.49	229.94	25148	3.00	0.75
8	1	1	63.51	86.43	168.35	25388	4.11	0.51
16	4	1	21.73	50.93	88.72	25736	7.79	0.49
32	4	2	15.82	54.16	85.67	26123	8.07	0.25
48	4	3	10.98	45.65	71.84	25920	9.62	0.20
64	4	4	8.90	49.92	74.16	26461	9.32	0.15

times that of the Norne model. On single-case and ensemble runs of this model, OPM Flow performance is generally better than ECLIPSE 100.

A comparison of performance of the solvent extension of the OPM Flow simulator is reported in [45] and shows that OPM Flow is significantly faster than ECLIPSE 100 for the solvent extension.

3.7.2. Parallel scaling results

Most commercial reservoir simulators use distributed memory with message passing interface (MPI) for their parallel implementation. Furthermore, domain decomposition of the reservoir grid is done to split the load between the processes. How this is done varies between simulators. The ECLIPSE 100 simulator decomposes the domain along one axis, while newer simulators tend to use graph-based partitioners such as Zoltan (used by OPM Flow) or Metis (used by INTERSECT). Similarly to ECLIPSE 100, the current version of OPM Flow does not support splitting a well between domains.

Tests of parallel scaling have been performed both on the Norne model and on the proprietary model. Scaling on both models are similar, and as of the 2019.10 release, OPM Flow scales well on both models up to 16 cores. Scaling beyond 16 cores cannot be expected at this point because of convergence issues. Comparing to ECLIPSE 100, the scaling of ECLIPSE 100 is similar to OPM Flow up to 16 CPUs on the Norne model despite its simpler partitioning approach. However, the limitations really show for the more computationally intensive proprietary model, where ECLIPSE 100 is back to the single CPU run time when using four CPUs. Attempting to use eight or 16 CPUs, ECLIPSE 100 will only increase run times to be much slower than the single CPU time.

Benchmark timing results are available for both models, with up to four and eight CPU cores, on linuxbenchmarking.com. These results are used by the developers for quality assurance, making sure that changes to the simulator do not have unintended adverse consequences for serial or parallel performance.

Table 1 reports parallel performance for Norne. Each simulation was repeated several times, and the best time measurements are displayed. Because 16 MPI-processes per node is sufficient to fully utilize a node's memory bandwidth, adding additional MPI-processes after 16 on a single node will not improve the performance of the bandwidth-bound linear solver. Simulations using more than 16 MPI-processes were therefore run on multiple nodes. Since the matrix assembly is also parallelized using threads in addition to the MPI parallelization, idle cores can be partially exploited by using additional threads to improve matrix assembly performance. We therefore use 4 threads per MPI-processes when assembling for simulations with 16 or more MPI-processes. Hence all available cores of a compute node are used at this stage. In addition to time measurements, we report speedup and efficiency with respect to the sequential simulation, as well as the total number of ILU0-BiCG-stab iterations needed to complete each simulation.

We observe improved total execution time for the parallel simulations, but also notice that the parallel efficiency drops when the number of MPI-processes increases. We observe that the matrix assembly code scales fairly well. Unfortunately, the linear solver does not see much improvement in performance when using more than 16 MPI-processes. Since the total number of linear iterations needed to complete each simulation remains almost constant, we can eliminate poor solver convergence as a reason for a drop in parallel efficiency. Therefore, the poor scaling of the linear solver for more than 16 processors can mainly be attributed to communication overhead. The lack of scaling in the I/O operations also impacts the overall parallel performance of OPM Flow. For example, when using 48 and 64 processors, the I/O operations account for a larger proportion of total execution time than the matrix assembly.

3.7.3. Ensemble simulation performance

Running ensembles of model realizations is commonly done in history matching and optimization studies. The run time of each realization can vary significantly due to instability in the nonlinear solution process. As a measure of the

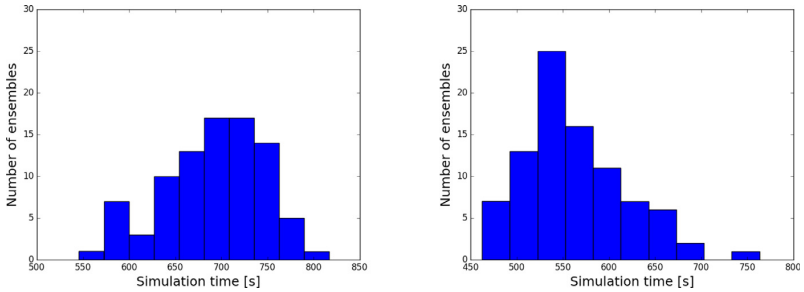


Fig. 20. Histogram over simulation times for an ensemble of Norne realizations for OPM Flow (left) and ECLIPSE (right).

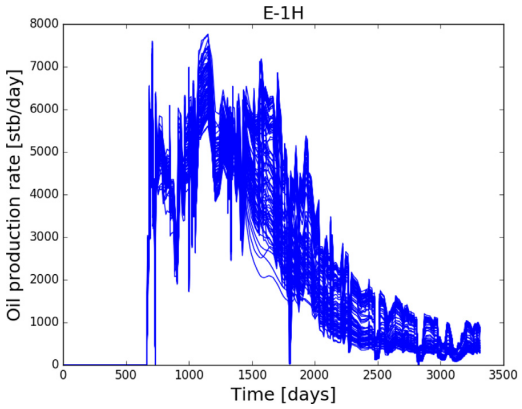


Fig. 21. Oil production rates for 100 realizations of the Norne model.

performance of OPM Flow, we therefore show results for a full ensemble and compare these with the ECLIPSE simulator. All ensemble members use the same grid, but permeability, relative permeability endpoints, and fault multipliers are varied to represent the uncertainty in the geological properties. The ensemble can be acquired from github.com/rolfjl/Norne-Initial-Ensemble; for more details we refer to [46].

The individual simulations were run in serial mode on an Intel i7-6700 CPU @ 3.40 Ghz with 16 GiB memory. Fig. 20 shows the resulting run time distributions. The mean run time for OPM Flow on the ensembles is 690 s, whereas for ECLIPSE it is 562 s, making OPM Flow approximately 25% slower than ECLIPSE on average. The ensemble is modified compared to the base case referred to in Section 3.7.1, so the results are not completely equivalent. We still expect that the difference can be reduced with aggressive compiler options and light tuning. Both simulators run through all ensemble members without having severe convergence issues. Tuning of numerical parameters such as tolerances can sometimes improve simulator performance. For this comparison, both simulators have been run with their respective default tuning parameters. Fig. 21 reports oil production results for a single well for the whole ensemble. Since unmatched realizations are used, there is a large spread in the results. This spread will typically be reduced if the models are constrained on the historical data using a history matching procedure.

4. Summary and outlook

This paper has described OPM Flow at the time of writing. The software is under continuous development, so changes and additions should be expected. New features in development include adjoint calculations, higher-order discretizations in space and time [47], new fluid models for CO_2 behavior, and sequential implicit methods. We think that at this point OPM forms a robust base for both research and industrial applications, and that going forward this combination will help shorten the research cycle by giving researchers an industrial-strength platform on which to build and test their methods, as well as satisfying engineering requirements.

CRediT authorship contribution statement

Atgeirr Flø Rasmussen: Conceptualization, Methodology, Software, Validation, Writing - original draft, Writing - review & editing, Supervision, Project administration, Funding acquisition. **Tor Harald Sandve:** Conceptualization, Methodology, Software, Validation, Writing - original draft, Writing - review & editing, Supervision, Project administration, Funding acquisition. **Kai Bao:** Methodology, Software, Validation, Writing - original draft. **Andreas Lauser:** Conceptualization, Methodology, Software, Validation, Writing - original draft, Supervision. **Joakim Hove:** Software, Validation, Writing - original draft, Supervision. **Bård Skaflestad:** Software, Validation. **Robert Klöfkorn:** Software, Writing - original draft, Writing - review & editing, Supervision, Funding acquisition. **Markus Blatt:** Conceptualization, Software, Validation, Writing - original draft, Supervision. **Alf Birger Rustad:** Conceptualization, Software, Validation, Resources, Data curation, Writing - original draft, Project administration, Supervision, Funding acquisition. **Ove Sævareid:** Software, Validation, Writing - review & editing. **Knut-Andreas Lie:** Conceptualization, Methodology, Writing - original draft, Writing - review & editing, Supervision, Funding acquisition. **Andreas Thune:** Software, Validation, Writing - original draft.

Acknowledgments

The authors thank Schlumberger for providing them with academic software licenses to ECLIPSE, used for comparing simulation results with OPM Flow. Work on OPM Flow and this article has been supported by Gassnova through the Climit Demo program under contract 617115 and by Equinor.

Robert Klöfkorn, Tor Harald Sandve, and Ove Sævareid acknowledge the Research Council of Norway and the industry partners (ConocoPhillips Skandinavia AS, Aker BP ASA, Vår Energi AS, Equinor ASA, Neptune Energy Norge AS, Lundin Norway AS, Halliburton AS, Schlumberger Norge AS, and Wintershall DEA) of the National IOR Centre of Norway (230303) for support.

Appendix A. Nomenclature

A.1. Symbol definitions for the continuous equations

ϕ	*porosity*, can be a function of (oil) pressure: $\phi = m_\phi(p_o)\phi_{\text{ref}}$.
m_ϕ	*pore volume multiplier* as function of pressure.
ϕ_{ref}	*reference porosity*, a constant in time but varying in space.
p_α	*phase pressure* for phase α.
b_α	*shrinkage/expansion factor* for phase α defined as the ratio of surface volume at standard conditions to reservoir volume for a given amount of fluid: $b_\alpha = V_{\text{surface},\alpha}/V_{\text{reservoir},\alpha}$. For the oil phase, b_o is called *shrinkage factor*, whereas b_g is called *expansion factor*. The reciprocal quantity is called *formation volume factor*, and is usually denoted with a capital B: $B_\alpha = 1/b_\alpha$. Usually a function of phase pressure and composition: $b_\alpha = b_\alpha(p_\alpha, r_{go}, r_{og})$. See Section 2.1.6.
r_{go}	*ratio of dissolved gas to oil* in the oleic phase. Often called r_S in other literature.
r_{og}	*ratio of vaporized oil to gas* in the gaseous phase. Often called r_V in other literature.
s_α	*saturation* of phase α, the pore volume fraction occupied by the phase. The saturations of all phases sum to 1.
$p_{c,\alpha\beta}$	*capillary pressure* between phases α and β. Typically a function of saturation: $p_{c,\alpha\beta} = p_{c,\alpha\beta}(s_\alpha)$.
$\mathbf{K}$	*permeability* of the porous medium.
$k_{r,\alpha}$	*relative permeability* for phase α, modeling the reduction in effective permeability for a fluid phase in the presence of other phases. Typically a function of saturation, see Section 2.1.5.
μ_α	*viscosity* of phase α, typically a function of phase pressure and composition.
λ_α	*mobility* of phase α, given by $\lambda_\alpha = k_{r,\alpha}/\mu_\alpha$.
$\mathbf{v}_\alpha$	*phase velocity* of phase α.
$\mathbf{u}_\alpha$	*component velocity* of component α.
$\rho_{S,\alpha}$	*surface density* of phase α at one atmosphere, a given constant.
ρ_α	*density* of phase α in the reservoir. For water, $\rho_w = b_w\rho_{S,w}$. For the oleic phase the relationship is more complex since it must include dissolved gas: $\rho_o = b_o(\rho_{S,o} + r_{go}\rho_{S,g})$, similar for the gaseous phase: $\rho_g = b_g(\rho_{S,g} + r_{og}\rho_{S,o})$.
$\mathbf{g}$	*gravitational acceleration* vector.
q_α	*well outflux density* of pseudo component α, (negative for well inflows). The form of this term depends on the well model used, see Section 2.2.

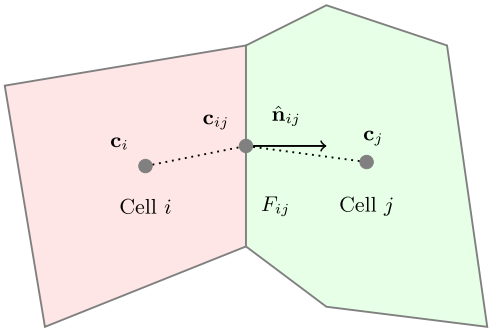


Fig. B.22. Quantities used to compute the transmissibility between two cells.

A.2. Symbol definitions for the discrete equations

V	*cell volume*.
Δt	*time step length* for the current Euler step.
v_α	*volume flux* of phase α (oriented quantity).
u_α	*surface volume flux* of pseudocomponent α (oriented quantity).
T_{ij}	*transmissibility* factor for a connection, derived from permeability, see Appendix B.
m_T	*transmissibility multiplier* as function of pressure.
$C(i)$	*connections* from cell i, i.e. the set of cells connected to it.
$U(\alpha, ij)$	*upwind cell* for phase α for the connection between cells i and j.
$\Delta\Phi_{\alpha,ij}$	*potential difference* for phase α for the connection between cells i and j (oriented quantity).
g	*gravitational acceleration* in the z-direction.
z_i	*depth* of center of cell i.

A.3. Symbol definitions for the well models

$T_{w,j}$	connection transmissibility factor.
$M_{\alpha,j}$	the mobility for phase α at the connection j.
p_j	pressure of the grid block that contains the connection j.
$p_{bhp,w}$	the bottom-hole pressure of the well w.
$h_{w,j}$	pressure difference within the wellbore between connection j and the well's bottom-hole datum depth.

Appendix B. Transmissibility

The transmissibility T_{ij} of a connection between two cells is a discrete measure of the fluid flow capacity of that connection. We can define T_{ij} by requiring that the discrete flow equations (9), (10), and (11) should be satisfied for a piecewise homogeneous medium, by a velocity field satisfying Darcy's law. For a single, incompressible fluid $b = 1$ and the discrete equations reduce to

$$v_{ij} = \frac{1}{\mu} T_{ij}\big(p_i - p_j - g\rho_{ij}(z_i - z_j)\big). \tag{B.1}$$

We can ignore gravity since the formula must hold also normal to the gravity direction, and assume without loss of generality that the viscosity is 1, yielding the simplified discrete equation

$$v_{ij} = T_{ij}\big(p_i - p_j\big) \tag{B.2}$$

and the simplified Darcy law

$$\mathbf{v} = -\mathbf{K}\nabla p. \tag{B.3}$$

Consider Fig. B.22, where $\mathbf{c}_i$ and $\mathbf{c}_j$ are the centroids of cells i and j and $\mathbf{c}_{ij}$ is the centroid of their common face F_{ij}. Then the flux v_{ij} is given by

$$v_{ij} = \int_{F_{ij}} \mathbf{v} \cdot \hat{\mathbf{n}}_{ij}\, dA \tag{B.4}$$

where $\hat{\mathbf{n}}_{ij}$ is the unit normal of the face F_{ij}. This can be approximated by evaluating at the face centroid and using the Darcy formula (B.3) in each of the cells i and j by

$$v_{ij} = -|F_{ij}|(p_{ij} - p_i)\frac{\mathbf{K}_i(\mathbf{c}_{ij} - \mathbf{c}_i)}{|\mathbf{c}_{ij} - \mathbf{c}_i|^2} \cdot \hat{\mathbf{n}}_{ij},\ v_{ij} = -|F_{ij}|(p_{ji} - p_j)\frac{\mathbf{K}_j(\mathbf{c}_{ij} - \mathbf{c}_j)}{|\mathbf{c}_{ij} - \mathbf{c}_j|^2} \cdot \hat{\mathbf{n}}_{ij}, \tag{B.5}$$

where $|F_{ij}|$ is the area of the face F_{ij} between cells i and j, p_{ij} is the pressure in cell i at the point $\mathbf{c}_{ij}$, and p_{ji} the pressure in cell j at $\mathbf{c}_{ij}$. We then define the half-transmissibility t_{ij} of cell i with respect to F_{ij} by

$$t_{ij} = |F_{ij}|\frac{\mathbf{K}_i(\mathbf{c}_{ij} - \mathbf{c}_i)}{|\mathbf{c}_{ij} - \mathbf{c}_i|^2} \cdot \hat{\mathbf{n}}_{ij} \tag{B.6}$$

with a similar definition for t_{ji} notably using $\hat{\mathbf{n}}_{ji} = -\hat{\mathbf{n}}_{ij}$. Then

$$v_{ij} = -t_{ij}(p_{ij} - p_i), \qquad v_{ij} = t_{ji}(p_{ji} - p_j). \tag{B.7}$$

Requiring that the interface pressures p_{ij} and p_{ji} are equal, we get

$$v_{ij} = T_{ij}(p_i - p_j) \qquad T_{ij} = \frac{t_{ij}t_{ji}}{t_{ij} + t_{ji}} \tag{B.8}$$

Finally, as a modeling tool one may multiply the transmissibilities by a pressure-dependent multiplier m_T. This can be used for example to model sub-scale features such as fractures opening and closing due to pressure changes, thereby changing overall permeability, without explicitly representing such fractures.

References

[1] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, M. Ohlberger, O. Sander, A generic grid interface for parallel and adaptive scientific computing. Part I: Abstract framework, Computing 82 (2–3) (2008) 103–119, http://dx.doi.org/10.1007/s00607-008-0003-x.

[2] B. Flemisch, M. Darcis, K. Erbertseder, B. Faigle, A. Lauser, K. Mosthaf, S. Müthing, P. Nuske, A. Tatomir, M. Wolff, R. Helmig, DuMux: DUNE for multi-{phase, component, scale, physics,...} flow and transport in porous media, Adv. Water Resour. 34 (9) (2011) 1102–1112, http://dx.doi.org/10.1016/j.advwatres.2011.03.007.

[3] E.G. Boman, U.V. Catalyurek, C. Chevalier, K.D. Devine, The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring, Sci. Program. 20 (2) (2012) 129–150, http://dx.doi.org/10.3233/SPR-2012-0342.

[4] K.-A. Lie, S. Krogstad, I.S. Ligaarden, J.R. Natvig, H.M. Nilsen, B. Skaflestad, Open-source MATLAB implementation of consistent discretisations on complex grids, Comput. Geosci. 16 (2) (2011) 297–322, http://dx.doi.org/10.1007/s10596-011-9244-4.

[5] S. Krogstad, K.-A. Lie, O. Møyner, H.M. Nilsen, X. Raynaud, B. Skaflestad, MRST-AD - an open-source framework for rapid prototyping and evaluation of reservoir simulation problems, in: Reservoir Simulation Symposium, Houston, Texas, USA, 23–25 February, 2015, http://dx.doi.org/10.2118/173317-MS.

[6] K.-A. Lie, An Introduction to Reservoir Simulation Using MATLAB/GNU Octave: User Guide for the MATLAB Reservoir Simulation Toolbox (MRST), Cambridge University Press, 2019, http://dx.doi.org/10.1017/9781108591416.

[7] O. Møyner, S. Krogstad, K.-A. Lie, The application of flow diagnostics for reservoir management, SPE J. 20 (2) (2014) 306–323, http://dx.doi.org/10.2118/171557-PA.

[8] R.D. Neidinger, Introduction to automatic differentiation and MATLAB object-oriented programming, SIAM Rev. 52 (3) (2010) 545–563, http://dx.doi.org/10.1137/080743627.

[9] J. Jansen, Adjoint-based optimization of multi-phase flow through porous media a review, Comput. & Fluids 46 (1) (2011) 40–51, http://dx.doi.org/10.1016/j.compfluid.2010.09.039, 10th ICFD Conference Series on Numerical Methods for Fluid Dynamics (ICFD 2010).

[10] D. Baxendale, A.F. Rasmussen, A.B. Rustad, T. Skille, T.H. Sandve, OPM Flow Documentation Manual, Open Porous Media Initiative, 2017, URL https://opm-project.org/?page_id=955.

[11] J.E. Killough, Reservoir simulation with history-dependent saturation functions, SPE J. 16 (1) (1976) 37–48, http://dx.doi.org/10.2118/5106-PA.

[12] F.M. Carlson, Simulation of relative permeability hysteresis to the non-wetting phase, in: SPE Annual Technical Conference & Exhibition, San Antonio, Texas, USA, Society of Petroleum Engineers, 1981, http://dx.doi.org/10.2118/10157-MS.

[13] J. Holmes, Enhancements to the strongly coupled, fully implicit well model: wellbore crossflow modeling and collective well control, in: SPE Reservoir Simulation Symposium, Society of Petroleum Engineers, 1983.

[14] J. Holmes, T. Barkve, O. Lund, Application of a multisegment well model to simulate flow in advanced wells, in: European Petroleum Conference, Society of Petroleum Engineers, 1998.

[15] R.M. Fonseca, E. Della Rossa, A.A. Emerick, R.G. Hanea, J.D. Jansen, Overview of the Olympus field development optimization challenge, in: ECMOR XVI-16th European Conference on the Mathematics of Oil Recovery, 2018, http://dx.doi.org/10.3997/2214-4609.201802246.

[16] J. Appleyard, I.M. Cheshire, Nested factorization, in: 7th SPE Symposium on Reservoir Simulation, San Francisco, USA, Society of Petroleum Engineers, 1983, http://dx.doi.org/10.2118/12264-MS.

[17] H.A. Van der Vorst, Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 13 (2) (1992) http://dx.doi.org/10.1137/0913035.

[18] Y. Saad, M. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (3) (1986) http://dx.doi.org/10.1137/0907058.

[19] J. Wallis, Incomplete Gaussian elimination as a preconditioning for generalized conjugate gradient acceleration, in: 7th SPE Symposium on Reservoir Simulation, San Francisco, USA, Society of Petroleum Engineers, 1983, http://dx.doi.org/10.2118/12265-MS.

[20] R. Scheichl, M. Roland, J. Wendebourg, Decoupling and block preconditioning for sedimentary basin simulations, Comput. Geosci. 7 (2003) 295–318.

[21] M. Blatt, A Parallel Algebraic Multigrid Method for Elliptic Problems with Highly Discontinuous Coefficients (Ph.D. thesis), Ruprecht–Karls–Universität Heidelberg, 2010.

[22] M. Blatt, P. Bastian, The iterative solver template library, in: B. Kågström, E. Elmroth, J. Dongarra, J. Waśniewski (Eds.), Applied Parallel Computing. State of the Art in Scientific Computing, in: Lecture Notes in Computer Science, vol. 4699, Springer, 2007, pp. 666–675.

[23] M. Blatt, P. Bastian, On the generic parallelisation of iterative solvers for the finite element method, Int. J. Comput. Sci. Engrg. 4 (1) (2008) 56–69, http://dx.doi.org/10.1504/IJCSE.2008.021112.
[24] S. Balay, et al., PETSc Users Manual, Tech. Rep. ANL-95/11 - Revision 3.11, Argonne National Laboratory, 2019, www.mcs.anl.gov/petsc.
[25] D. Demidov, AMGCL: an efficient, flexible, and extensible algebraic multigrid implementation, 2018, URL https://arxiv.org/abs/1811.05704.
[26] Y. Zhou, H.A. Tchelepi, B.T. Mallison, Automatic differentiation framework for compositional simulation on unstructured grids with multi-point discretization schemes, in: SPE Reservoir Simulation Symposium, Society of Petroleum Engineers, 2011, http://dx.doi.org/10.2118/141592-MS.
[27] A. Griewank, A. Walther, Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation, second ed., SIAM, Philadelphia, 2008.
[28] Sacado documentation website. URL https://docs.trilinos.org/dev/packages/sacado/doc/html/index.html.
[29] A. Walther, A. Griewank, Getting started with ADOL-C, in: U. Naumann, O. Schenk (Eds.), Combinatorial Scientific Computing, Chapman-Hall CRC Computational Science, 2012, pp. 181–202.
[30] R. Younis, K. Aziz, Parallel automatically differentiable data-types for next-generation simulator development. society of petroleum engineers, in: SPE Reservoir Simulation Symposium, Society of Petroleum Engineers, 2007, http://dx.doi.org/10.2118/106493-MS.
[31] AD-GPRS website. URL https://supri-b.stanford.edu/research-areas/ad-gprs.
[32] A. Lauser, A. Rasmussen, T. Sandve, H. Nilsen, Local forward-mode automatic differentiation for high performance parallel pilot-level reservoir simulation, in: ECMOR XVI-16th European Conference on the Mathematics of Oil Recovery, 2018, http://dx.doi.org/10.3997/2214-4609.201802153.
[33] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, R. Kornhuber, M. Ohlberger, O. Sander, A generic grid interface for parallel and adaptive scientific computing. Part II: Implementation and tests in DUNE, Computing 82 (2–3) (2008) 121–138, http://dx.doi.org/10.1007/s00607-008-0004-9.
[34] E.G. Boman, U.V. Catalyurek, C. Chevalier, K.D. Devine, The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring, Sci. Program. 20 (2) (2012).
[35] K.D. Devine, E.G. Boman, R.T. Heaphy, R.H. Bisseling, U.V. Catalyurek, Parallel hypergraph partitioning for scientific computing, in: Proceedings 20th IEEE International Parallel & Distributed Processing Symposium, IEEE, 2006, http://dx.doi.org/10.1109/IPDPS.2006.1639359.
[36] U. Meier Yang, On the use of relaxation parameters in hybrid smoothers, Numer. Linear Algebra Appl. 11 (2–3) (2004) 155–172.
[37] M. Todd, W. Longstaff, The development, testing, and application of a numerical simulator for predicting miscible flood performance, J. Pet. Technol. 24 (07) (1972) 874–882.
[38] C.A. Chase Jr., M.R. Todd, Numerical simulation of CO_2 flood performance (includes associated papers 13 950 and 13 964), SPE J. 24 (06) (1984) 597–605.
[39] S. Jakupsstovu, D. Zhou, J. Kamath, L. Durlofsky, E.H. Stenby, Upscaling of miscible displacement processes, in: Proceedings of the 6th Nordic Symposium on Petrophysics, 2001, pp. 15–16.
[40] A.S. Odeh, Comparison of solutions to a three-dimensional black-oil reservoir simulation problem (includes associated paper 9741), J. Pet. Technol. 33 (01) (1981) 13–25.
[41] D. Kenyon, Third SPE comparative solution project: gas cycling of retrograde condensate reservoirs, SPE J. 39 (08) (1987) 981–997, http://dx.doi.org/10.2118/12278-PA.
[42] J.E. Killough, C.A. Kossack, Fifth comparison solution project: Evaluation of miscible flood simulators, in: SPE Symposium on Reservoir Simulation, 1987, http://dx.doi.org/10.2118/16000-MS.
[43] J.E. Killough, Ninth SPE comparative solution project: a reexamination of black-oil simulation, in: SPE Reservoir Simulation Symposium, Society of Petroleum Engineers, 1995, http://dx.doi.org/10.2118/29110-MS.
[44] K. Bao, K.-A. Lie, O. Møyner, M. Liu, Fully implicit simulation of polymer flooding with MRST, Comput. Geosci. 21 (5) (2017) 1219–1244, http://dx.doi.org/10.1007/s10596-017-9624-5.
[45] T.H. Sandve, A. Rasmussen, A.B. Rustad, Open reservoir simulator for CO2 storage and CO2-EOR, in: 14th Greenhouse Gas Control Technologies Conference Melbourne, 2018, pp. 21–26.
[46] R. Lorentzen, X. Luo, T. Bhakta, R. Valestrand, History matching the full Norne field model using seismic and production data, SPE J. 24 (4) (2019) 1452–1467, http://dx.doi.org/10.2118/194205-PA.
[47] A. Kvashchuk, R. Klöfkorn, T.H. Sandve, Comparison of higher order schemes on complicated meshes and reservoirs, in: SPE Reservoir Simulation Conference, Society of Petroleum Engineers, 2019, http://dx.doi.org/10.2118/193839-MS.

Appendix B

Simulators for the Gigaton Storage Challenge. A Benchmark Study on the Regional Smeaheia Model

Tor Harald Sandve, Alf Birger Rustad, Andreas Thune, Bamshad Nazarian, Sarah Gasda, Atgairr Flø Rasmussen

Introduction

The gigaton-storage challenge is to accelerate scale-up of CO_2 storage to meet global emissions targets by mid-century. Continental margins are particularly well-suited, and initial estimates point to a realistic build-out of injection. Significant investments are underway to spur national and pan-national transport and storage infrastructure projects. Capitalizing on these investments depends on operators being able to access sufficient capacity in these regions for future CCS growth scenarios. Significant efforts in geological characterization of highly prospective offshore regions, e.g., North Sea, US Gulf Coast, and Australia, have resulted in regional atlases and other high-resolution datasets that are highly valuable for storage assessment. These systems are laterally extensive, geologically complex, and span several vertical strata and impose a particular set of requirements on simulators – robust, sufficiently resolved, and efficient at large-scale and long timescales – that goes beyond the scope of existing tools. To meet this challenge, both commercial and research codes have been under significant development recently. For the operators to fully utilize this on-going development the simulators need to be tested and compared on realistic models. Important measures to compare are accuracy, functionality, speed, and compatibility with existing workflows.

In this paper the OPM Flow simulator (Rasmussen et.al. 2021) is benchmarked against the compositional simulator Eclipse 300 (E300) on the regional Smeaheia model developed by Equinor. An example set-up was created for the purpose of demonstration only and does not reflect any actual injection conditions or storage potential of Smeaheia. The current model setup assumes no dissolution of CO_2 into the brine and uses a constant temperature gradient during the simulations. Before we present the benchmark results, we shortly present the regional model and the simulation approaches used by the OPM Flow simulator. For details on E300 simulator we refer to the Eclipse user and technical manual. The paper concludes with a discussion on the results and the need for further benchmarking including more simulators as well as more advanced physics.

CO_2 simulations using the CO2STORE option in OPM Flow

The open-source reservoir simulator OPM Flow has recently been extended with a CO_2 storage dedicated module. The module computes PVT properties as density and viscosity for the CO_2 brine mixture internally in the simulator and thus simplifies model setup considerably. The PVT properties are computed based on state-of-the art functional relations from the literature and depends on pressure, temperature, and salinity. Details on the different relations used by the simulator is presented in Sandve et.al. (2021). The OPM Flow simulator is a dedicated black-oil simulator and the PVT properties thus needs to be translated to the equivalent black-oil properties as formation volume factor and gas/oil ratios. This conversion happens automatically during the simulations. A benefit of this approach is the immediate accessibility of all the functionality available in the black-oil simulator. The approach differs from the commonly used practice of providing black-oil tables for the CO_2-brine mixture directly in one significant way. In standard black-oil simulation, tables are pre-computed based on an assumption of constant temperature and salinity. Temperature and salinity variations to the PVT properties can be modelled by adding temperature and/or salinity dependent multipliers, but this process is both cumbersome and assumes a multiplicative functional relation between the temperature and salinity effects and the PVT properties. In the CO_2 module in OPM Flow, the black-oil properties are computed dynamically during the simulations and the temperature and salinity dependence is therefore naturally included in the model. As a result, including effects of CO_2 dissolution, diffusion or dynamic temperature variations in the simulation model is trivial for the user perspective as no modification is needed in the input other than a flag of whether the feature should be included or not. More details on the setup can be found in Sandve et.al. (2021) and under the CO2STORE keyword in the OPM manual (Baxendale et. al. 2021).

The Smeaheia regional model

Equinor has developed a simulation model with 9-million active cells of the Smeaheia region that includes a 1000-m vertical stack of potential storage layers. The model includes complex fault

structures, numerical aquifers to model pressure dissipation to more distant aquifers and a set of pseudo-producers to model depletion from the Troll field. An example set-up was created for the purposes of demonstration only and does not reflect any actual injection conditions or storage potential of Smeaheia. Here, 15 million tons CO_2 is injected annually into a vertical well over a large injection interval spanning different multiple storage layers. Injection starts after an initial drawdown period of 25 years that replicate the depletion effect from the Troll field. The CO_2 injection period is 40 years and after that the simulations continues for 35 more years to model post-injection migration.

The model is setup as a E300 compositional case with water and a CO_2 component. CO_2 solubility in brine is not modelled in this study. A Peng Robinson EOS is used to compute the PVT properties of the CO_2 with Pedersen correlation for the viscosity. For details we refer to the Eclipse manual. The brine density and viscosity are given as tabulated values to E300 in this model setup. This differs from the way it is modeled in OPM Flow where density and viscosity are functions of the cell temperature and salinity. As the temperature is set to vary linearly with depth in the model as shown in first plot in Figure 1, this introduces some discrepancy between the brine density and viscosity used by the simulators as shown the second plot in Figure 1. Slight variations are also expected for the CO_2 properties as OPM Flow uses the Span-Wagner model while the E300 uses Peng Robinson EOS. Note that the differences in the PVT approach could have been minimized by using the CO2STORE option in E300. As the CO2STORE option in E300 also include mutual solubility of CO_2 and brine, the DISGAS and VAPOIL option must be enabled in OPM Flow for comparable results. Accurate and efficient inclusion of solubility in field scale simulations are important as discussed for instance by Sandve et.al. (2021) but is beyond the limited scope of this paper. For further discussions on the effect of using different EOS models we refer to Aavatsmark et. al. (2016).

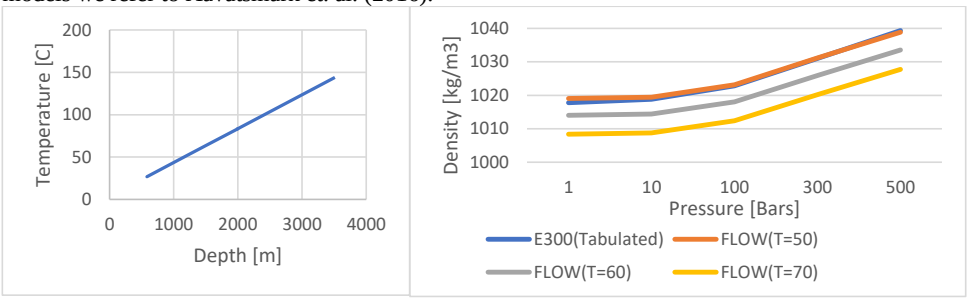


***Figure 1**. Left: Temperature vs depth for the Regional Smeaheia model. Right: Brine density computed for different pressure and temperatures with salinity 0.85 using the PVT model in OPM Flow and the tabulated value used in the E300 simulations. Note that the tabulated values are almost on top of the values used by Flow for T=50.*

The benchmark results

We now want to compare the simulators in terms of functionality, accuracy, speed, and compatibility with existing workflow. For functionality both simulators can set up the complex geology with faults and layers in addition to the necessary boundary conditions e.g., group-controlled wells for the Troll depletion and analytic aquifers for pressure support for adjacent aquifers. Only minor modification related to removing compositional specific input and setup of the CO2STORE option was needed in the original E300 input data to be able to run OPM Flow with the same data set. This simplified the comparison considerably. Since the two simulators uses different PVT options, a direct comparison of the accuracy is not straight forward as already discussed in the previous section. Nevertheless, we include some comparisons for relevant metrics as the CO_2 plume at year 2075 seen from top in Figure 2 and from a vertical cross section cut through the well cells in the X direction in Figure 3. A close look shows differences as expected but the overall results seem to agree reasonably well. The pore-volume weighted average pressure is shown in Figure 4 for the simulations. Here also results from OPM Flow simulations with fresh water (no salinity) and constant temperature of 51.5C (the temperature that gives best match to the tabulated brine properties used in the E300 model) is included to highlight the impact

of the variations in the PVT models. Note for instance that the results with fixed temperature in OPM Flow gives nearly the same average pressure values as the E300 simulations up-to 2050, while the increasing trend in the average pressure after 2050 is more align for the simulations with varying temperature.

Simulating multi-million models are time-consuming, and therefore, fast, and parallel scalability is mandatory for the models to be usable in practice. Figure 4 shows a comparison of results up to 16 MPI processes between the simulators on an Equinor internal workstation and as well results from 16 to 512 MPI processes on 1 to 4 dual AMD EPYC 7763 nodes located on the Simula eX^3 HPC infrastructure. First note that OPM Flow is significantly faster than E300 for the comparison up to 16 processes. This is especially evident in the later part of the simulations where the simulation time with E300 increases significantly. This is the period of the simulations where all wells are closed, and the flow is dominated by buoyancy driven CO_2 migration. Both simulators show similar parallel scalability for the internal Equinor setup. Due to restriction on licenses for the E300 simulator, only OPM Flow was run on the eX^3 cluster. The simulation time on eX^3 shows that the OPM Flow simulator scales well up to 512 processes. With 512 processes the simulation time until year 2200 is less than an hour. Note that the initial setup time is almost 1/3 of the total simulation time which points to further potential reduction in simulation time.

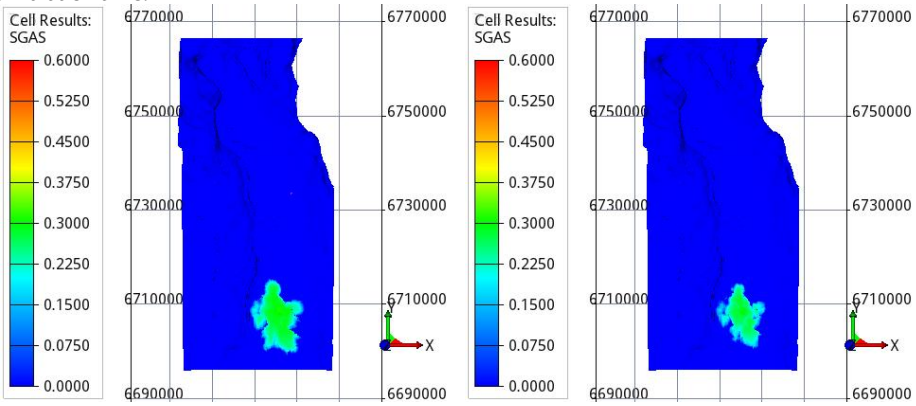


Figure 2. *CO_2 plume seen from top at year 2075. Left: OPM Flow Right: E300.*

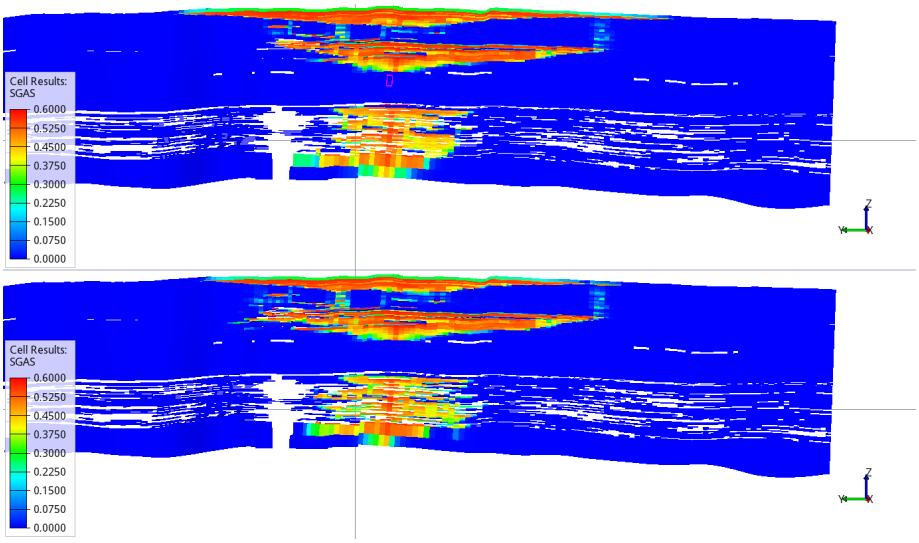


Figure 3. *CO_2 plume from seen from a cross-section at year 2075. Top: OPM Flow Bottom: E300*

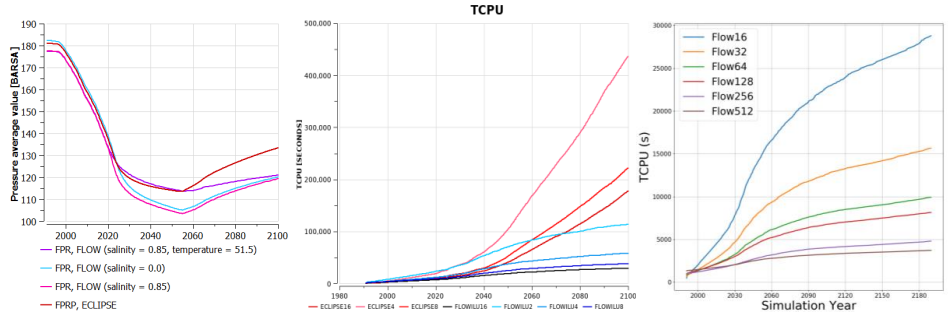


Figure 4. *Left: Pore-volume weighted average pressure. Mid and right: Simulation time for E300 vs OPM Flow up-to 16 MPI processes on the Equinor cluster and OPM Flow on eX^3 HPC cluster from 16-512 MPI processes, respectively.*

Conclusions

Initial benchmarking of the OPM Flow simulator and the Eclipse E300 compositional simulator on the Smeaheia regional model concludes that both simulators can handle the needed complexity of the model and are equally compatible to existing industry workflows. Both simulators show good scalability up to 16 MPI processes on the Equinor workstation, but the OPM Flow simulator was significantly faster than E300 for this case. This is in particular evident for the late stage of the simulation where wells are shut, and the flow dominated by gravity driven CO_2 migration. The OPM Flow simulator shows good scaling up-to 512 MPI processes on the eX^3 cluster and can simulate the multi-million model in less than an hour. The results differ between the simulators. This is expected as they use different PVTs: The E300 uses the Peng-Robinson EOS model for CO_2 while OPM Flow use Span-Wagner. E300 uses tabulated values for the brine properties assuming constant temperature while OPM Flow computes the properties directly and thus considers the initial temperature gradient in the model. To further build trust in the simulation results further benchmarking is needed. Most significantly the model needs to include the effect of dissolving CO_2 in brine and local temperature effects around the injection wells. Also, more simulators should be benchmarked and compared to fully being able to exploit the full potential of the recent advancements in simulation capabilities targeting the gigaton CO_2 storage challenge.

Acknowledgements

The authors thanks Equinor for giving access to the regional Smeaheia model and Gassnova and Equinor for funding through the HPC-G CLIMIT-Demo project (620073).

References

Aavatsmark, I., Kometa, B. K., Gasda, S. E., Sandve, T. H., & Nilsen, H. M. (2016). A generalized cubic equation of state with application to pure CO 2 injection in aquifers. *Computational Geosciences*, *20*(3), 623-635.

Baxendale, D., Rasmussen, A. F., Rustad, A. B., Skille, T., & Sandve, T. H. (2021). OPM Flow documentation manual.

Rasmussen, A. F., Sandve, T. H., Bao, K., Lauser, A., Hove, J., Skaflestad, B., ... & Thune, A. (2021). The open porous media flow reservoir simulator. *Computers & Mathematics with Applications*, *81*, 159-185.

Sandve, T. H., Gasda, S. E., Rasmussen, A., & Rustad, A. B. (2021). Convective Dissolution in Field Scale Co2 Storage Simulations Using the OPM Flow Simulator. In *TCCS–11. CO2 Capture, Transport and Storage. Trondheim 22nd–23rd June 2021 Short Papers from the 11th International Trondheim CCS Conference*. SINTEF Academic Press.