A GPU-based integrated simulation framework for modelling of complex subsurface applications

Khait, Mark; Voskov, Denis

DOI
10.2118/204000-MS

Publication date
2021

Document Version
Final published version

Published in
Society of Petroleum Engineers - SPE Reservoir Simulation Conference 2021, RSC 2021

SPE-204000-MS

A GPU-Based Integrated Simulation Framework for Modelling of Complex Subsurface Applications

Mark Khait, Delft University of Technology; Denis Voskov, Delft University of Technology, Stanford University

This paper was prepared for presentation at the SPE Reservoir Simulation Conference, available on-demand, 26 October 2021 – 25 January 2022. The official proceedings were published online 19 October 2021.

Abstract

Alternative to CPU computing architectures, such as GPU, continue to evolve increasing the gap in peak memory bandwidth achievable on a conventional workstation or laptop. Such architectures are attractive for reservoir simulation, which performance is generally bounded by system memory bandwidth. However, to harvest the benefit of a new architecture, the source code has to be inevitably rewritten, sometimes almost completely. One of the biggest challenges here is to refactor the Jacobian assembly which typically involves large volumes of code and complex data processing. We demonstrate an effective and general way to simplify the linearization stage extracting complex physics-related computations from the main simulation loop and leaving only an algebraic multi-linear interpolation kernel instead. In this work, we provide the detailed description of simulation performance benefits from execution of the entire nonlinear loop on the GPU platform. We evaluate the computational performance of Delft Advanced Research Terra Simulator (DARTS) for various subsurface applications of practical interest on both CPU and GPU platforms, comparing particular workflow phases including Jacobian assembly and linear system solution with both stages of the Constraint Pressure Residual preconditioner.

Introduction

Numerical simulations are essential for the modern development of subsurface reservoirs (Aziz and Settari, 1979; Dake, 1983; Peaceman, 2000). They are widely used for the evaluation of oil recovery efficiency, performance analysis, and various optimization problems. Due to the complexity of the underlying physical processes and considerable uncertainties in the geological representation of reservoirs, there is a persistent demand for more accurate models.

Fully implicit methods (FIM) are conventionally used in reservoir simulation because of their unconditional stability (Aziz and Settari, 1979). On the other hand, after discretization is applied to governing Partial Differential Equations (PDE) of a problem, the resulting nonlinear system represents different tightly coupled physical processes, which is difficult to solve. Usually, a Newton-based iterative method is applied, which demands an assembly of the Jacobian and the residual for the combined system of equations (i.e., linearization) at every iteration forming a linear system of an equal size (often illconditioned). Precisely the solution of such systems takes most of the simulation time in most practical applications.

Several conventional linearization approaches exist, though neither of them is robust, flexible, and computationally efficient all at once. Numerical derivatives provide flexibility in the nonlinear formulation (see Xu et al., 2011, for example), but a simulation based on numerical derivatives may lack robustness and performance (Vanden and Orkwis, 1996). Straightforward hand-differentiation is the state-of-the-art strategy in modern commercial simulators (Cao et al., 2009; Schlumberger, 2011). However, this approach requires an introduction of a complicated framework for storing and evaluating derivatives for each physical property, which in turn reduces the flexibility of a simulator to incorporate new physical models and increases the probability for potential errors. The development of the Automatic Differentiation (AD) technique allows preserving both flexibility and robustness in derivative computations. In reservoir simulation, the Automatically Differentiable Expression Templates Library (ADETL) was introduced by Younis (2011). Being attractive from the perspective of flexibility, the AD technique by design inherits computational overhead, which affects the performance of reservoir simulation (Khait and Voskov, 2017a).

Another linearization approach called Operator-Based Linearization (OBL) was proposed by Voskov (2017). It could be seen as an extension of the idea to abstract the representation of properties from the governing equations, suggested in Zaydullin et al. (2013) and Haugen and Beckner (2015). In the OBL approach, the parameterization is performed based on the conventional molar variables. All properties involved in the governing equations are lumped in a few operators, which are parameterized in the physical space of the simulation problem either in advance or adaptively during the simulation process. The control on the size of parameterization hyper-rectangle helps to preserve the balance between the accuracy of the approximation and the performance of nonlinear solver (Khait and Voskov, 2017b). Note, that the OBL approach does not require the reduction in the number of unknowns, and only employs the fact that physical description (i.e., fluid properties) is approximated using piecewise linear interpolation.

Delft Advanced Research Terra Simulator (DARTS) was introduced and described in Khait (2019). It exploits the OBL approach to decouple the computations of physical properties from the main simulator core. Jacobian assembly in DARTS is therefore simplified and generalized increasing its portability to alternative computational architectures, such as GPU. In this work, we demonstrate the viability of GPU-based compositional simulation in DARTS with up to 10 components. We evaluate the computational performance of DARTS for various subsurface applications relevant to energy transition on both CPU and GPU platforms. We provide a detailed performance comparison of particular workflow pieces composing Jacobian assembly and linear system solution, including both stages of CPR preconditioner.

Method

Here, we briefly describe various ingredients of the Delft Advanced Reservoir Terra Simulation (DARTS, 2019) framework.

Governing equations

First, we describe the conventional nonlinear formulation for a general purpose thermal compositional model. Mass and energy transport for a system with n_p phases and n_c components is considered. For this model, the n_c component mass and energy conservation equations can be written as

$$\frac{\partial m_c(\boldsymbol{\xi}, \boldsymbol{\omega})}{\partial t} + \text{div}\ f_c(\boldsymbol{\xi}, \boldsymbol{\omega}) + q_c(\boldsymbol{\xi}, \boldsymbol{\omega}, u) = 0, \quad c = 1, \ \dots, n_c + 1. \tag{1}$$

Here, $\boldsymbol{\xi}$ are space-dependent parameters, $\boldsymbol{\omega}$ are state-dependent parameters and $\mathbf{u}$ are control variables and

$$m_c(\boldsymbol{\xi}, \boldsymbol{\omega}) = \phi \sum_{j=1}^{np} x_{cj} \rho_p s_j, \qquad c = 1, \ \dots, n_c, \tag{2}$$

and

$$m_c(\xi, \boldsymbol{\omega}) = \phi \sum_{j=1}^{np} \rho_p s_j u_j + (1-\phi) u_r, \; = c = n_c + 1, \tag{3}$$

where t is time, ϕ is effective rock porosity, x_{cj} is component c concentration in phase j, ρ_j denotes phase j molar density, s_j is saturation of phase j and u_j is phase internal energy. Similarly,

$$f_c(\xi, \boldsymbol{\omega}) = \sum_{j=1}^{np} x_{cj} \rho_j \vec{v}_j + s_j \rho_j \mathrm{J}_{cj}, \qquad c = 1, \ldots, n_c, \tag{4}$$

and

$$f_c(\xi, \boldsymbol{\omega}) = \sum_{j=1}^{np} h_j \rho_j \vec{v}_j - \left(\phi \sum_{j=1}^{np} \kappa_j s_j + (1-\phi) \kappa_r \right) \nabla T, \;\; c = n_c + 1, \tag{5}$$

where h_j is the phase enthalpy, κ_j is phase thermal conduction, $\vec{v}_j$ is Darcy velocity and $\mathbf{J}_{cj}$ is Fick's diffusion flux. Here, $\mathbf{K}$ is the effective permeability tensor, k_{rj} is relative permeability, μ_j is phase viscosity, p_j is phase pressure, γ_j is hydrostatic gradient, and D is depth.

Operator form of governing equations

According to the Operator Based Linearization (OBL) method proposed in Voskov (2017), all terms in the Equation 1 are written as functions of a physical state $\boldsymbol{\omega}$ and a spatial coordinate ξ. The physical state represents a unification of all state variables (i.e., nonlinear unknowns: pressure, temperature/enthalpy, saturations/compo- sitions, etc.) of a single control volume. In the overall molar formulation, the nonlinear unknowns are pressure p, fluid enthalpy h and overall composition z_c, therefore the physical state $\boldsymbol{\omega}$ is completely defined by these variables. The spatial coordinate ξ defines the location of a given control volume which reflects the distribution of heterogeneous rock properties (e.g., porosity, thermal conduction) and elements of space discretization (e.g., transmissibility). Besides, well control variables $\mathbf{u}$ are introduced to represent various well management strategies.

Equation 1 is discretized in space using finite-volume two-point flux approximation and in time using backward Euler approximation. The applied Fully Implicit Method (FIM) yield that the convective flux term depends on the values of nonlinear unknowns at the current time step. Next, we rewrite Equation 1 neglecting for simplicity buoyancy and capillary forces (see more general treatment in Khait and Voskov, 2018a; Lyu et al., 2021a), and represent each term as a product space-dependent properties and of state-dependent operators (Khait and Voskov, 2018b). The resulting conservation equations read

$$\begin{aligned} \frac{V\phi_0}{\Delta t}(\alpha_c(\boldsymbol{\omega}) - \alpha_c(\boldsymbol{\omega}^n)) \quad & - \sum_l \left[\beta_c^l(\boldsymbol{\omega}) \Gamma^l(\xi) \Delta \Phi_j - \gamma_{cj}^l(\boldsymbol{\omega}) \Gamma_d^l(\xi) \Delta x_{cj} \right] \\ & - \beta_c^w(\boldsymbol{\omega}, u) \Gamma^w(\xi) \Delta p^w = 0, \qquad c = 1, \ldots, n_c, \end{aligned} \tag{6}$$

and

$$\begin{aligned} \frac{V}{\Delta t} \left[\phi_0(\alpha_e(\boldsymbol{\omega}) - \alpha_e(\boldsymbol{\omega}^n)) + (1-\phi_0) u_r(\xi)(\alpha_r(\boldsymbol{\omega}) - \alpha_r(\boldsymbol{\omega}^n)) \right] - \sum_l \beta_h^l(\boldsymbol{\omega}) \Gamma^l(\xi) \Delta \Phi_j \\ - \sum_l \Gamma_r^l(\xi) [\phi_0 \beta_e^l(\boldsymbol{\omega}) + (1-\phi_0) \kappa_r \alpha_r(\boldsymbol{\omega})] \Delta T^l - \beta_h^w(\boldsymbol{\omega}, u) \Gamma^w(\xi) \Delta p^w = 0, \end{aligned} \tag{7}$$

where V is the volume of mesh grid block, ϕ_0 is rock porosity at the reference pressure, Γ^l, Γ_d^l and Γ_r^l is the space-dependent part of convective, diffusive and conductive transmissibility respectively, Φ_j is phase potential. A state-dependent operator is defined as a function of the physical state only. Therefore, it is independent of spatial position and represents physical properties of fluids and rock

$$\alpha_c(\boldsymbol{\omega}) = (1 + c_r(p - p_0))\sum_{j=1}^{np} x_{cj}\rho_j s_j, \qquad c = 1, \ldots, n_c, \tag{8}$$

$$\alpha_e(\boldsymbol{\omega}) = (1 + c_r(p - p_0))\sum_{j=1}^{np} u_j\rho_j s_j, \tag{9}$$

$$\alpha_r(\boldsymbol{\omega}) = \frac{1}{1 + cr(p - p_0)}, \tag{10}$$

$$\beta_c(\boldsymbol{\omega}) = \sum_{j=1}^{np} x_{cj}\rho_j \frac{k_{rj}}{\mu_j}, \quad c = 1, \ldots, n_c, \tag{11}$$

$$\beta_h(\boldsymbol{\omega}) = \sum_{j=1}^{np} h_j\rho_j \frac{k_{rj}}{\mu_j}, \tag{12}$$

$$\beta_e(\boldsymbol{\omega}) = (1 + c_r(p - p_0))\sum_{j=1}^{np} s_j\kappa_j, \tag{13}$$

$$\gamma_c(\omega) = (1 + c_r(p - p_0))\rho_j s_j. \tag{14}$$

In the equations above, ϕ_0 - rock porosity at the reference pressure, c_r is rock compressibility, p_0 - reference pressure, while $\boldsymbol{\omega}$ and $\boldsymbol{\omega}_n$ represent the nonlinear state (unknowns of a single reservoir block) at the current and previous time step respectively.

The physical meaning of mass accumulation operator α_c is the molar mass of component c per unit pore volume of uncompressed rock under physical state $\boldsymbol{\omega}$. The physical meaning of the mass flux operator for component c is the total mobile molar mass of that component in all phases of the mixture under physical state $\boldsymbol{\omega}$ per unit time, pressure gradient, and constant geometrical part of transmissibility. This representation allows us to identify and distinguish the physical state-dependent operators in the governing conservation equations 6 and 7.

DARTS Structure

From the perspective of the simulation nonlinear loop, the operator interpolation replaces properties calculations in equations 8–13 during the Jacobian assembly step following the idea of operator-based linearization (Voskov, 2017). Besides, it also ‘shadows’ physical phenomena behind the operators, leaving out only the values of supporting points, which are rarely computed but utilized all the time during interpolation for Jacobian evaluation. This allows to detach fluid and rock properties calculations (now only performed during operator evaluation at supporting points) from the main nonlinear loop, as well as to relax the performance requirements for such calculations.

The Jacobian assembly depends on the choice of the nonlinear variables and the governing physical mechanisms which are taken into account. The former determine the dimensionality of parameter space, while the latter define the operators required for the assembly. Once the choice is made, the Jacobian assembly becomes simply a combination of interpolated operator values and their partial derivatives with spatial properties, encapsulated in a simulation *engine*. It is connected with an *interpolator* which is responsible for computing interpolated operator values and derivatives. This connection represents the major data workflow occurring during a simulation. Finally, the interpolator is connected to a specific set of properties (i.e., *operator set*) which are used for the simulation. Operator sets must be chosen in agreement with the selected engine. They are only invoked when a new supporting point is needed to perform the interpolation.

The structure of DARTS is summarized in Figure 1. On the left, four simulation multiphase multi-component engines are shown:

- *engine _pz* – mass flow and transport, $\boldsymbol{\omega} = \{p, z1, \ldots, zn_c - 1\}$;
- *engine _pz_gc* – mass flow and transport with gravity and capillarity, $\boldsymbol{\omega} = \{p, z1, \ldots, zn_c - 1\}$;
- *engine_pz_gcd* – mass flow and transport with gravity, capillarity and diffusion, $\boldsymbol{\omega} = \{p, z1, \ldots, zn_c - 1\}$;
- *engine phz g* – mass and energy flow and transport with gravity, $\boldsymbol{\omega} = \{p, h, z1, \ldots, zn_c - 1\}$.

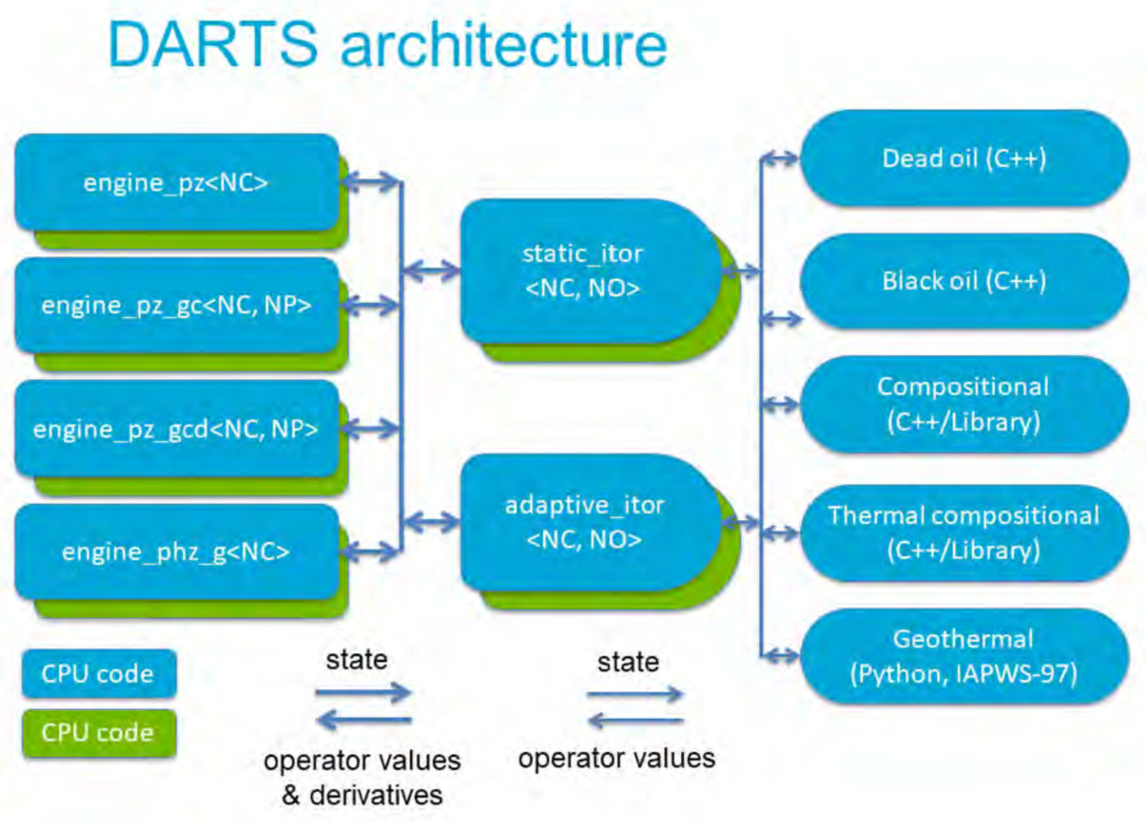


Figure 1—Delft Advanced Research Terra Simulator (DARTS) modular structure

All engines are written in a general manner for *nc* components (and n_p phases for *engine _pz_gc* and *engine_pz_gcd*). Notion < *NC* > here indicates that the variable represents integer template parameter of the corresponding class, known at compile time. This approach allows to maximize various compiler optimizations (e.g., loop unrolling, vectorization).

Next, two interpolators are available (Figure 1, middle):

- *static_itor* – pre-computes all supporting points in advance, and can be useful for coarse physical representation and low-dimensional parameter space;
- *adaptive_itor* – adaptively computes supporting points along with the simulation (see Khait and Voskov, 2018a, for details).

Both interpolators are written in a general way for n_d degrees of freedom and n_0 operators. All operator values are stored together to benefit from faster search and interpolation. Moreover, operator values computed during simulation can be stored into a file and loaded before subsequent simulation, which can be extremely beneficial in case of running multiple models with the same physical properties common for inverse modelling or optimization.

Finally, several operator sets are present (see Figure 1, right):

- Dead-oil – water and oil components, water and oil phases, $\boldsymbol{\omega} = \{p, z_w\}$;
- Black-oil – water, oil, and gas components, water, oil, and gas phases, $\boldsymbol{\omega} = \{p, z_g, z_o\}$;
- Compositional – n_c components, liquid and vapor phases, $\boldsymbol{\omega} = \{p, z_1, \ldots, zn_c - 1\}$;
- Thermal-compositional – n_c components, liquid and vapor phases, $\boldsymbol{\omega} = \{p, T, z_1, \ldots, zn_c - 1\}$;
- Geothermal – water component, liquid and vapor phases, $\boldsymbol{\omega} = \{p, h\}$.

GPU implementation has been developed for all the engines and interpolators described above. Naturally, all of the GPU engines utilize GPU-based linear solver, which still occupies the overwhelming majority of total simulation time. Therefore, dead-oil, black-oil, geothermal and even compositional models can be modeled in DARTS entirely on GPU.

Compared to our previous work on this topic described in (Khait and Voskov, 2017a), several major improvements have been made:

- GPU implementation has been embedded into the DARTS framework;
- GPU implementation has been added to several engines;
- Jacobian assembly together with adaptive and static interpolation GPU kernels have been generalized for n_c components, n_d degrees of freedom and n_o operators;
- Linear solver on GPU has been written using a two-stage CPR preconditioning technique.

Jacobian Assembly on GPU

DARTS has the same initialization scheme for both GPU and CPU versions, which are developed as interchangeable parts. The GPU version first loads the required initial data to GPU memory and then performs all major computations on the device. To initialize the GPU version of the static interpolator, all supporting points are first computed on CPU and then copied to device memory. Alternatively, they can be loaded from a file if one was created during a previous simulation and the fluid properties and physical space parameterization settings have not been changed since then. In particular, for Jacobian assembly, this data includes a connection list, pore volume and initial reservoir state arrays. The Jacobian structure is assumed to be fixed during simulation, so it is also initialized once and copied to the device memory before the run.

Interpolation of operator values and derivatives is performed as a preparatory step before Jacobian assembly. The interpolation kernel is implemented on a thread-per-cell basis, such that every GPU thread is responsible for computation of all operators and their derivatives for a given state $\boldsymbol{\omega}$. The data layout is analogous to the one used for CPU interpolator version, where values corresponding to a given cell are grouped, so coalesced memory access does not take place. However, global memory accesses are minimized as the interpolation uses a workspace array placed in register memory (unless register spill occurs, which is possible for high-dimensional parameter space and a large number of operators).

The GPU version of DARTS includes both static and adaptive interpolators. The static interpolator is much easier for parallel execution: all operator values are generated in advance, during the initialization stage, and then transferred to the device before simulation starts. Therefore, interpolation becomes an embarrassingly parallel procedure in this case. On the other hand, for an adaptive interpolator, there is a necessity to synchronize threads when new operator values are requested for the interpolation. The implementation of adaptive interpolator for the GPU version is therefore more complicated, as the data exchange between GPU and CPU, required for the generation of new operator values, needs to be overlapped with interpolation computations. Within the scope of this work, we used the static interpolator for both CPU and GPU versions of DARTS.

The Jacobian assembly, as well as all components of GPU-based linear solver in DARTS (except AMG), is based on classical Block CSR matrix format. It was chosen to preserve compatibility both with DARTS CPU code base and available linear solver implementations on GPU. Previous investigations showed that this format is not the best in terms of performance (Bell and Garland, 2009; Abdelfattah et al., 2015). However, our tests performed using more recent hardware/software identified that the situation might have changed.

The well control part of Jacobian is first formed on a host system and then sent to a device after the assembly for the reservoir part is complete. Usually, the size of the well control part is negligible compared to that of the full system, therefore the overhead is small even with synchronous memory operations. It is, however, possible to eliminate the overhead using asynchronous memory routines and CUDA streams or even recently introduced CUDA graphs. Those instruments allow overlapping of kernel execution and memory transfer. In that scenario, while the reservoir part is assembling on GPU during corresponding kernel invocation, the well part is computed on CPU and transferred to device memory. While the size of the well control part is small, its computations and transfer can be entirely hidden behind the assembly of the reservoir part.

The basic kernel for Jacobian assembly is also implemented on a thread-per-cell basis, and therefore global memory accesses are not coalesced here similarly to the interpolation kernel. To minimize those, the diagonal entry of Jacobian, as well as corresponding right-hand side block, are accumulated in register memory. Once the matrix row is completely processed, final values are written to corresponding global memory arrays. As opposed to diagonal entries of Jacobian, off-diagonal ones depend only on a single connection and their values are written directly to global memory.

The Jacobian assembly kernel was then improved. Each block matrix row is now processed with a group of n_d*n_d threads ($n_d = n_c$ for isothermal formulations and $n_d = n_c + 1$ for thermal ones), so that each GPU thread is pinned to a specific location of a matrix block for a specific matrix block row. The amount of required registers for such kernel does not depend on the matrix block size, while significant amount of memory accesses becomes coalesced, especially for larger block sizes. However, threads within a warp can process different matrix rows and therefore access memory in a non-contiguous fashion, especially for smaller block sizes. Similar to the basic kernel, the workload balance between different thread groups will only be close to ideal if the number of non-zeros per row is relatively stable (i.e. the majority of grid blocks have the same number of neighbouring blocks).

Linear Solver on GPU

Khait and Voskov (2017a) used a simple configuration of GPU linear solver based on a Krylov subspace iteration method with ILU(0) preconditioner. It has limited applicability to highly heterogeneous reservoir simulation problems requiring many iterations for convergence. Significantly more robust and efficient preconditioning scheme is based on the CPR technique. Wallis (1983); Wallis et al. (1985) designed it for efficient treatment of linear systems with mixed elliptic-hyperbolic unknowns. Such systems arise, in particular, from FIM discretization scheme for reservoir simulation problem. They are comprised of a near-elliptic pressure equation, a near-hyperbolic composition (saturation) equation, while the temperature equation can be either type depending on whether the process is diffusion-dominated (thermal conduction in energy equation) or convection-dominated. The CPR strategy has proved to be very robust and efficient even for highly heterogeneous reservoirs with strong coupling between elliptic and hyperbolic parts of the linear system. This results in stable convergence within a limited number of linear solver iterations even when simulation time steps are very large.

The linear system in DARTS is solved entirely on GPU using the Flexible Generalized Minimum Residual (FGMRES) iterative method (Saad, 1993) with CPR-based preconditioner. Figure 2 illustrates the normal workflow of linear solution strategy. All matrix operations are performed in native row-major BCSR format. During each nonlinear iteration, the Jacobian assembly is performed first. Then, Jacobian is passed over to the linear solver, where during setup phase additional matrices are computed. Finally, iterative linear solution takes place, where both Jacobian and additional matrices are used.

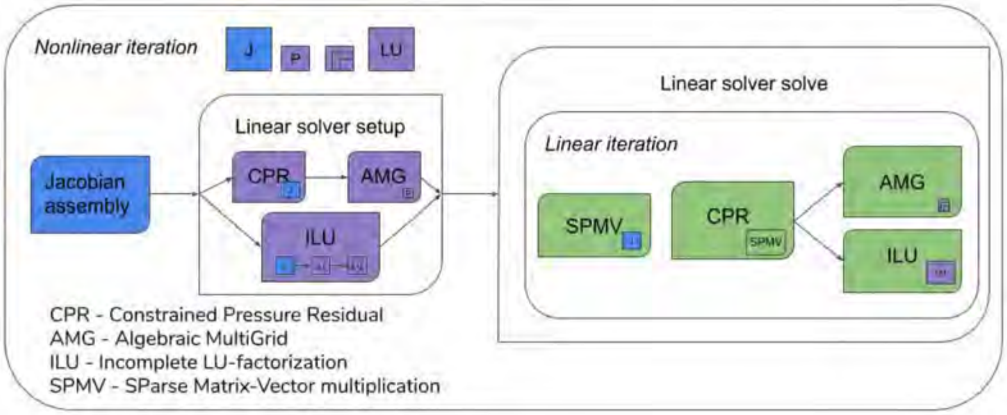

Nonlinear iteration
J
P
LU
Jacobian assembly
Linear solver setup
CPR
AMG
ILU
Linear solver solve
Linear iteration
SPMV
CPR
SPMV
AMG
ILU
CPR - Constrained Pressure Residual
AMG - Algebraic MultiGrid
ILU - Incomplete LU-factorization
SPMV - SParse Matrix-Vector multiplication

Figure 2—Linear solver workflow in DARTS

The setup phase is performed independently for the two stages of CPR-based preconditioner. At the first stage, the pressure system is decoupled from the full system using a True-IMPES reduction approach directly from the BCSR storage. It is then passed over to AMG solver, where an hierarchy of operators at different coarsening level is algebraically built. At the second stage, the values of Jacobian matrix are copied to a separate storage where block ILU(0) factorization is performed in-place.

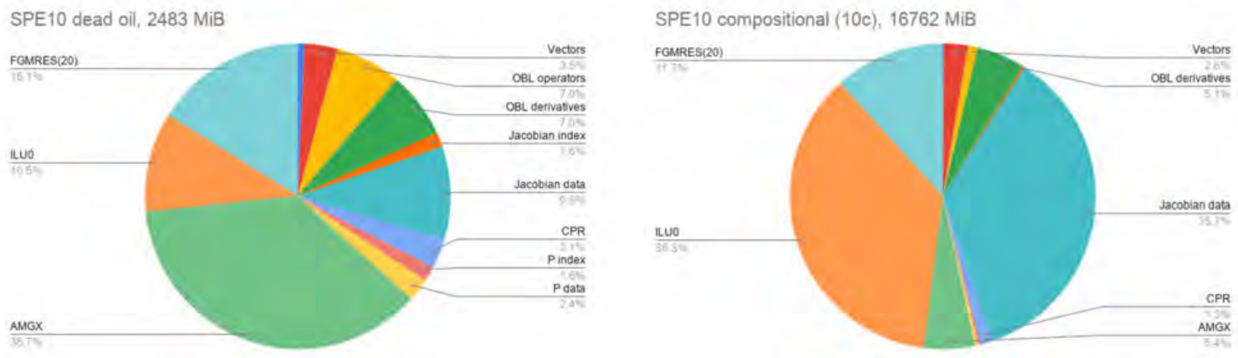
The solve phase consists of several iterations which are repeated until the desired convergence is reached. Each iteration of FGMRES involves an invocation of CPR preconditioning operator, where two-stage solution strategy is used. First, a single V-cycle of the AMG solver is performed to obtain an approximation of the pressure solution using the hierarchy of operators obtained during the setup phase. The approximation is then substituted into the full system to perform forward and backward substitution using ILU factorization. Both CPR and iterative solver require Sparse Matrix dense Vector multiplication (SpMV) operation performed on the Jacobian matrix.

Implementation details and memory consumption. GPU implementation of FGMRES method is straightforward. All vectors with the linear system size, as well as the matrix itself, reside on GPU, while the plane rotation procedure is kept on CPU. Vector and matrix routines are taken from the cuBLAS (dot, scale, axpy) and the cuSPARSE (bsrmv) libraries.

CPR preconditioner is implemented in DARTS analogously to the true-IMPES reduction in AD-GPRS simulation framework described by Zhou (2012). Decoupling is performed by a single kernel which fills out the values of the pressure matrix. Its structure is equivalent to the Jacobian matrix with the only difference that the former is pointwise. Each GPU thread is again assigned to a single matrix row. For the solution phase, we developed three simple kernels (for right-hand side reduction, solution prolongation, and stage solutions summation) and employed the matrix linear combination routine from cuSPARSE (bsrmv). To ensure efficient multiprocessor occupancy, launch grid dimensions for all kernels are computed via *cudaOccupancyMaxPotentialBlockSize* routine.

The first stage of CPR preconditioner is based on the open-source library AMGX (Naumov et al., 2015) (specifically, the commit ID 0e32e35 was used). The second stage of preconditioning relies on the bsrilu02 routine of the cuSPARSE library.

Along with the basic *engine pz*, the capabilities of GPU linear solver were extended to support up to 10 degrees of freedom per control volume (i.e., the maximum block size of matrix block is 10). The comparison of memory consumption profile on GPU for different number of components is demonstrated on Figure 3. It is evident that for 2 component model AMGX is responsible for the biggest chunk of allocated memory on GPU. At the same time, ILU(0) and Jacobian data storages come to the foreground for 10 component case, since the size of pressure matrix stays the same.



SPE10 (dead oil), 2483 MiB

SPE10 (10 components), 16762 MiB

Figure 3—GPU memory consumption profile for SPE 10 model with different physical kernels

Results

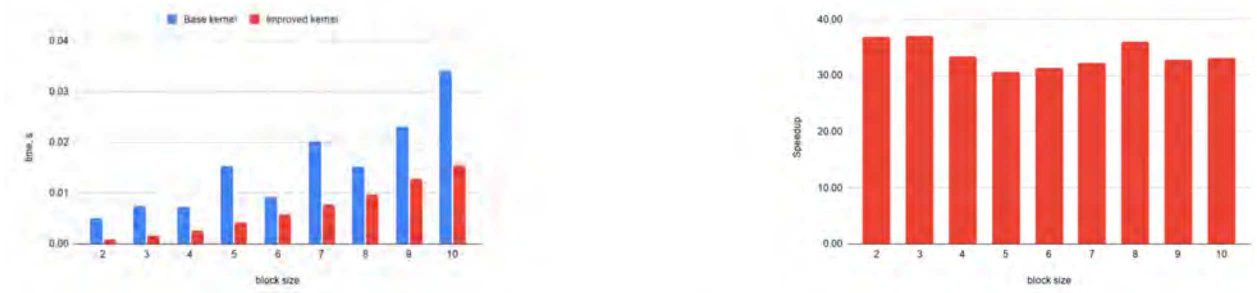
We used two computer systems to perform simulations: a workstation with a top-tier GPU and dual-processor cluster node. The workstation is based on Intel Core i7-8086K CPU clocked at 4.2 GHz with 32 GB DDR4 memory clocked at 2.4 GHz with a peak memory bandwidth of 41.6 GB/s and NVidia GeForce RTX2080 Ti graphics card with 11GB GDDR6 memory onboard with a peak memory bandwidth of 616 GB/s. For such generally memory-bound problems as FIM reservoir simulation, in case of parallel execution, peak memory bandwidth is a key performance indicator, not the clock speed. Therefore, the gaming card is expected to perform on the level of NVidia Tesla P100 GPU accelerator having only 15% smaller peak memory bandwidth and newer microarchitecture. At the same time, the gap between peak memory bandwidth for CPU and GPU hints the difference in the performance capacity of these platforms for reservoir simulation. The software configuration of the workstation included Ubuntu 18.04.3 operating system, GCC 7.5.0 compiler and CUDA Toolkit 10.2. The cluster node included two Intel Xeon E5-2640 v4 CPU processors clocked at 2.4 GHz with 256 Gb memory with a peak memory bandwidth of 136.6 GB/s for the two-socket system. The software configuration of the cluster node included CentOS Linux 7 operating system and GCC 4.8.5 compiler. As opposed to the workstation, the cluster node is a system with non-uniform memory access (NUMA). For such systems, it is important to prevent OpenMP threads from moving between processors to achieve higher memory bandwidth (provided that the implementation is also NUMA-aware). We performed our tests with OMP PLACES=cores and OMP PROC BIND=spread environment variables achieving noticeable improvement in multithread performance in case of the cluster node.

GPU kernels benchmark

We investigate the performance of basic and improved Jacobian assembly GPU kernels written for BCSR matrix format for engine pz and different block sizes. The SPE10 model (see detailed description in section) with the first 40 layers was taken to fit the GPU size (11 Gb) even for 10 component case. We used an artificial compositional physical model with simple operators in the following form:

$$\alpha_c(\boldsymbol{\omega}) = \beta_c(\boldsymbol{\omega}) = z_c, \qquad c = 1, \ldots, n_c, \tag{15}$$

The operator derivatives in this case are trivial so the interpolator can be excluded from the workflow. Otherwise, the current interpolation kernel based on the multilinear approach would require too much time for simulations with large number of components. This problem can be easily overpass by utilizing the barycentric interpolation similar to Zaydullin et al. (2013). In any case, that does not affect Jacobian assembly kernel in any respect, since pre-computed operator values and derivatives serve as an input. Figure 4(a) shows significant improvement of Jacobian assembly kernel especially for small block sizes. It is also clear that the basic kernel performance is unstable for various block sizes, while the improved kernel demonstrates gradual increase of time required for assembly. The speedup over single-thread CPU assembly for improved kernel is also stable fluctuating between 30 and 37 for various block sizes.



a. Time for a single Jacobian assembly with base and improved kernel b. Speedup of improved kernel over CPU

Figure 4—Jacobian assembly kernel benchmark for SPE10 model, 40 layers

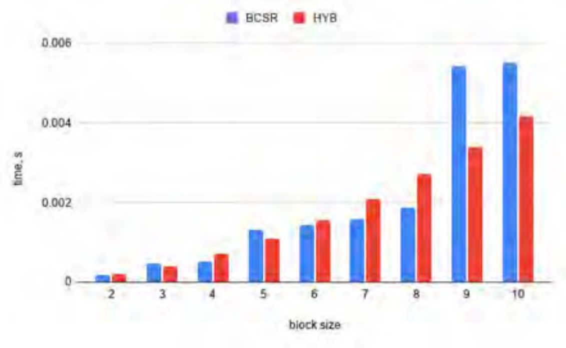
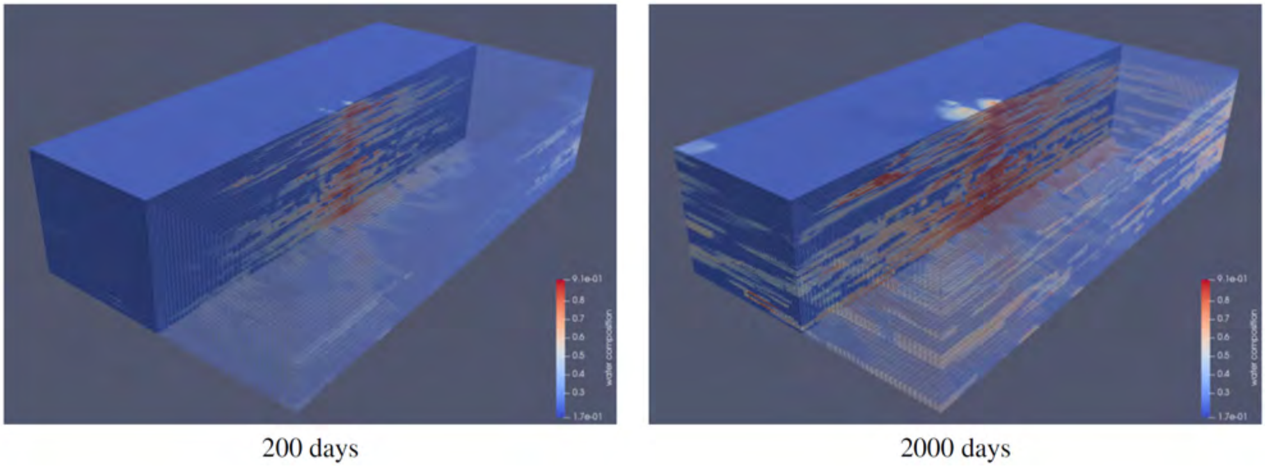


Figure 5—SPMV benchmark for SPE10 model, 20 layers

SPE10

The SPE10 test case (Christie and Blunt, 2001), initially created to compare upscaling techniques, is probably the most commonly used model to benchmark the performance of reservoir simulators. Due to its highly heterogeneous permeability distribution, achieving 10 orders of magnitude, and considerable size of 1.1 million cells, this model is quite challenging for both linear and nonlinear solvers. An extensive overview of the performance achievements by different simulators on this model is given by Esler et al. (2014).

Figure 6 demonstrates the heterogeneous behaviour of waterflooding process of this artificial model scaled vertically by a factor of 3. Water displaces oil from the center, where the injection well is located, to corners with producers, following various flow paths. Water injection is performed ata high rate of Q_{inj} = 794.93 m^3d^{-1} causing very fast breakthrough to all four producers, as can be seen from Figure 6. Performance results can be found in Table 1 and Table 2.



200 days

2000 days

Figure 6—Water composition distribution in SPE10 model at different timesteps.

Table 1—Overall simulation performance of SPE10 on different platforms: CPU1 - Intel Core i7-8086K; CPU2 - 2 × Intel Xeon E5-2640 v4; GPU - NVidia GeForce RTX2080 Ti. Sequential run is denoted by 's' in brackets, multithread run - with the amount of threads.

Platform	TS	NI	LI	Init, s	Jacobian, s	Setup, s	Solve, s	Total, s
CPU1 (s)	68	383	1767	7.36	124.70	363.31	534.46	1039.32
CPU2 (20)	68	376	1751	14.82	22.00	128.16	215.74	394.88
GPU	68	417	1925	7.47	10.47	43.80	39.11	120.80

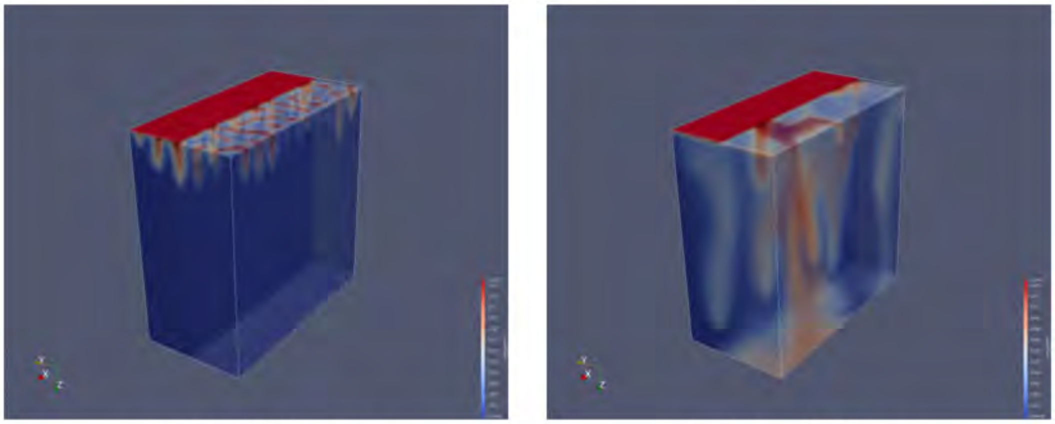
Table 2—Detailed simulation performance of SPE10 on different platforms: CPU1 - Intel Core i7-8086K; CPU2 - 2 × Intel Xeon E5-2640 v4; GPU - NVidia GeForce RTX2080 Ti. Sequential run is denoted by 's' in brackets, multithread run - with the amount of threads.

Platform	CPR	setup, s AMG	ILU(0)	GMRES	SPMV	solve, s CPR	AMG	ILU(0)
CPU1 (s)	23.39	313.29	26.62	102.75	103.35	66.86	279.87	84.98
CPU2 (20)	35.76	54.54	37.85	27.86	21.83	13.61	60.50	113.76
GPU	2.13	28.67	12.99	6.57	4.39	2.62	5.55	24.37

Carbon dioxide storage

Geological storage of CO_2 is critically important for the reduction of greenhouse gas emissions. Due to the buoyant characteristic of injected gas and the complex geology of subsurface reservoirs, most injected CO_2 rapidly migrates to the top of the reservoir. The detailed behavior of gravity-induced instabilities can be modeled using two-phase flow with gravity currents and convective dissolution in the presence of the capillary transition zone (CTZ), see detailes in Lyu et al. (2021b).

Figure 7 demonstrates the CO_2 concentration for the simulation with single-phase brine after 100 and 400 years of CO_2 injection. The model is based on an unstructured triangular extruded mesh with 100 layers and 1.1 million grid blocks in total. The top layer was initialized with constant CO_2 composition of 0.0125 (dissolution limit), while pure brine filled the rest of the reservoir. The model was simulated for 3000 years with a maximum timestep of 365 days. Simulation timings can be found in Table 3 and Table 4.



a. After 100 years

b. After 400 years

Figure 7—*CO_2* concentration in brine during a long-term sequestration

Table 3—Overall simulation performance of carbon dioxide storage model on different platforms: CPU1 - Intel Core i7-8086K; CPU2 - 2 × Intel Xeon E5-2640 v4; GPU - NVidia GeForce RTX2080 Ti. Sequential run is denoted by '*s*' in brackets, multithread run - with the amount of threads.

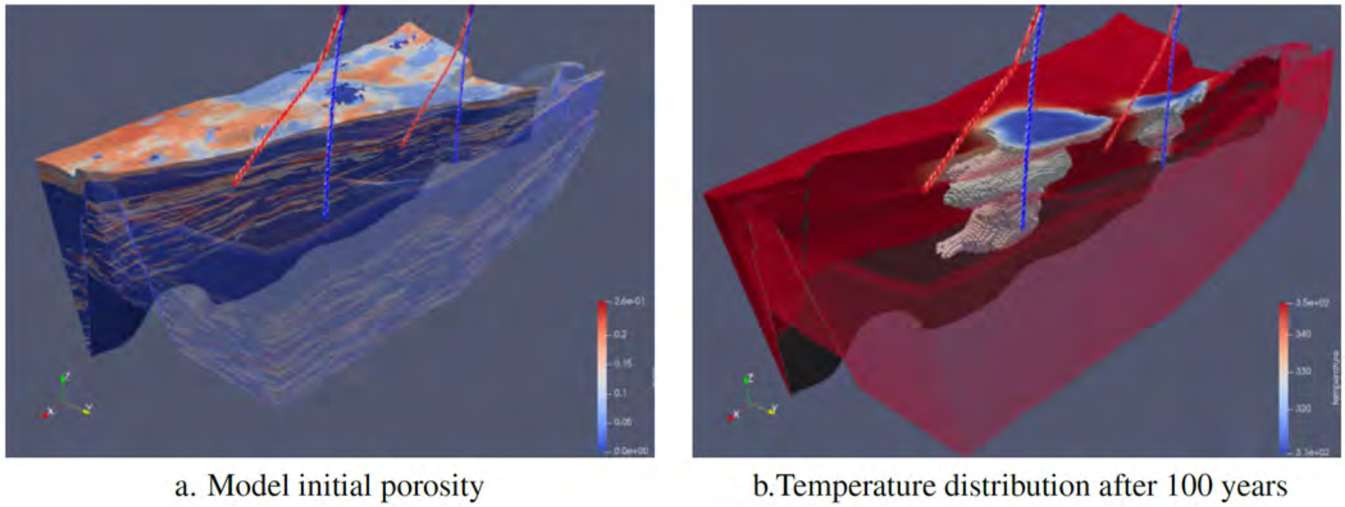
Platform	TS	NI	LI	Init, s	Jacobian, s	Setup, s	Solve, s	Total, s
CPU1 (s)	3015	6696	57726	25.4	3523.0	24193.9	20110.4	48400.1
CPU2 (20)	3015	6790	61438	35.6	623.7	3761.3	8363.8	13682.3
GPU	3015	6725	57487	37.5	204.1	1114.1	1183.1	3333.9

Table 4—Detailed simulation performance of carbon dioxide storage model on different platforms: CPU1 - Intel Core i7-8086K; CPU2 - 2 × Intel Xeon E5-2640 v4; GPU - NVidia GeForce RTX2080 Ti. Sequential run is denoted by '*s*' in brackets, multithread run - with the amount of threads

Platform	CPR	setup, s AMG	ILU(0)	GMRES	SPMV	solve, s CPR	AMG	ILU(0)
CPU1 (s)	415.81	23319.64	458.39	3334.55	1580.94	1985.67	12205.89	2584.30
CPU1 (20)	451.97	2594.61	714.56	680.02	370.41	535.49	2998.15	4150.05
GPU	24.91	890.56	198.64	119.93	109.92	71.87	254.68	736.57

Realistic geothermal model

The reservoir under investigation is located in the West Netherlands Basin (WNB), which is an inverted rift basin in the Netherlands. The reservoir properties of Delft Sandstone have been extensively studied before by Willems et al. (2016, 2017). Figure 8(a) shows the porosity distribution at the geological resolution of the target reservoir scaled vertically by a factor of 3. The model includes intersections of sandstone and shale facies. The facies distribution corresponds to circa 0.8 million grid blocks for the sandstone and 2.4 million blocks for shale facies.



a. Model initial porosity

b.Temperature distribution after 100 years

Figure 8—Geological model of aqueous reservoir with two geothermal doublets.

Even though the water mainly flows through the sandstone formation, for thermal simulation it is crucially important to take shale facies into account too, as was shown in the recent benchmark study by Wang et al. (2020). The presence of the shale layers in the simulation allows the use of higher discharge rates that result in higher energy production for an equivalent system lifetime. The predicted lifetime of both doublets is significantly extended when the shale layers are included in the model. As the injected cold water transports through the sandstone layers, it is re-heated, extracting energy from the sandstone layers. As time evolves, a temperature gradient is built up between the sandstone bodies and the neighbouring shale layers with the shales providing thermal recharge by heat conduction. The spatial intercalation of the sandstone and shale facies increases the contact area between them and amplifies the effect of the thermal recharge.

Using the full model with 3.2 million cells, we computed the forecast for 100 years of two geothermal doublets production with the maximum time step of 365 days. The simulation results (cold water plums distribution) can be seen in Figure 8(b). Simulation performance of this model is shown in Table 5 and Table 6.

Table 5—Overall simulation performance of geothermal model on different platforms: CPU1 - Intel Core i7-8086K; CPU2 - 2 × Intel Xeon E5-2640 v4; GPU - NVidia GeForce RTX2080 Ti. Sequential run is denoted by 's' in brackets, multithread run - with the amount of threads.

Platform	TS	NI	LI	Init, s	Jacobian, s	Setup, s	Solve, s	Total, s
CPU1 (s)	107	287	4819	78.88	219.99	819.40	3885.12	5019.17
CPU2 (20)	107	291	4916	103.04	35.21	305.41	1506.01	1976.64
GPU	107	288	5161	78.72	18.03	78.51	276.98	486.60

Table 6—Detailed simulation performance of geothermal model on different platforms: CPU1 - Intel Core i7-8086K; CPU2 - 2 × Intel Xeon E5-2640 v4; GPU - NVidia GeForce RTX2080 Ti. Sequential run is denoted by 's' in brackets, multithread run - with the amount of threads

Platform	CPR	setup, s AMG	ILU(0)	GMRES	SPMV	solve, s CPR	AMG	ILU(0)
CPU1 (s)	47.30	716.40	55.70	984.31	639.01	448.76	1885.87	566.18
CPU2 (20)	80.23	141.41	83.77	221.20	120.91	97.44	395.57	791.80
GPU	4.04	49.53	24.93	43.95	25.94	16.66	31.93	184.45

Conclusions

Delft Advanced Research Terra Simulator (DARTS) framework is built on top of the Operator-Based Linearization approach. It substantially simplifies Jacobian construction and reduces the time required for porting simulation code to different architectures, such as GPU. Proving this claim, we demonstrated three different examples of fully-offloaded GPU simulations relevant to energy transition: the classical hydrocarbon production SPE10 case, carbon dioxide long-term storage and a realistic model of geothermal energy production. To the best of our knowledge, these are the first simulations of a carbon storage model and a geothermal field fully offloaded to a GPU device.

Compared to sequential execution on a high-frequency modern desktop CPU, the multithread version reduces simulation time by 2.5-3.5 folds on a server node with two sockets, depending on a particular application. At the same time, the GPU version demonstrated overall improvement in the range of 8x-14x (10x-14x without sequential initialization stage). The SPE10 problem takes DARTS only around 100 seconds which is comparable to industrial-grade simulators. DARTS provides a forecast for 100 years with a 3.2 million grid blocks geological model in only 7 minutes, while the forecast for 3000 years for carbon dioxide sequestration scenario on a 1.1 million unstructured mesh takes less than an hour. All results were achieved on a regular workstation equipped with a gaming GPU graphics card.

The developed GPU linear solver uses available open-source codes as much as possible and is based on the standard BCSR matrix format. This minimized the development time and maximized the applicability of the linear solver. It can be immediately used for the whole variety of other problems which can be solved in DARTS: modelling of chemical reactions with dissolution and precipitation at reservoirs and lab scales, simulation of flow fully coupled with geomechanics, etc. The improvement of Jacobian assembly and the speedup of professionally-tuned individual parts of the linear solver indicate that overall performance can be improved even more. Revision of underlying storage structures and access patterns is required to increase the simulation efficiency further and will be the focus of our future research.

Acknowledgements

We would like to acknowledge TOTAL S.A. for financial support of this research. We would like to thank Hui Cao, Rustem Zaydullin and Arthur Yuldashev for their insightful comments and suggestions.

References

Abdelfattah, A., Ltaief, H., and Keyes, D. (2015). High performance multi-gpu spmv for multi-component pde-based applications. In European Conference on Parallel Processing, pages 601–612. Springer.

Aziz, K. and Settari, T. (1979). *Petroleum Reservoir Simulation*. Applied Science Publishers.

Bell, N. and Garland, M. (2009). Implementing sparse matrix-vector multiplication on throughput-oriented processors. In Proceedings of the conference on high performance computing networking, storage and analysis, pages 1–11.

Cao, H., Crumpton, P., and Schrader, M. (2009). Efficient general formulation approach for modeling complex physics. volume **2**, pages 1075–1086.

Christie, M. and Blunt, M. (2001). Tenth spe comparative solution project: A comparison of upscaling techniques. *SPE Reservoir Evaluation and Engineering*, **4**(4):308–316.

Dake, L. P. (1983). *Fundamentals of reservoir engineering*, volume **8**. Elsevier.

DARTS (2019). *Delft Advanced Research Terra Simulator*. https://darts.citg.tudelft.nl.

Esler, K. et al. (2014). Realizing the potential of gpus for reservoir simulation. In ECMOR XIV-14th European Conference on the Mathematics of Oil Recovery.

Haugen, K. and Beckner, B. (2015). A general flow equation framework. SPE Reservoir Simulation Symposium 2015, **2**:1310–1318.

Khait, M. (2019). *Delft Advanced Research Terra Simulator: General Purpose Reservoir Simulator with Operator-Based Linearization*. PhD thesis, TU Delft.

Khait, M. and Voskov, D. (2017a). Gpu-offloaded general purpose simulator for multiphase flow in porous media. In SPE Reservoir Simulation Conference. Society of Petroleum Engineers.

Khait, M. and Voskov, D. (2018a). Adaptive parameterization for solving of thermal/compositional nonlinear flow and transport with buoyancy. *SPE Journal*, **23**:522–534.

Khait, M. and Voskov, D. (2018b). Operator-based linearization for efficient modeling of geothermal processes. *Geothermics*, **74**:7–18.

Khait, M. and Voskov, D. V. (2017b). Operator-based linearization for general purpose reservoir simulation. *Journal of Petroleum Science and Engineering*, **157**:990 – 998.

Lyu, X., Khait, M., and Voskov, D. (2021a). Operator-based linearization approach for modelling of multiphase flow with buoyancy and capillarity. *SPE Journal*.

Lyu, X., Voskov, D., and Rossen, W. R. (2021b). Numerical investigations of foam-assisted co2 storage in saline aquifers. *International Journal of Greenhouse Gas Control*, **108**:103314.

Naumov, M., Arsaev, M., Castonguay, P., Cohen, J., Demouth, J., Eaton, J., Layton, S., Markovskiy, N., Reguly, I., Sakharnykh, N., et al. (2015). Amgx: A library for gpu accelerated algebraic multigrid and preconditioned iterative methods. *SIAM Journal on Scientific Computing*, **37**(5):S602–S626.

Peaceman, D. W. (2000). *Fundamentals of numerical reservoir simulation*, volume **6**. Elsevier.

Saad, Y. (1993). A flexible inner-outer preconditioned gmres algorithm. *SIAM Journal on Scientific Computing*, **14**(2):461–469.

Schlumberger (2011). *Eclipse reference manual 2011.2. Technical report*, Schlumberger, Houston.

Vanden, K. and Orkwis, P. (1996). Comparison of numerical and analytical jacobians. *AIAA Journal*, **34**(6):1125–1129.

Voskov, D. V. (2017). Operator-based linearization approach for modeling of multiphase multi-component flow in porous media. *Journal of Computational Physics*, **337**:275–288.

Wallis, J. (1983). Incomplete gaussian elimination as a preconditioning for generalized conjugate gradient acceleration. Proc. of 7th SPE Symposium on Reservoir Simulation.

Wallis, J., Kendall, R., Little, T., and Nolen, J. (1985). Constrained residual acceleration of conjugate residual methods. Proc. of 8th SPE Symposium on Reservoir Simulation.

Wang, Y., Voskov, D., Khait, M., and Bruhn, D. (2020). An efficient numerical simulator for geothermal simulation: A benchmark study. *Applied Energy*, 264.

Willems, C., Nick, H., Weltje, G., and Bruhn, D. (2017). An evaluation of interferences in heat production from low enthalpy geothermal doublets systems. *Energy*, **135**:500–512.

Willems, C. J. L., Goense, T., Nick, H. M., and Bruhn, D. F. (2016). The relation between well spacing and net present value in fluvial hot sedimentary aquifer geothermal doublets; a West Netherlands Basin case study. In Workshop on Geothermal Reservoir Engineering.

Xu, T., Spycher, N., Sonnenthal, E., Zhang, G., Zheng, L., and Pruess, K. (2011). Toughreact version 2.0: A simulator for subsurface reactive transport under non-isothermal multiphase flow conditions. *Computers and Geosciences*, **37**(6):763–774.

Younis, R. (2011). *Modern Advances in Software and Solution Algorithms for Reservoir Simulation*. PhD thesis, Stanford University.

Zaydullin, R., Voskov, D., and Tchelepi, H. (2013). Nonlinear formulation based on an equation-of-state free method for compositional flow simulation. *SPE Journal*, **18**(2):264–273.

Zhou, Y. (2012). *Parallel General-Purpose Reservoir Simulation with Coupled Reservoir Models and Multi-Segment Wells*. PhD Thesis, Stanford University.