Proceedings World Geothermal Congress 2020+1
Reykjavik, Iceland, April - October 2021

High Performance Simulation Framework for Geothermal Applications

Mark Khait[1], Yang Wang[1], Denis Voskov[1,2]

[1]Delft University of Technology, Stevinweg 1, 2628 CN Delft, Netherlands

[2]Stanford University, CA 94305-4007, USA

Keywords: Low-Enthalpy Geothermal, Fully Implicit, Operator-Based Linearization, Constraint Pressure Residual, GPU

ABSTRACT

Numerical modelling of multiphase multicomponent flow coupled with mass and energy transport in porous media is crucially important for geothermal applications. To deliver robust simulation results, a fully or adaptive implicit method is usually employed, creating a highly nonlinear system of equations. It is then solved with the Newton-Raphson method, which requires a linearization procedure to assemble a Jacobian matrix.

Operator-Based Linearization (OBL) approach allows detaching property computations from the linearization stage by using piece-wise multilinear approximations of state-dependent operators related to complex physics. The values of operators used for interpolation are computed adaptively in the parameter-space domain, which is uniformly discretized with the desired accuracy. As the result, the simulation performance does not depend on the cost of property computations, making possible to use expensive equation-of-state formulations including external packages (e.g., IAPWS-97) for an accurate representation of governing physics without penalizing runtime. On the other hand, the implementation of the simulation framework is significantly simplified, which allows improving the simulation performance further by executing the complete simulation loop on GPU architecture.

The integrated open-source framework Delft Advanced Research Terra Simulator (DARTS) is built around OBL concept and provides flexible, modular and computationally efficient modelling package. In this work, we evaluate the computational performance of DARTS for geothermal applications using realistic reservoir model 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 Constraint Pressure Residual solver.

1. INTRODUCTION

Numerical modelling of multiphase multicomponent flow coupled with mass and energy transport in porous media is crucially important for geothermal applications. 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, 1979) On the other hand, after discretization is applied to governing Partial Differential Equations (PDEs) of a problem, the resulting nonlinear system represents different tightly coupled physical processes and 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 ill-conditioned). Precisely the solution of such systems takes most of the simulation time in most practical applications.

Conventionally used in most practical applications, Newton-based nonlinear solvers require linearization. Several conventional linearization approaches exist, though none of them is robust, flexible, and computationally efficient all at once. Numerical derivatives provide flexibility in the nonlinear formulations (Xu et. al., 2011), but a simulation based on numerical derivatives may lack robustness and performance (Vanden, 1996). Straightforward hand-differentiation is the state-of-the-art strategy in modern commercial simulators (Schlumberger, 2011, Cao et. al., 2009). 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 Automatic Differentiation (AD) technique allows preserving both flexibility and robustness in derivative computations. In reservoir simulations, 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 simulations.

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. The proposed approach was utilized for molar formulation. A similar approach can be designed for the natural formulation, but it requires dealing with several parameter spaces and switching between them.

In the OBL approach, 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 hyperrectangle helps to preserve the balance between the accuracy of the approximation and the performance of the nonlinear solver (Khait and Voskov, 2017). 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. At the same time, the simulation performance does not depend on the cost of property computations, making possible to use expensive equation-of-state formulations including external packages (e.g., IAPWS-97) for an accurate representation of governing physics without penalizing runtime.

In this work, we evaluate the computational performance of DARTS for a geothermal application of practical interest on both CPU and GPU platforms. We provide a detailed performance comparison of workflow pieces composing Jacobian assembly and linear system solution, including both stages of Constraint Pressure Residual (CPR) preconditioner.

2. METHOD

We briefly describe the governing conservation equations used in this study in the conventional and operator forms.

2.1 Governing equations

In general, aqueous brine is used as the fluid for thermal circulation in geothermal development. For some applications, CO2 has been proposed as a heat carrier. In addition, minerals can be dissolved by the brine with a number of chemical reactions (Kala and Voskov, 2019), making the fluid chemistry even more complicated, and hydrocarbon components can be mixed with brine and co-produced (Khait and Voskov, 2018). Such type of models requires a complicated equation-of-state (EoS) to describe realistic phase behavior.

In the basic situation, only the water component exists in the studied system. Although only a single component is involved, both liquid and gaseous phases are present in high-enthalpy systems. In this case, the complex EoS of water is required for accurate characterization, as described by Kretzschmar and Wagner (2007). The large contrast in thermodynamic properties between liquid water and saturated steam should also be taken into consideration for efficient simulations in high-enthalpy systems.

Here, we consider the governing equations and nonlinear formulations for a single-phase thermal simulation with water, which can be described by mass and energy conservation equations:

$$\frac{\partial}{\partial t}(\phi\rho) - div(\rho u) + \rho\tilde{q} = 0, \tag{1}$$

$$\frac{\partial}{\partial t}(\phi U + (1-\phi)U_r) - div(h\rho u) + div(\kappa_t \nabla T) + h\rho\tilde{q} = 0, \tag{2}$$

where: ϕ is porosity, ρ is water molar density, $U = h\rho$ is water internal energy, $U_r = C_r(T - T_{ref})$ is rock internal energy, h is water enthalpy, κ_t is total thermal conduction, C_r is the volumetric heat capacity of rock while T and T_{ref} are current and reference temperatures. Darcy's law is used to describe the velocity of fluid flow

$$u = \frac{K}{\mu}\nabla p, \tag{3}$$

where: K is permeability tensor, μ is water viscosity, p is pressure. The difference in water density for different temperatures is neglected and therefore buoyancy forces are not taken into account. Lastly, rock is considered to be compressible, which is reflected by the change of porosity through:

$$\phi = \phi_0(1 + c_r(p - p_{ref}), \tag{4}$$

where ϕ_0 is initial porosity at reference pressure p_{ref}, while c_r is rock compressibility. Next, Darcy's law is substituted into the governing equation and the resulting nonlinear equations are discretized with finite-volume method in space on a general unstructured mesh and with backward Euler approximation in time:

$$V[(\phi\rho) - (\phi\rho)^n] - \Delta t \sum_l \frac{\rho^l \Gamma^l \Delta p^l}{\mu^l} + \Delta t \rho q = 0, \tag{5}$$

$$[(\phi U + (1-\phi)U_r) - (\phi U + (1-\phi)U_r)^n] - \Delta t \sum_l \left(\frac{h^l \rho^l \Gamma^l \Delta p^l}{\mu^l} + \Gamma_c^l \Delta T^l\right) + \Delta t h\rho q = 0, \tag{6}$$

where V is the control volume, n denotes variables defined at a previous timestep, $q = \tilde{q}V$ is the fluid source/sink term, Γ^l is the geometric part of convective transmissibility, and Γ_c^l is the conductive transmissibility which is expressed with the help of geometrical part of conductive transmissibility Γ_g^l as:

$$\Gamma_c^l = \Gamma_g^l \kappa_t = \Gamma_g^l(\phi\kappa + (1-\phi)\kappa_r), \tag{7}$$

The set of equations above are solved in a fully-coupled fully-implicit manner in DARTS using the molar nonlinear formulation (Faust and Mercer, 1979 and Wong et al., 2015), where pressure and enthalpy are selected as primary variables.

2.1 Operator form

The discretized mass and energy governing equations are represented in the operator form follwoing the OBL approach. State operators are distinquished as functions of a physical state $\omega = \{p, h\}$ (Voskov, 2017; Khait and Voskov, 2018). Pressure and enthalpy are taken as the unified state variables for a given control volume. Flux-related fluid properties are defined by the physical state of the upstream block, determined at interface l. Therefore, the discretized mass conservation equation in operator form reads

$$\phi_0 V\big(\alpha(\omega) - \alpha(\omega^n)\big) + \sum_l \Delta t \Gamma^l \Delta p^l \beta(\omega) + \theta(\xi, \omega, u) = 0, \tag{8}$$

$$\alpha(\omega) = \rho\,\big(1 + c_r(p - p_{\text{ref}})\big), \tag{9}$$

$$\beta(\omega) = \frac{\rho}{\mu}. \tag{10}$$

Similarly, the discretized energy conservation equation in operator form becomes

$$\begin{aligned} &V\big[\phi_0\big(\alpha_e(\omega) - \alpha_e(\omega^n)\big) + (1 - \phi_0)U_r\big(\alpha_r(\omega) - \alpha_r(\omega^n)\big)\,\big] + \\ &+\sum_l \Delta t \Gamma^l \beta_e(\omega)\Delta p^l + \sum_l \Delta t \Gamma_c^l[\phi_0 \gamma_e(\omega) + (1 - \phi_0)\kappa_r \alpha_r(\omega)]\,\Delta T^l + \theta_e(\xi, \omega, u) = 0, \end{aligned} \tag{11}$$

$$\alpha_e(\omega) = U\big(1 + c_r(p - p_{\text{ref}})\big), \tag{12}$$

$$\alpha_r(\omega) = \frac{1}{1 + c_r(p - p_{\text{ref}})}, \tag{13}$$

$$\beta_e(\omega) = h\,\frac{\rho}{\mu}, \tag{14}$$

$$\gamma_e(\omega) = \kappa\big(1 + c_r(p - p_{\text{ref}})\big). \tag{15}$$

This representation significantly simplifies the general-purpose simulation framework. Instead of performing a complex evaluation of each property and its derivatives during the simulation, we can parameterize the state-dependent operators $\alpha, \beta, \alpha_e, \alpha_r, \beta_e, \gamma_e$ in the two-dimensional space of unknowns p, h with a limited number of supporting points and use bilinear interpolation to evaluate them (Voskov, 2017). This not only makes the Jacobian assemble simpler and more general, but also improves the performance sincealmost all expensive property evaluations can be skipped. Besides, due to the piece-wise multilinear approximation of physical operators, the system will become more linear and the performance of nonlinear solver can be improved (Khait and Voskov, 2018).

3. IMPLEMENTATION

DARTS is a Python package with binary extensions written on C++/CUDA. This choice allows for simple embedding of the simulation framework into existing history matching/uncertainty quantification/optimization frameworks written in Python. At the same time, the computational performance is provided by the binary extensions, compiled for Windows/Linux platforms and different Python versions. Switching between CPU and GPU versions is performed through Python interface, triggering substitution of conventional CPU-based Jacobian assembly and linear solver with their GPU couterparts. The other components of simulation workflow, such as space discretization, wells and physical property computations remain unchanged.

DARTS has the same initialization scheme for both GPU and CPU versions. The GPU version first loads the required initial data to GPU memory and then performs all major computations on the device. For example, in order to initialize the GPU version of static interpolator responsible for interpolation of all state-dependent operators and evaluation of their derivatives, all supporting points are first computed on CPU and then copied to device memory. Alternatively, they can be loaded from a file and then also be sent down to device memory if one was created during a previous simulation provided that the fluid properties and physical space parameterization settings have not been changed since then.

3.1 Jacobian assembly on GPU

The initialization data for Jacobian assembly includes a connection list, rock properties (i.e., porosity, heat capacity and thermal conduction) and initial reservoir state arrays. The Jacobian structure is assumed to be fixed during simulation, so it is also initialized once on CPU and copied to the device memory prior to the run.

Interpolation of operator values and derivatives is performed as a preparatory step before every Jacobian assembly. The 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 ω. The data layout is analogous to the one used for CPU interpolator version, where values corresponding to a given cell are grouped together, 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.

The Jacobian assembly GPU kernel, as well as all components of GPU-based linear solver (except AMG), is based on classical Block CSR matrix format. It is known that this format is not the best in terms of performance (Bell and Graland, 2009). However, it was chosen out of compatibility considerations: both with DARTS code base and available linear solver implementations on GPU. This kernel is also implemented on a thread-per-cell basis, and therefore global memory accesses are not coalesced here similarly to the interpolation kernel. In order 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, the 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 well 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 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 reduce the overhead to a minimum using asynchronous memory routines and CUDA streams or even CUDA graphs. Those instruments allow overlapping of kernel execution and memory transfer. In that scenario, while the reservoir part is being assembled on GPU during corresponding kernel invocation, the well part is computed on CPU and transferred to device memory simultaneously. While the size of the well part is usually small, its evaluation and transfer should be possible to hide behind the assembly of the reservoir part completely.

3.2 Linear solver on GPU

A simple configuration of GPU linear solver based on a Krylov subspace iteration method with ILU(0) preconditioner 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) and 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 conduction- (thermal diffusion) or convection-dominated.

The CPR method is a two-stage preconditioner, where at the first stage, the pressure system is decoupled from the full system and solved separately, usually with AMG-based solver. Often a single V-cycle is enough for efficient preconditioning. At the second stage, the full system is processed by an ILU(0) preconditioner using the pressure solution from the first stage. This 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 either entirely on CPU or entirely on GPU using the Flexible Generalized Minimum Residual (FGMRES) iterative method (Saad, 1993) with CPR-based preconditioner. All matrix operations are performed in native BCSR format. The pressure system is decoupled from the FIM system using a True-IMPES reduction approach directly from the BCSR storage. Then, a single V-cycle of the AMG solver is used to obtain an approximation of the pressure solution. Finally, it is substituted in the full system and a block ILU(0) preconditioner is applied.

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 \textit{cudaOccupancyMaxPotentialBlockSize} routine.

For the first stage of CPR preconditioner in the CPU version of DARTS linear solver, we use a proprietary AMG library. The GPU version is based on the open-source library AMGX (Naumov et. al., 2015) - specifically, the commit ID 0e32e35 was used. It is worth noting that the particualr choice of AMG settings and algorithms on CPU and GPU is similar, but not identical. The second stage of preconditioning uses block ILU(0) algorithm for both CPU and GPU. The GPU version of linear solver relies on the bsrilu02 routine of the cuSPARSE library.

4. NUMERICAL RESULTS AND ANALYSIS

We used two computer systems to perform simulations: a workstation with a top-tier gaming GPU and a 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 the case of the cluster node.

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 1 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.

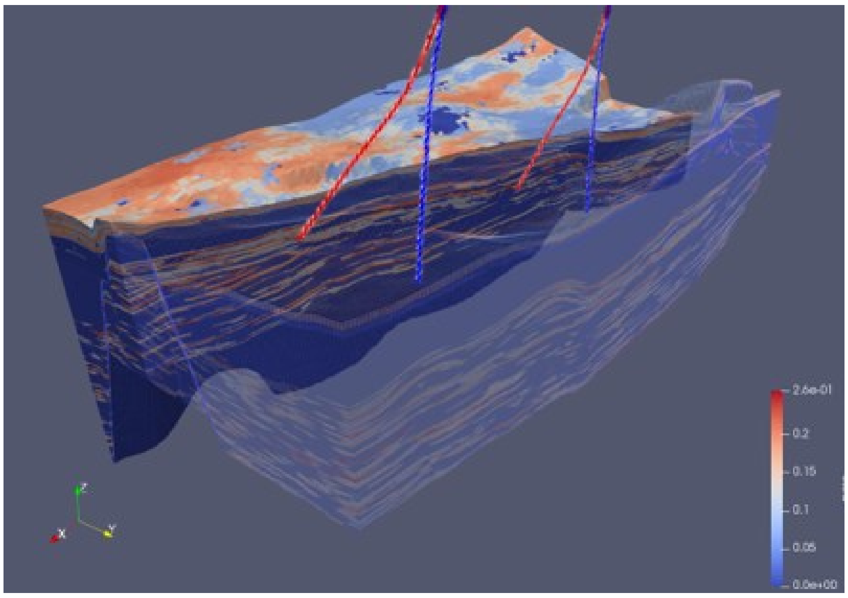

Figure 1: Initial porosity destribution of the geothermal model.

Even though the water mainly flows through the sandstone formation, for the thermal simulation it is crucially important to take shale facies into account too, as was shown in the recent benchmark study by Wang (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 neighboring 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.

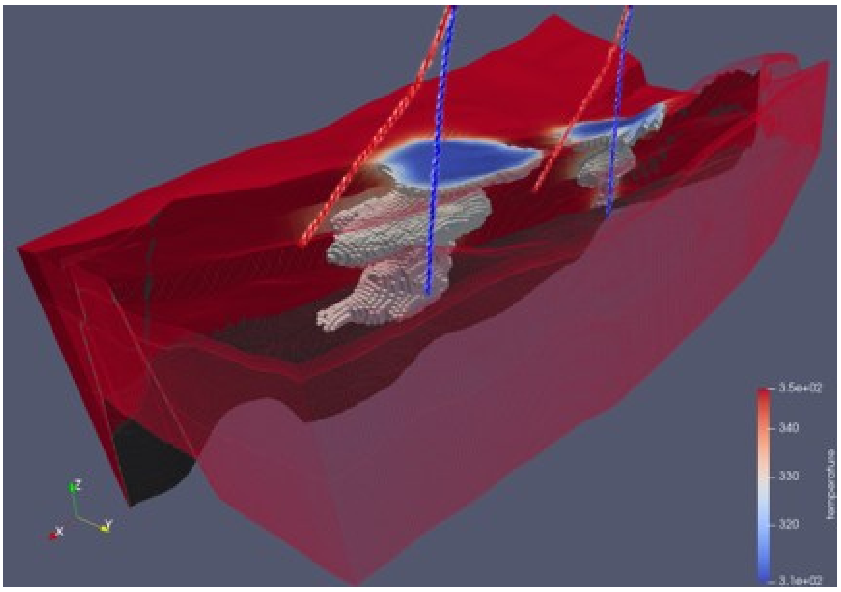

Figure 2: Temperature distribution after 100 years of production

Using the full model with 3.2 mln. cells, we computed the forecast for 100 years of two geothermal doublets production with the maximum time step of 365 days. Final cold water plumes distribution can be seen in Figure 2.

Table 1: Overall simulation performance on different platforms: CPU1 - Intel Core i7-8086K; CPU2 - 2 x 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

	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

The overall simulation performance of this model is shown in Table 1. The first line corresponds to the fastest single-thread CPU simulation run: it was achieved on the workstation with disabled OpenMP clauses. Such configuration always leads to the fastest single-thread simulation: enabling OpenMP instructions and limiting the number of threads to one lead to a noticeable overhead of 10-15%. The second line corresponds to the fastest multi-thread simulation run on CPU, achieved on a cluster node with 20 OpenMP threads. We also tried to use 16, 18, 20, 22 and 24 threads for the simulation, but starting from 20 threads, simulation time stopped to improve and only fluctuated around the same value. It is expected, since Intel Xeon E5-2640 processor has 10 hardware cores (consequently, a node with 2 such processors has 20 hardware cores in total), while additional logical cores (or Threads, introduced by Hyper-Threading) do not help in case of memory-bound problems. The last line refers to the fully offloaded GPU simulation. The columns 2-4 show the number of time steps, nonlinear and linear iterations. The rest columns present the amount of time spent in initialization, Jacobian assembly, setup and solve phases of linear solver and, finally, the total simulation time.

The number of timesteps is identical between all runs – all timesteps have successfully converged using about the same number of nonlinear iterations. The difference in linear solver iterations is within 8% and is considered as insignificant. The initialization stage takes quite significant time. It is mostly explained by the time required for generation of operator values for the entire parameter space since the properties are computed by an external Python library, which is noticeably less efficient than property calculations based on C++. Nevertheless, in the scenarios where the simulation runtime matters the most (e.g., inverse modeling, uncertainty quantification or optimization), the fluid properties most likely to be constant, allowing to store calculated values of state operators during the first run and load them in the subsequent ones.

The overall performance of a single-thread CPU simulation was improved in multi-thread run by a factor of 2.5, while Jacobian assembly was 6.2 times faster. Initialization stage is not parallel; hence the 1.3 times slowdown is explained by lower single-thread performance of cluster CPU. The setup and solve phases almost equally benefited more from the multithread execution being 2.7 and 2.6 times faster accordingly. The GPU version is 10.3 times faster overall, while the simulation excluding initialization is 12 times faster than a single-thread CPU version. In particular, Jacobian assembly is 12 times faster, while setup and solve phases have been accelerated by 10.5 and 14 times respectively.

Table 2: Linear solver performance on different platforms: CPU1 - Intel Core i7-8086K; CPU2 - 2 x 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

	setup, s			**solve, s**				
	CPR	**AMG**	**ILU(0)**	**GMRES**	**SPMV**	**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

From Table 2 one can see the detailed information about the performance of all components of the linear solver on different platforms, all times are exclusive. For example, CPR setup column shows only time spent on the construction of the pressure matrix, not taking into account time spent on subsequent calls of AMG and ILU(0) preconditioning steps. The only exception is GMRES column - it includes time spent on sparse matrix-vector multiplication (SPMV, which is also shown separately), but excludes CPR solve time.

First of all, it is easy to see that some parts of the linear solver do not speed up during the parallel run on CPU: CPR setup and both ILU(0) stages. These procedures were not made parallel, so they run sequentially in multithread execution and together with initialization contribute to sequential part of code. The gain of multithread execution for parallel phases of linear solver is surprisingly consistent: all speedup factors range in between 4.4 and 5.3. Taking into consideration that Jacobian assembly became 6.2 times faster, one can estimate the upper bound of potential overall simulation speedup, theoretically achievable on a similar cluster node once every part of simulation code is made parallel with the same efficiency, to just 5-6 times.

The spectrum of speedup factors for various part of linear solver on GPU is much wider: from only 2.2 for ILU(0) setup phase to 59 for AMG solve. ILU(0) algorithm is substantially sequential, hence there is no surprise that both setup and solve stages have low speedup. The implementation from CUSPARSE library exploits multilievel parallel strategy being algorithmically equivalent to the sequential procedure. Multicoloring ILU(0) strategy should be able to use GPU potential better, though might result in higher number of linear iterations. CPR and AMG setup phases provide mid-range speedups of 11-15, explained by simplest kernel implementation for the first and complicated underlying algorithms fro the second. Finally, the algorithms related to sparse matrix-vector multiplication (GMRES, CPR solve and SPMV itself) exhibit similar efficiency being 22-27 times faster on GPU that on CPU. In overall, the GPU simulation on a single gaming card is already an order of magnitude faster, while the speedup factors of specific linear solver parts indicate the high potential for further improvement.

5. CONCLUSIONS

Delft Advanced Research Terra Simulator (DARTS) is a simulation framework capable of efficient modeling of geothermal processes. Its key component is Operator-Based Linearization (OBL) 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 performed a simulation of a realistic geological-scale model of geothermal energy production completely on GPU. To the best of our knowledge, this is the first simulation of a geothermal field fully offloaded to a GPU device.

The sequential CPU version of DARTS provides a forecast for 100 years for low-enthalpy realistic geological model with 3.2 million grid blocks in 1 hour 23 minutes using regular gaming workstation. The multithread version on a cluster node with two CPU sockets reduces the overall simulation time to 33 minutes. Finally, the GPU version allows to compute the same forecast in just 9 minutes using regular gaming GPU 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 minimizes the development time and maximizes the applicability of the linear solver. It can be immediately used for the whole variety of problems which can be solved in DARTS: low- and high-enthalpy geothermal energy production, subsurface storage and CO_2 sequestration, modeling of chemical reactions with dissolution and precipitation at reservoirs and lab scales, simulation of flow with geomechanics, etc.

Based on the speedup of professionally-tuned individual parts of the linear solver on GPU, its overall performance can be improved even more. The same stands for Jacobian assembly. 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.

REFERENCES

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

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

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

Cui, G., Zhang, L., Ren, B., Enechukwu, C., Liu, Y. and Ren, S.: Geothermal exploitation from depleted high temperature gas reservoirs via recycling supercritical CO2: Heat mining rate and salt precipitation effects. Applied Energy,183, (2016), 837–852

Faust, C. R. and Mercer, J. W: Geothermal reservoir simulation: 2. Numerical solution techniques for liquid-and vapor-dominated hydrothermal systems. *Water Resources Research*, 15, (1979), 31-46.

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

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

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

Schlumberger: ECLIPSE Reference Manual 2011.2. *Tech. rep., Schlumberger*, (2011), Houston

Kala K., Voskov D.: Element balance formulation in reactive compositional flow and transport with parameterization technique. *Computational Geosciences* (2019), 1-6.

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

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

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

Kretzschmar, H.J. and Wagner, W.: International Steam Tables: Properties of Water and Steam based on the Industrial Formulation IAPWS-IF97, *Springer Science & Business Media*, (2007).

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

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

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

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

Willems, C.J.L., Goense, T., Nick, H.M. and Bruhn, D.F.: The relation between well spacing and Net Present Value in fluvial Hot Sedimentary Aquifer geothermal doublets; a West Netherlands Basin case study. *Workshop on Geothermal Reservoir Engineering,* (2016)

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

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

Wong, Z.Y., Horne, R. and Voskov, D.: A Geothermal Reservoir Simulator in AD-GPRS. *World Geothermal Congress,* (2015)

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

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

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

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