Accelerating PFLOTRAN-OGS on GPUs using PETSc

Tobias Dyngeland and Anne C. Elster

Norwegian University of Science and Technology,
Dept. of Computer Science, Trondheim, Norway
`tobias.dyngeland@ntnu.no`, `elster@ntnu.no`

Abstract. With the evident effects of rapid climate change and society´s continuing dependence on fossil fuels, efficient modelling of reservoir behaviour for emerging CO_2 storage projects is in high demand. Due to the models' physical complexity and the non-linear nature of CO_2 storage processes, the computational demands are so significant that physical aspects, such as the dissolution effect, are ignored due to computational limits. State-of-the-art flow simulation codes use highly parallel hardware and programming models to handle large-domain simulations. At the time of this work, complex flow simulators had yet to fully investigate the benefits and impact of utilising accelerators like GPUs. This study explores the performance of accelerating the production code PFLOTRAN-OGS, developed by OpenGoSim, using GPUs through PETSc's recently built-in accelerated solvers. Our accelerated simulation is run on two test cases: `GW1`, a CO_2 storage case from OpenGoSim, and `SPE1`, a simple Black Oil benchmark from the Society of Petroleum Engineers. The preliminary benchmark indicates that a GPU-accelerated solver with a CPU-based framework gives an overall slower simulation. However, our profiling verifies that most of the time was spent on transferring matrices back and forth between the CPU and GPU, while the solver steps have significant speedup on the GPU. Our results thus show that the CUDA-accelerated PETSc FGMRES solver will be faster than its CPU counterpart once the complete code is moved to the GPU.

Keywords: GPU · Fluid simulation · CO_2 storage · PFLOTRAN.

1 Introduction

Modern-day oil and gas extractions from subsurface reservoirs and permanent CO_2 storage projects need detailed fluid flow models to ensure safety and efficiency. These models should predict field performance and ultimate recovery for various field development scenarios under different operational conditions [1]. However, accurate simulations of the reservoir development bring many challenges, as porous media is strongly heterogeneous and anisotropic [2]. Grids with a high aspect ratio are fully unstructured with polygonal cells. The phase behaviour is non-trivial as *phases* can appear and disappear as fluid *components* dissolve or vaporise. Coupling to wells can connect regions far away from each other, and the reservoir models are highly non-linear.

The Newton-Raphson (NR) method is commonly used for solving non-linear fluid equations due to its fast convergence towards the root [3]. This method requires the Jacobian inversion to calculate the input vector's next iteration. Inverting matrices is, however, computationally expensive, and today's approach is typically to iterate towards the inverted Jacobian by solving the linear system

$$\mathbf{J}(\mathbf{x}_k)\mathbf{x}_{k+1} = \mathbf{J}(\mathbf{x}_k)\mathbf{x}_k - R(\mathbf{x}_k).$$

Major contrasts in material properties, such as permeability and porosity, combined with high grid cell aspect ratios can lead to an ill-conditioned Jacobian and non-convergence with NR [4]. Using an ineffective preconditioner for the linear system can lead to the Krylov solver failing due to reaching the maximum number of linear iterations, resulting in smaller time steps.

State-of-the-art simulators use highly parallel hardware and programming models to distribute the workload and solve the equations efficiently. Message Passing Interface (MPI) [5] is a standard for communication libraries on compute clusters and includes implementations such as OpenMPI and MPICH. Note, however, that classical MPI does not have the concept of host and device memory [6, 7]. GPU-aware MPI does accept GPU memory pointers, but many MPI implementations do not utilise the Stream Queue, preventing pipelining.

At the start of this work, in the Summer of 2022, complex fluid simulators that solve systems of implicit equations primarily leveraged computing power from multiple CPUs. In [8, 9], Hammond predicted that accelerating such simulators would only achieve an x4 speed-up for realistic data. Das [10] showed through a simplified study of groundwater flow scenarios that numerical subsurface flow equations, such as Advection-Diffusion Eqn. (ADE) with the Finite Volume Method (FVM) discretisation and a Backward Euler scheme, can be accelerated with GPUs.

In our work, we investigate PFLOTRAN-OGS, a real-world multi-phase flow production code (e.g., it simulates both water and gas and mixing) by OpenGoSim, and explore how it can gain performance by using the new accelerated solvers in the Scalable Nonlinear Equation Solver (SNES) provided by The Portable Extensible Toolkit for Scientific Computing (PETSc) [11]. In the Summer of 2022, PFLOTRAN-OGS only used MPI as their programming model for parallelisation and had just started experimenting with accelerated kernels for their solvers. Later, OpenGoSim developed a proprietary, multi-threaded GPU/CPU C++ 11 version of the preconditioner Compressed Pressure Residual (CPR) that uses a multi-colour Incomplete Lower-Upper (ILU) factorisation method as the second stage and is now a stand-alone library called Wisp [12].

2 Mathematical Basis of Reservoir Simulations

Reservoir models consist of a set of *phases* and *components*. A component is a substance in the reservoir, e.g. *water* or CO_2 and the phase represents the state of a component, e.g. *liquid* and *gaseous*. Components can exist in multiple phases. This means that the gas component can dissolve in the liquid phase, and more than one phase can contribute to the flow of a component.

2.1 The Black Oil Model

The Black Oil Model is a three-phase, three-pseudo-component petroleum reservoir model incorporating compressibility and general mass transfer between phases [13, 14]. The components in this model are *water*, *oil*, and *gas*, which can exist in the phases *liquid*, *oleic*, and *gaseous*. The model's core is mass conservation, where the change in mass over a time step Δt should equal the difference in mass flowing in and out of a given volume. That is, following the mass balance equation

$$\phi\frac{\partial c}{\partial t} + \nabla(\mathbf{q}c) = Q, \tag{1}$$

where ϕ is the porosity, c is the concentration, $\mathbf{q}$ is the Darcy velocity, representing the volume of fluid flowing per unit area per unit time, and Q is a source term.

2.2 The Gas-Water Model

The Gas-Water model is a two-phase, two-component flow model that can describe CO_2-storage in deep saline water formations [15]. The conserved components, *CO_2* and *water*, can exist in the phases *liquid* and *gaseous*. Like the Black Oil model, the Gas-Water model revolves around a mass balance equation called the ADE. It combines *Reactive Transport* with the use of Darcy Flow and *Molecular Diffusion* through *Fick's First Law of Diffusion* in

$$\phi\frac{\partial c}{\partial t} + \nabla(\mathbf{q}c) - \phi D_m \nabla^2 c = Q, \tag{2}$$

where D_m is the *effective molecular diffusion coefficient* in the pore space.

2.3 Energy Balance

The energy balance equation

$$\frac{\partial}{\partial t}\left(\phi\sum_{\beta} S_\beta \eta_\beta U_\beta + (1-\phi)\rho_r \zeta_r T\right) + \nabla\cdot\sum_{\beta}\left(\mathbf{q}_\beta \eta_\beta H_\beta - \kappa\nabla T\right) = Q_h$$

describes the conservation of energy by equating the sum of the time rate of change of the energy term and the summation of the advection and conduction terms where β describes the different phases [16, 17, 14]. The rest of the variables are as follows: S is the saturation, η describes the molar density, U_β is the internal energy, ρ_r is the rock density, ζ_r is the heat capacity of the rock, T is the temperature in Kelvin, H is the enthalpy, and κ is the thermal conductivity.

2.4 The Numerical Solution

Common discretisation schemes for fluid flow simulations include variations of the Finite Element Method (FEM) and the FVM. PFLOTRAN discretises the reservoir with FVM. This method enforces flux conservation in and out of a Control Volume (CV) by directly integrating the governing equations and keeping track of the flux passing through the cell surfaces, as shown in Fig. 1. This gives the advantage of enhancing accuracy through two main approaches: restructuring the grid and increasing the number of integration points.

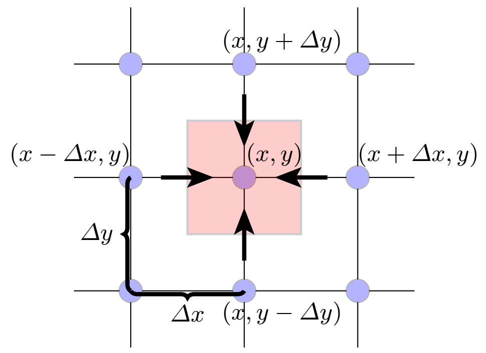

Fig. 1. Neighbouring fluxes into a CV in a 2D vertex-centred, structured FVM discretisation. Flux is illustrated above by the four arrows representing $\mathbf{F}_{(x,y-\Delta y)}$, $\mathbf{F}_{(x+\Delta x,y)}$, $\mathbf{F}_{(x-\Delta x,y)}$, $\mathbf{F}_{(x,y+\Delta y)}$, respectively. Their sign determines the direction of the flow.

A general approach for FVM:

1. Discretise the spatial domain with a number of grid points.
2. Integrate the governing equation over the domain.
3. Approximate the resulting integrals numerically.
4. Assemble and solve the discrete algebraic system of equations.

PFLOTRAN uses a first-order, cell-centred FVM scheme with upwind weighting and approximates the integrals numerically with a single integration point [18]. Today's industry standard is fully implicit solvers for the time step due to the stiffness of the equations and the ability to introduce constraints such as energy balance, faster convergence, and high parallelisability.

Applying the FVM to Eqn. 1 and 2 by using a single integration point per cell and utilising the Divergence Theorem on the flux integrals yields

$$\frac{A_{\alpha i}^{t+\Delta t} - A_{\alpha i}^{t}}{\Delta t} \Delta V_i + \sum_{n \in \{neighbours\}} \mathbf{F}_{\alpha,(i,n)}^{t+\Delta t} \Delta S_{(i,n)} - Q_{\alpha i}^{t+\Delta t} \Delta V_i = 0. \tag{3}$$

Eqn. 3 is then numerically integrated over the time domain using the Backwards Euler method and is solved with NR at each time step. The overall approach is the same for the Black Oil and Gas-Water models, with the difference being in the flux term, **F**, where the Gas-Water model includes molecular diffusion. The advective and diffusive fluxes have been combined into a single flux for simplicity.

3 The PFLOTRAN-OGS Simulation Code

Our work uses the PFLOTRAN-OGS application, Parallel Flow and Reactive Transport, an open-source branch of the original PFLOTRAN code primarily focusing on CO_2 storage simulations. PFLOTRAN is a highly parallel FORTRAN code running on a multi-CPU front end for simulating multi-phase, multicomponent and multiscale subsurface flow and reactive transport in a porous medium [19, 20]. Parallelism is achieved through domain decomposition to calculate the flux across subdomains with MPI, exemplified in Fig. 2.

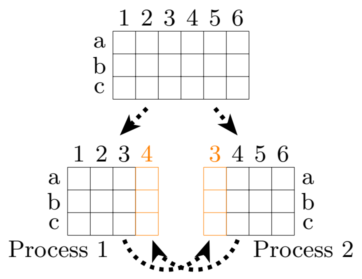

Fig. 2. Example of domain decomposition where the domain is split between two processes that exchange borders by sending a ghost vector. Each process creates an extra local column to store the received information.

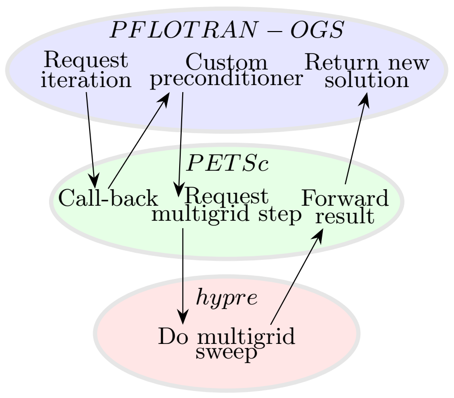

Fig. 3. PFLOTRAN-OGS - PETSc - *hypre* communication structure for the two-stage CPR preconditioner. Figure based on [12].

It incorporates the numerical software suites, PETSc, for solvers, data structures and communication, and *hypre* for its Algebraic Multigrid (AMG) preconditioner, BoomerAMG, which can be called through PETSc. Fig. 3 illustrates the communication paths between PFLOTRAN-OGS, PETSC, and *hypre* for the custom preconditioner CPR. CPR is the default preconditioner in PFLOTRAN-OGS and is a two-stage AMG-based preconditioner that uses Jacobi as a smoother and a processor-block Jacobi ILU(0) in the second stage (solve stage). Fig. 4 illustrates the V-cycle approach for an AMG.

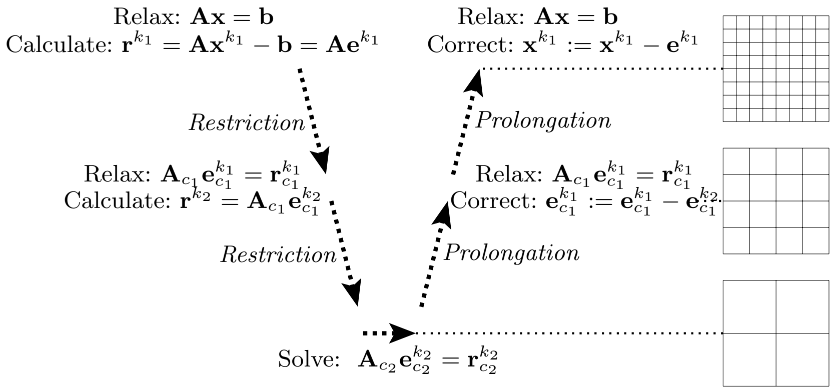
Relax: $\mathbf{Ax} = \mathbf{b}$
Calculate: $\mathbf{r}^{k_1} = \mathbf{Ax}^{k_1} - \mathbf{b} = \mathbf{Ae}^{k_1}$
Restriction
Relax: $\mathbf{A}_{c_1}\mathbf{e}^{k_1}_{c_1} = \mathbf{r}^{k_1}_{c_1}$
Calculate: $\mathbf{r}^{k_2} = \mathbf{A}_{c_1}\mathbf{e}^{k_2}_{c_1}$
Restriction
Solve: $\mathbf{A}_{c_2}\mathbf{e}^{k_2}_{c_2} = \mathbf{r}^{k_2}_{c_2}$
Prolongation
Relax: $\mathbf{A}_{c_1}\mathbf{e}^{k_1}_{c_1} = \mathbf{r}^{k_1}_{c_1}$
Correct: $\mathbf{e}^{k_1}_{c_1} := \mathbf{e}^{k_1}_{c_1} - \mathbf{e}^{k_2}_{c_1}$
Prolongation
Relax: $\mathbf{Ax} = \mathbf{b}$
Correct: $\mathbf{x}^{k_1} := \mathbf{x}^{k_1} - \mathbf{e}^{k_1}$

Fig. 4. V-cycle for 3-level multigrid calculations and corresponding grid coarsening on right. r describes the residual of approximation. The smoother is applied during the relaxation stage to smooth out high-frequency error field. Restriction operation interpolates the residual to a coarser version and recursively calls the multigrid method, making it multilevel. At coarsest grid, the equation can be solved with a direct solver.

4 Integration, Test Setup, and Testing Approach

PFLOTRAN-OGS installs a copy of PETSc 3.12.2, which predates the accelerated solvers in PETSc. Thus, to access the accelerated solver, PFLOTRAN-OGS had to be rebuilt with an updated PETSc version. To take advantage of PETSc and avoid issues with "PCShell", we intercepted the custom CPR preconditioner in the "CPRMake" subroutine and defaulted to the standard PETSc preconditioners. As of PETSc 3.16, PETSc requires that FORTRAN bindings be explicitly added since these are no longer added by default. In addition, CUDA options need to be added to access the accelerated solvers. To enable the accelerated AMG from *hypre*, *hypre* had to be configured with unified memory. We tried this, but our initial results were disappointing and not the focus of our study, so BoomerAMG was left out as a standalone preconditioner. Table 1 shows the four versions of PFLOTRAN-OGS, whereas Figure 5 shows the targeted acceleration and the simulation flow.

Table 1. Overview of the tested versions.

Version	**Preconditioner**	**Solver**	**Proc. Unit**	**PETSc**	**PFLOTRAN-OGS**
original	Shell	FGMRES	CPU	3.12.2	1.6
orig-P3.17	Shell	FGMRES	CPU	3.17.4	1.6
gpu	ILU	FGMRES	GPU	3.17.4	1.6
gpu	ILU/Block Jacobi	FGMRES	CPU	3.17.4	1.6
ilu-bjacobi	ILU/Block Jacobi	FGMRES	CPU	3.12.2	1.6

The versions in the above table are described as follows:

– *original* - baseline version following the PFLOTRAN-OGS installation guide.
– *ilu-bjacobi* - same as *original*, but with an interception of CPR.
– *orig-P3.17* - CPU version with an updated version of PETSc.
– *gpu* - version targetting the GPU with an updated version of PETSc and an interception of CPR. PETSc needs CUDA vectors and matrices to use the accelerated solver.

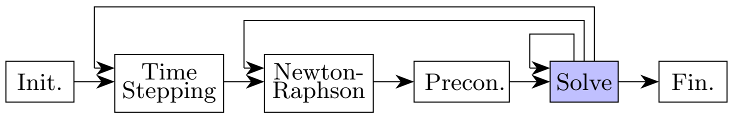

Fig. 5. Simplified overview of a simulation. Flow calculations are run first, followed by reactive transport at each time step. PETSc includes Time Stepping, Newton-Raphson, preconditioning, and the linear solver. When the solver has evaluated the Jacobian, the next NR iteration can start. If NR is within the error bounds, the next time step can happen. The Solve step, marked in blue, is the accelerated part of our experiments.

We run the simulations on two different test cases, `SPE1` [21] and `GW1` [12]. `SPE1` is a standard oil extractions benchmark from the Society of Petroleum Engineers. It's a small 10x10x3 test case, shown in Fig. 6, and functioned as an integration test for our acceleration [21]. It's a live oil/dry gas black-oil model with nearly immobile water. A single producer is controlled by bottom-hole pressure, and a single injector injects gas into the reservoir initially filled with undersaturated oil.

The second test case, `GW1`, is an isothermal CO_2 storage simulation case meant to put a heavier load on the GPU. It uses an external CO_2 database [22] with sampled characteristics for density, viscosity, enthalpy, and fugacity to increase the accuracy of the simulation. The domain is a 100x100x10 grid with the injector located at the southwest corner, as shown in Figure 7.

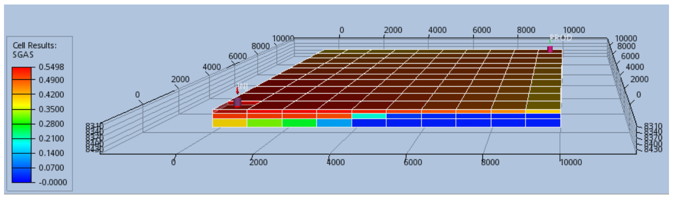

Fig. 6. The `SPE1` output at the end of its simulation. The figure shows the cell percentage that consists of the gas component. Generated with ResInsight.

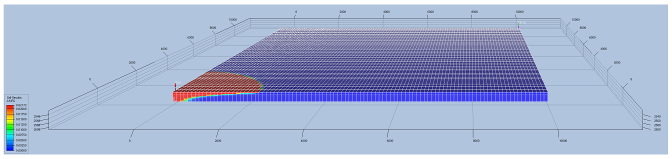

Fig. 7. Aqueous mole fraction of CO_2 at the end of the simulation of the `GW1` test case. Generated with ResInsight.

The main specifications of our machine are listed in Table 2. The simulations were run with one and four MPI processes. We chose one due to the issues presented in the Introduction and our initial testing showing a massive slowdown when launching the simulation with acceleration and multiple MPI processes. The execution time of `SPE1` went from roughly 5.2 seconds with one MPI process to approximately 9.5 hours and 5400 steps with two MPI processes. We used four MPI processes as an arbitrary power of two that was strictly less than the number of physical cores on our test system. Scalability studies such as Hammond et al. [23] dive deeper into the multi-CPU performance of PFLOTRAN, while our research explores the impact of utilising a GPU.

5 Results

The input deck for both `SPE1` and `GW1` contains no random variables for the geophysical properties, meaning the outputs of the simulations produce the same geophysical result each run within roundoff errors. To verify the correctness between the different versions, we compared the tables containing the solution at set time steps from the output files.

Table 2. Specifications of the system used for simulation.

Operating System	Ubuntu 20.04.5 LTS
CPU	Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz
GPU	NVIDIA GeForce RTX 2080 Ti
NVIDIA Driver Version	510.85.02

The performance results reported in the following tables are based on the arithmetic mean taken over five separate runs, with rounded decimals, to ensure a reliable representation. This will reduce the impact of varying underlying conditions, such as the state of caches or interleaved system tasks. The performance measurement is split in two: the efficiency of the different preconditioners is presented in Table 4, and the execution time of the whole simulation is presented in Table 3. The columns in the performance of the preconditioner are as follows: *Steps* refers to the total number of timesteps, Δt, required to complete the simulation within the constraints of the input deck. *Newton Iterations* represents the total number of Newton iterations required to solve the system. *Linear Iterations* is the total number of iterations it took to approximate the inverse Jacobian for all Newton iterations. A *cut* happens when the time step is halved due to non-convergence with NR.

Table 3. Execution time of the benchmarks.

	Time, n=1 [s]	**Time, n=4 [s]**	**Run**
	1.8119	5.9724	original
	1.7728	1.0103	orig-P3.17
SPE1	1.6451	4.5809	ilu-bjacobi
	1.7126	1.4596	gpu(CPU)
	5.1476	Not Run	gpu(GPU)
	392.62	155.59	original
	384.90	144.71	orig-P3.17
GW1	2710.2	1598.9	ilu-bjacobi
	2763.2	1440.6	gpu(CPU)
	1825.0	Not Run	gpu(GPU)

Note that from Table 3, only the versions with updated PETSc software reduced their execution time with increased processing power on the `SPE1` benchmark. The execution time was similar across the versions, except for the accelerated run, which needed three times as long as the others. The more complex `GW1` benchmark, however, shows the impact of using an efficient preconditioner. The CPR setup was almost seven times faster than the ILU-based setup and nearly 4.6 times faster than utilising the multi-threaded GPU with one MPI process. Note that leveraging the GPU in this test case outperforms the same preconditioner setup that only utilised the CPU.

Our results in Table 4 show that CPR required about a third of the iterations of ILU and Block Jacobi to solve the linear system while completing the simulation within a similar time frame (Table 3) on the `SPE1` benchmark. CPR is, therefore, the least likely to hit the max iteration constraint, presumably resulting in fewer cuts on more extensive simulations. The `GW1` benchmark further

Table 4. Preconditioner metrics for the benchmarks. Columns with multiple preconditioners show the results as ILU/Block Jacobi.

	Steps	**Newton Iterations**	**Linear Iterations**	**Cuts**	**Run**
	131	494	2748	0	original
	131	494	2748	0	orig-P3.17
SPE1	131	494	7995/9069	0	ilu-bjacobi
	131	494	7995/9069	0	gpu(CPU)
	131	494	7995	0	gpu(GPU)
	42	256	1663	0	original
	42	256	1663	0	orig-P3.17
GW1	83/90	882/944	70368/75332	2	ilu-bjacobi
	83/88	882/921	70368/73492	2	gpu(CPU)
	83	882	70368	2	gpu(GPU)

enhances the performance gap between CPR, ILU, and Block Jacobi for the `GW1` test case, where CPR needs about half the number of steps to complete the simulation. Both ILU and Block Jacobi had two cuts, indicating non-convergence with NR, possibly due to the linear solver not completing. Note the two-step improvement when running Block Jacobi on the newer PETSc software.

We can see how the different kernels on the *gpu* version performed by running the profilers Nvidia Nsight Systems and Nsight Compute. Results in Table 5 and 6 and Fig. 8 and 9 are based on output from these profilers. The CUDA kernels had mostly identical performance patterns, showing signs of branch divergence and memory stalls. The test cases `GW1` and `SPE1` showed the same general behaviour, though the grid on `SPE1` was too small to fill the available resources on the device.

Note that from Table 6, only 72 seconds, out of the total 1825 seconds of execution, was spent doing compute, which means that only about 3.9% of the total execution time was utilising the GPU. This indicates that very little work is being sent to the GPU compared to its computational power or that memory constraints and dependencies are stalling the required work.

Fig. 8 shows that each warp of the `axpy` kernel is being stalled for 46.8 cycles waiting for a data dependency, as denoted by *Stall Long Scoreboard*. In addition, `axpy` requires about 57 cycles to issue a single instruction and 58 instructions to complete an instruction. The `dot_kernel` required about twice as many cycles, requiring 107–113 warp cycles per issued instruction.

Furthermore, Fig. 9 shows that the memory bandwidth is highly utilised while the compute capabilities are mostly idling, indicating that the kernel is memory-bound. The kernel only used double-precision (FP64) with a peak performance of 11% and did not utilise the single-precision (FP32) pipeline at all.

Table 5. The first few entries of the CUDA API calls from the `GW1` test case. The leftmost column shows the time in percentage. *Total Time*, *Avg*, and *StdDev* shows the time in nanoseconds.

Time	Total Time	Instances	Avg	StdDev	Name
34.8	72,667,956,378	3,001,783	24,208.3	19,449.1	`cudaMemcpyAsync`
20.7	43,250,746,221	5,848,861	7,394.7	9,414.9	`cudaEventSynchronize`
20.7	43,154,003,436	220,758	195,481.0	27,111.6	`cudaMemcpy`
12.7	26,556,879,803	8,837,573	3,006.1	4,810.3	`cudaLaunchKernel`
7.4	15,517,363,763	14,620,734	1,061.3	4,616.4	`cudaEventRecord`
1.4	2,859,605,060	2,923,995	978.0	528.2	`cudaStreamSynchronize`
1.1	2,308,729,302	8,769,036	263.3	183.9	`cudaStreamGetCaptureInfo`
0.7	1,553,648,018	2,923,012	531.5	267.1	`cudaEventQuery`
0.2	444,325,063	276	1,609,873.4	20,038,808.5	`cudaFree`
0.1	115,020,158	198	580,909.9	670,053.6	`cudaMallocHost`
0.0	84,228,075	198	425,394.3	397,110.0	`cudaFreeHost`
0.0	52,245,154	3	17,415,051.3	30,159,111.0	`cudaStreamCreateWithFlags`

Table 6. The first few entries of the kernels launched during the `GW1` test case. The leftmost column shows the time in percentage. *Total Time*, *Avg*, and *StdDev* shows the time in nanoseconds.

Time	Total Time	Instances	Avg	StdDev	Name
40.6	29,130,955,334	2,846,130	10,235.3	843.5	`dot_kernel`
32.4	23,282,160,176	2,918,262	7,978.1	625.5	`axpy_kernel_val`
16.9	12,113,501,445	2,846,130	4,256.1	361.4	`reduce_1Block_kernel`
7.2	5,192,775,712	74,066	70,110.1	1,092.4	`nrm2_kernel`
2.1	1,514,058,114	74,066	20,442.0	501.5	`nrm2_kernel`
0.7	494,398,173	72,217	6,846.0	534.9	`scal_kernel_val`
0.0	29,950,337	2,816	10,635.8	736.1	`iamax_kernel`

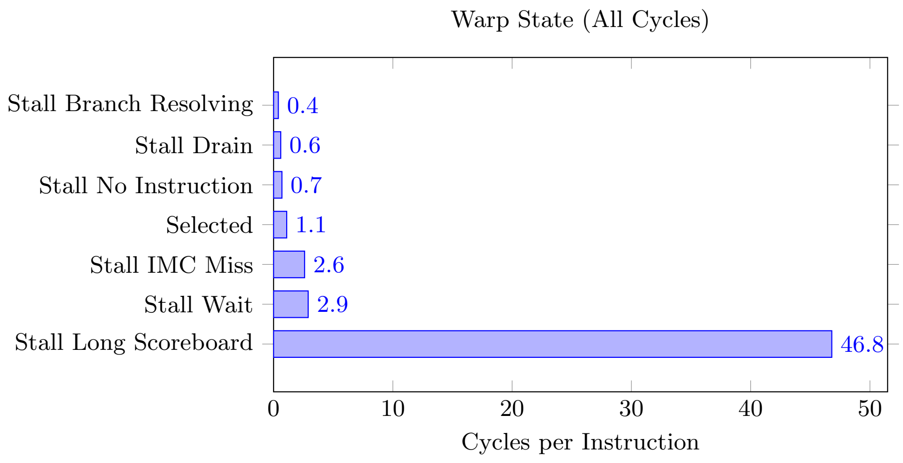

Fig. 8. Warp statistics for the `axpy` kernel from the `GW1` test case. The warp states describe a warp's readiness or inability to issue its next instruction. The chart shows the average number of cycles spent in that state per issued instruction.

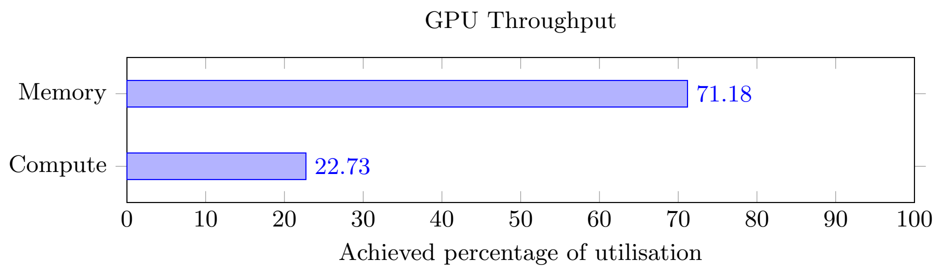

Fig. 9. Scheduler statistics for the `axpy` kernel from the `GW1` test case.

We note that the work done on the GPU is relatively low. This can be traced back to the operations done on the GPU, which mainly consisted of simple vector operations. Small amounts of data were sent to the GPU, computed, and returned. This illustrates our main issue with our acceleration: the data is not staying on the device, and only a portion is being moved to the GPU for computing.

The data flow is not available for the threads partly because only the solver was accelerated. The TOWG module that constructs the Jacobian was still on the CPU and is part of PFLOTRAN-OGS, not PETSc. PFLOTRAN-OGS passes PETSc-pointers to the TOWG functions that tell it how to construct the Jacobian. When SNES needs to solve a linear system involving the Jacobian, it calls the TOWG module to construct it. It then uses the given preconditioner and solver to solve the system. This means we have a construction and preconditioning step on the CPU while the solver is run on the GPU. This required the Jacobian to be copied to the GPU at every iteration.

6 Conclusion

The physical complexity and non-linear nature of multi-phase flow simulations, including modelling of CO_2, demand a lot of computation power.

In this study, we explored GPU acceleration of the production application PFLOTRAN-OGS by utilising GPU-accelerated functions provided by recent versions of PETSc. The successful integration of these complex codes was tested using the `SPE1` and `GW1` benchmarks, where the latter is also a smaller test case but uses more relevant numerical equations for CO_2 simulations.

Our results showed that only about 72 seconds, out of 1825, were spent doing computing on the `GW1` benchmark, indicating the GPU was mostly idle. To alleviate this problem, more of the computations should be moved to the GPU, for example, the construction of the Jacobian matrix. OpenGoSim has since implemented this in the upcoming releases of PFLOTRAN-OGS. Our profiled GPU kernels exhibited a high degree of long scoreboard stalls. This stemmed from unresolved data dependencies from the memory system. Since the simulation proceeded on both the CPU and the GPU, the data is frequently moved between them.

To improve the implementation, it would be useful to look into optimising the use of the memory hierarchy and utilising more advanced features of the CUDA Programming Model to improve memory characteristics. Finally, since PetscSF, the communication module in PETSc is exploring NVIDIAs' NVSHMEM as a solution to add stream support, this could also be worth further study.

Overall, our work showed that GPUs hold much promise for speeding up complex multi-phase flow simulations and similar codes. We also illustrated some challenges of accelerating only parts of the iterative steps that depend on other numerical methods. Various storage formats for the associated matrices used by the solvers could also be further explored.

Acknowledgements

Thanks are due to the people at OpenGoSim, especially Dave Ponting and Daniel Stone, for their invaluable assistance with understanding the code and answering questions during this study. Many thanks to Prof. Philip S. Ringrose for his insight and assistance with the geophysical aspect. We would also like to acknowledge NTNU, including Dept. of Computer Science, for supporting the HPC-Lab equipment used in our experiments. This work was partially supported by the SFI Centre for Geophysical Forecasting under RCN grant 309960.

References

[1] Denise Watts. *Reservoir simulation models in production forecasting.* Accessed: 2023-05-26. 2016. URL: `https://petrowiki.spe.org/Reservoir_simulation_models_in_production_forecasting`.

[2] Atgeirr Flø Rasmussen. *OPM Flow in MSO4SC and other stories.* Accessed: 2023-05-26. 2017. URL: `https://opm-project.org/wp-content/uploads/2018/01/MSO4SC-OPM-slides-updated.pdf`.

[3] Manoj Kumar, Akhilesh Kumar Singh and Akanksha Srivastava. "Various Newton-type iterative methods for solving nonlinear equations". In: *Journal of the Egyptian mathematical society* 21.3 (2013), pp. 334–339.

[4] Heeho D Park et al. "Linear and nonlinear solvers for simulating multiphase flow within large-scale engineered subsurface systems". In: *Advances in Water Resources* 156 (2021), p. 104029.

[5] Jack Dongarra et al. "MPI-a message-passing interface standard". In: *International Journal of Supercomputer Applications and High Performance Computing* 8.3-4 (1994), p. 165.

[6] Junchao Zhang et al. "The PetscSF Scalable Communication Layer". In: *IEEE Transactions on Parallel and Distributed Systems* 33.4 (2022), pp. 842–853.

[7] Richard Tran Mills et al. "Toward performance-portable PETSc for GPU-based exascale systems". In: *Parallel Computing* 108 (2021), p. 102831.

[8] PETSC Development Team. *Frequently Asked Questions*. Accessed: 2022-09-08. 2022. URL: `https://www.pflotran.org/documentation/user_guide/how_to/faq.html`.

[9] Glenn Edward Hammond. *PFLOTRAN: Practical Application of High Performance Computing to Subsurface Simulation*. Tech. rep. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2016.

[10] Rajeev Das. "GPUs in subsurface simulation: an investigation". In: *Engineering with Computers* 33.4 (2017), pp. 919–934.

[11] Satish Balay et al. *PETSc Web page*. `https://petsc.org/`. 2023. URL: `https://petsc.org/`.

[12] David Ponting. *e-mail*. Private Communication. 2022.

[13] John A Trangenstein and John B Bell. "Mathematical structure of the black-oil model for petroleum reservoir simulation". In: *SIAM Journal on Applied Mathematics* 49.3 (1989), pp. 749–783.

[14] OpenGoSim. *The Mathematical Formulation of the Black Oil Model*. Accessed: 2023-05-26. 2023. URL: `https://docs.opengosim.com/theory/mathematical_formulation_of_black_oil/`.

[15] OpenGoSim. *The Mathematical Formulation of GAS_WATER*. Accessed: 2022-08-09. 2021. URL: `https://docs.opengosim.com/theory/mathematical_formulation_of_gw/`.

[16] Auli Niemi, Jacob Bear, Jacob Bensabat et al. *Geological storage of CO2 in deep saline formations*. Vol. 29. Springer, 2017.

[17] Temitope Ajayi, Jorge Salgado Gomes and Achinta Bera. "A review of CO2 storage in geological formations emphasizing modeling, monitoring and capacity estimation approaches". In: *Petroleum Science* 16.5 (2019), pp. 1028–1063.

[18] PFLOTRAN. *Method of Solution*. Accessed: 2023-05-29. 2022. URL: `https://www.pflotran.org/documentation/theory_guide/appendixB.html`.

[19] Peter C. Lichtner et al. *PFLOTRAN Web page*. Accessed: 2022-08-04. 2020. URL: `http://www.pflotran.org`.

[20] Peter C. Lichtner et al. *PFLOTRAN User Manual*. Tech. rep. Accessed: 2022-08-04. 2020. URL: `http://documentation.pflotran.org`.

[21] Aziz S Odeh. "Comparison of solutions to a three-dimensional black-oil reservoir simulation problem (includes associated paper 9741)". In: *Journal of Petroleum Technology* 33.01 (1981), pp. 13–25.

[22] OpenGoSim. *CO2 Database*. Accessed: 2023-10-25. 2023. URL: `https://bit.ly/477aXps`.

[23] Glenn E Hammond, Peter C Lichtner and RT Mills. "Evaluating the performance of parallel subsurface simulators: An illustrative example with PFLOTRAN". In: *Water resources research* 50.1 (2014), pp. 208–228.