The Journal of Open Source Software

gpu-ISTL - Extending OPM Flow with GPU Linear Solvers

Kjetil Olsen Lye [1], **Tobias Meyer Andersen** [1], **Atgeirr Flø Rasmussen** [1], **and Jakob Torben** [1]

1 Mathematics and Cybernetics, SINTEF Digital, Oslo, Norway

DOI: 10.21105/joss.07740

Software

- Review
- Repository
- Archive

Editor: Prashant Jha

Reviewers:

- @pratikvn
- @berenger-eu

Submitted: 21 October 2024
Published: 08 May 2025

Summary

The gpu-ISTL framework provides Dune-ISTL compatible sparse linear operations on the GPU for the OPM Flow simulator (Rasmussen et al., 2021). OPM Flow is an open-source fully implicit simulator for subsurface reservoir simulation of industrial complexity, where efficient linear solvers are critical for computational performance. It is written in C++ and relies on the Distributed and Unified Numerics Environment (Dune) (Bastian et al., 2021) for several numerical algorithms. In particular, templated linear-solver algorithms provided by the Dune Iterative Solver Template Library (Dune-ISTL) (Blatt & Bastian, 2007) are essential in the simulator. To GPU-accelerate the simulator with minimal effort, gpu-ISTL provides classes that can instantiate Dune-ISTL algorithms with types that automatically runs the algorithms provided on the GPU. Both AMD and Nvidia GPUs are supported in the same source code by utilizing hipify-perl (Advanced Micro Devices, Inc., 2024b) to generate HIP code (Advanced Micro Devices, Inc., 2024a) from the existing CUDA code (NVIDIA et al., 2023) when compiling OPM Flow.

Statement of Need

Simulating multiphase and multicomponent fluid behavior within complex geological formations is crucial for modern geoenergy operations. The repeated solution of large, sparse systems of linear equations is typically the most computationally expensive part of these simulations. Therefore, it is essential for a reservoir simulator like OPM Flow, which is used both commercially and for research, to have flexible linear algebra libraries that optimally utilize contemporary computer hardware. To fill this need, we introduce gpu-ISTL, a generic Dune-ISTL compatible GPU-accelerated linear solver library and employ it within OPM Flow.

A current and prominent trend in computer architecture is hardware specialization (Hennessy & Patterson, 2017), where modern computational units are increasingly tailored to specific workloads. Having to write tailored implementations of each numerical method for every possible hardware provider is generally undesirable, especially in a large codebase like OPM Flow, which exceeds half a million lines of code. The gpu-ISTL library leverages the parallel computational resources of GPUs to meet industry performance demands. It also facilitates further research into the development of efficient GPU-accelerated linear solvers, without requiring intrusive and substantial changes to the OPM Flow codebase.

Our new library enhances research in numerical methods for reservoir simulation by leveraging OPM Flow's extensive infrastructure for complex reservoirs. It enables investigations into GPU-based linear solvers, preconditioners, autotuning and mixed-precision computations, see for instance (Andersen et al., 2025). By integrating gpu-ISTL into the OPM Flow linear solver subsystem, development remains synchronized with the simulator's rapidly evolving codebase, ensuring compatibility and continuity in development efforts.

Several standalone libraries GPU-accelerate linear algebra operations. Some also support at least both Nvidia and AMD cards, as well as the iterative solvers required by OPM Flow. Examples of open-source libraries fulfilling all these criteria are `ViennaCL` (Rupp et al., 2016) and `PETSc` (Mills et al., 2021), though those are not `Dune-ISTL` compatible. The Bandicoot library (Curtin et al., 2023), an extension to the Armadillo (Sanderson & Curtin, 2019) library, is another worthy mention of a GPU library that effectively computes linear algebra operations, although it does not feature the iterative solvers that are vital to OPM.

The `gpu-ISTL` Components

The `gpu-ISTL` library encompasses numerical algorithms, Dune adapters, and essential linear algebra components such as `GpuVector` and `GpuSparseMatrix` that leverage optimized libraries from Nvidia and AMD for efficient mathematical operations. Preconditioners such as Jacobi, ILU(0), and DILU are implemented with custom GPU kernels to enhance the performance of the bi-conjugate gradient stabilized (BiCGSTAB) method within `gpu-ISTL`. Adapter classes like `SolverAdapter` and `PreconditionerAdapter` allow mixing GPU and CPU solvers or preconditioners for ease of development, testing and validation. Moreover, `GpuOwnerOverlapCopy` extends the MPI functionality in Dune to support multi-GPU simulations, including CUDA-aware MPI for Nvidia cards to accelerate inter-process memory transfers. The library also provides capabilities for autotuning thread-block sizes in user-implemented GPU kernels. Figure 1 contains a simplified class diagram with of some of the components that constitute `gpu-ISTL`.

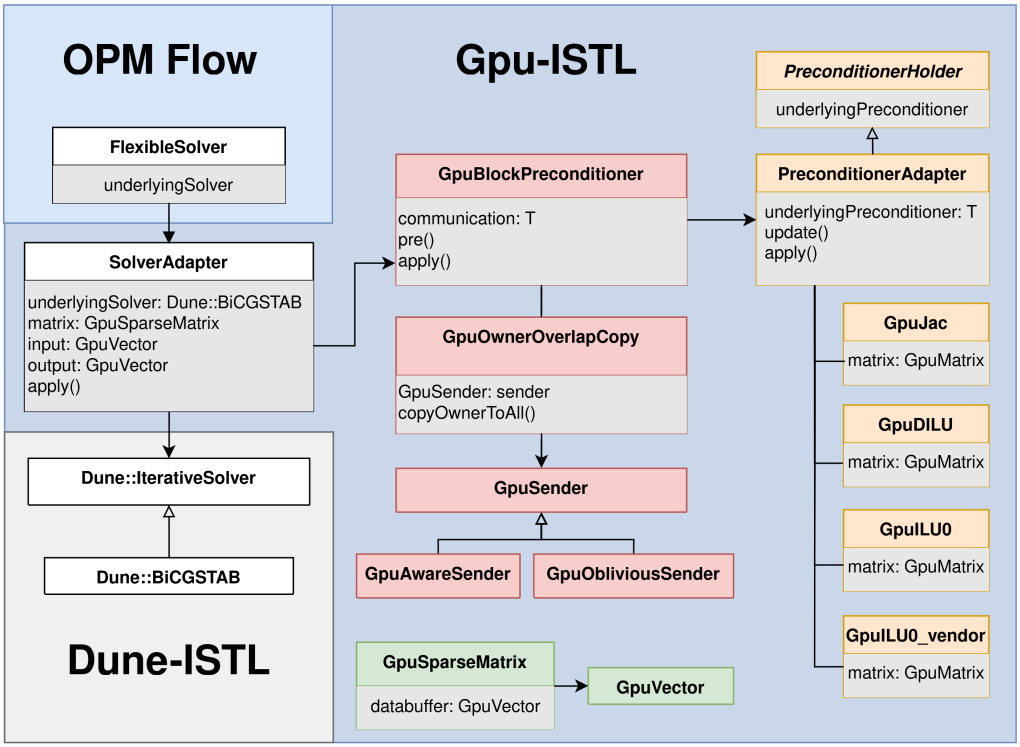

Figure 1: Class diagram showing a simplified view of how `gpu-ISTL` is implemented. The colored backgrounds indicate which namespace the classes belong to. Colors on individual classes correspond to what conceptual part of the simulator they are a part of. White represents linear solvers, red represents the classes enabling multiprocess simulations, orange represents the implementation linear solver preconditioners, and green represents general linear algebra functionality.

Furthermore, Figure 2 provides a sequence diagram for the call stack of a single linear solve in OPM Flow using the gpu-ISTL framework. We note that the sequence diagram is for the MPI

free version. In Figure 2 we denote the (indirect) calls to specific GPU kernels as an arrow from the corresponding gpu-ISTL classes to the right-hand red box representing the GPU.

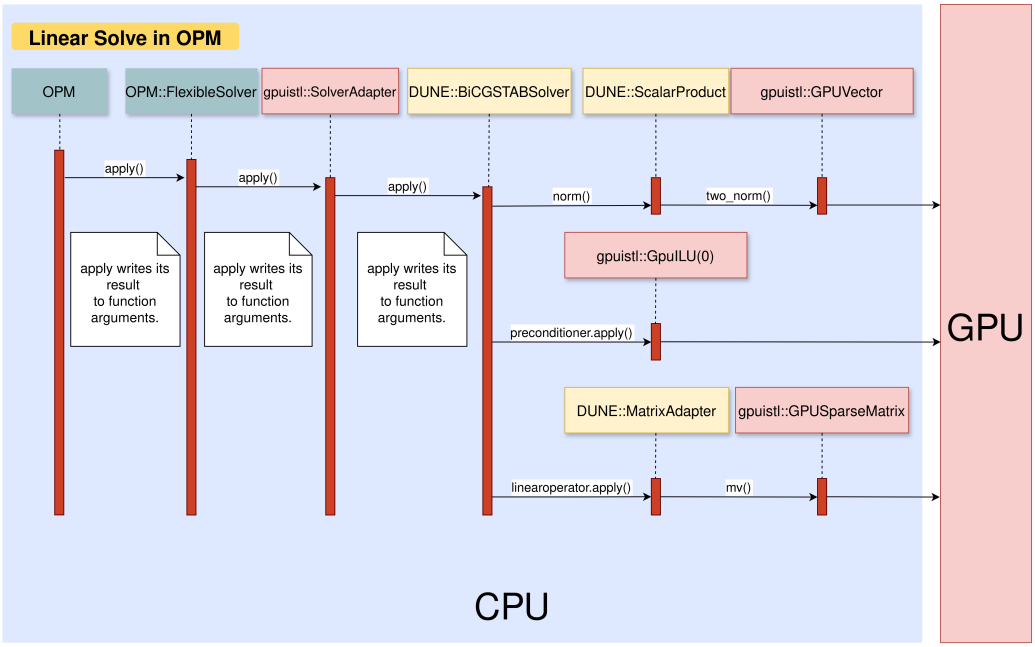

Figure 2: A sequence diagram for a single linear solver in OPM Flow using the gpu-ISTL framework. For readability, components from OPM are marked in green, components from DUNE in yellow and components from gpu-ISTL are marked in red. The classes GpuVector, GpuSparseMatrix and GpuILU0 of gpu-ISTL will (through some indirection) call GPU kernels, which is visualized as a big red block on the right-hand side of the diagram.

OPM Flow's build system utilizes hipify-perl (Advanced Micro Devices, Inc., 2024b), which translates CUDA (NVIDIA et al., 2023) code to HIP (Advanced Micro Devices, Inc., 2024a) code if one wants to compile for AMD GPUs. Incorporating the translation in the build system ensures that the hardware of the two largest GPU vendors are supported without incurring any extra overhead for the developers of the CUDA code in the repository.

Performance Case Study of CO_2 Storage Simulation

To demonstrate the effectiveness of gpu-ISTL, we conduct a scaling study on geological carbon storage cases derived from the 11th Society of Petroleum Engineers (SPE) Comparative Solutions Project (CSP) (Nordbotten et al., 2024). Specifically, we simulate Case C from the SPE11 CSP using Pyopmspe11 (Landa-Marbán & Sandve, 2025) with successively finer discretizations.

We simulate using the ILU0 preconditioned BiCGSTAB as the linear solver on three different hardware configurations, tracking the time spent in the linear solver normalized by the number of linear iterations. Specifically, we utilize 16 cores of an AMD 7950X CPU for the first run, an AMD Radeon RX 7900XTX GPU for the second run, and an Nvidia RTX 4070Ti GPU for the third run. Figure 3 shows the speedup of the GPU compared to the normalized CPU performance. The results demonstrate that GPU runtimes scale better than CPU runtimes as problem sizes increase. This highlights the performance benefits of using the advanced linear solvers with preconditioners implemented in gpu-ISTL. Furthermore, it underscores how gpu-ISTL and OPM Flow can serve as a robust platform for exploring and testing novel preconditioners within an industry-relevant environment, offering rapid evaluation on both synthetic and real-life cases.

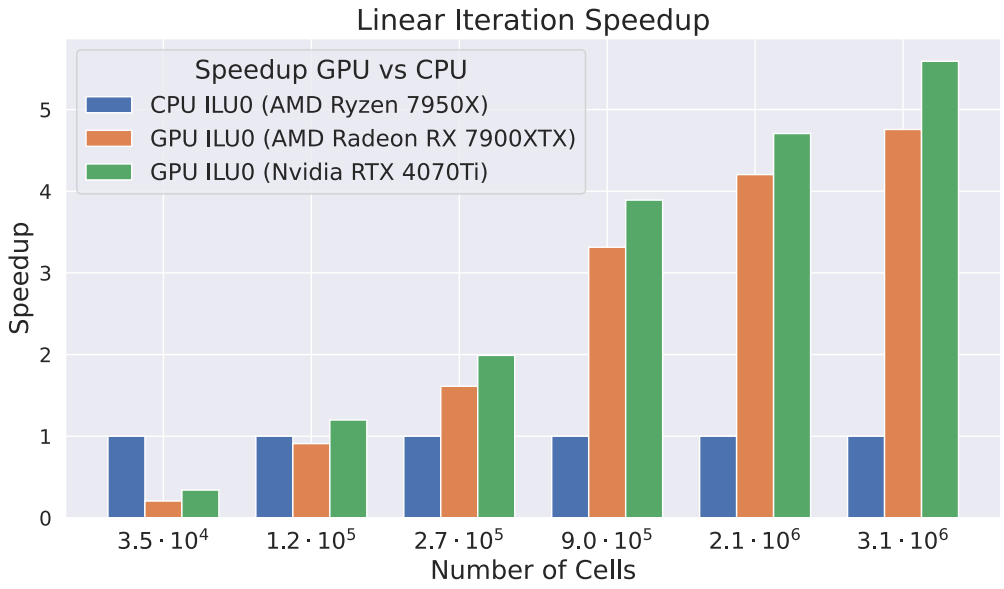

Figure 3: Speedup of the mean time per linear iteration across a 2000-year simulation case derived from Case C of the 11th SPE Comparative Solution Project compared to the CPU implementation. A speedup of 5.6 and 4.8 is achieved on the largest case for the RTX 4070Ti and Radeon RX 7900XTX respectively.

Case Study Configuration

The cases generated by Pyopmspe11 can be found in (Lye et al., 2025), in the ZIP-file "SPE11C_LONG.zip". The file names include the sidelength of the domain in terms of cells to distinguish between the cases. The total number of cells in the domain is then the cube of that number, although Pyopmspe11 will add some extra cells as padding on the domain.

To run OPM Flow simulations for the case study on CPU, we use the following command to run OPM Flow with 16 processes in parallel, where we specify that the linear solver is described in a JSON file called "cpu_ilu0.json". The "<S>" represents the sidelength in the file path and name.

```
mpirun -n 16 flow \
    /path/to/SPE11C_LONG/SPE11C<S>CUBE_LONG/deck/SPE11C<S>CUBE_LONG.DATA \
    --output-dir=/path/to/output \
    --linear-solver=/path/to/cpu_ilu0.json \
    --threads-per-process=1 \
    --newton-min-iterations=1 \
    --matrix-add-well-contributions=true
```

To recreate the CPU runs, use the following "cpu_ilu0.json".

```
{
    "tol": "0.01",
    "maxiter": "200",
    "verbosity": "0",
    "solver": "bicgstab",
    "preconditioner": {
        "type": "ILU0",
        "ilulevel": "0"
    }
}
```

We use the command below to run the GPU simulations, where we only use one process, because we run it on a single GPU. To save simulation time we use 16 threads in parallel, which does not affect the timing for the linear solver which exists entirely on the GPU. The "<S>" represents the sidelength in the file path and name.

```
mpirun -n 1 flow \
    /path/to/SPE11C_LONG/SPE11C<S>CUBE_LONG/deck/SPE11C<S>CUBE_LONG.DATA \
    --output-dir=/path/to/output \
    --linear-solver=/path/to/gpu_ilu0.json \
    --threads-per-process=16 \
    --newton-min-iterations=1 \
    --matrix-add-well-contributions=true
```

The "gpu_ilu0.json" file enables the use of `gpu-ISTL` by specifying that we want to use the `gpubicgstab` linear solver, and the `OPMGPUILU0` preconditioner.

```
{
    "tol": "0.01",
    "maxiter": "200",
    "verbosity": "0",
    "solver": "gpubicgstab",
    "preconditioner": {
        "type": "OPMGPUILU0",
        "ilulevel": "0"
    }
}
```

Acknowledgements

Development of `gpu-ISTL` in OPM Flow from 2021 to 2024 was part of the EU project *HPC, Big Data, and Artificial Intelligence convergent platform* (ACROSS). Development in 2024 has been financed by Equinor.

References

Advanced Micro Devices, Inc. (2024a). *Heterogeneous-computing interface for portability*. https://rocm.docs.amd.com/projects/HIP/en/latest/

Advanced Micro Devices, Inc. (2024b). *Hipify-perl*. https://rocmdocs.amd.com/projects/HIPIFY/en/latest/hipify-perl.html#hipify-perl

Andersen, T. M., Torben, J., Lye, K. O., Rasmussen, A. F., & Lie, K. A. (2025). *A comparison of DILU and ILU(0) as GPU-accelerated preconditioners: Vols. SPE Reservoir Simulation Conference* (p. D021S009R004). https://doi.org/10.2118/223873-MS

Bastian, P., Blatt, M., Dedner, A., Dreier, N.-A., Engwer, C., Fritze, R., Gräser, C., Grüninger, C., Kempf, D., Klöfkorn, R., Ohlberger, M., & Sander, O. (2021). The Dune framework: Basic concepts and recent developments. *Computers & Mathematics with Applications*, *81*, 75–112. https://doi.org/10.1016/j.camwa.2020.06.007

Blatt, M., & Bastian, P. (2007). The iterative solver template library. In B. Kagström, E. Elmroth, J. Dongarra, & J. Wasniewski (Eds.), *Applied parallel computing – state of the art in scientific computing* (pp. 666–675). Springer. https://doi.org/10.1007/978-3-540-75755-9_82

Curtin, R. R., Edel, M., & Sanderson, C. (2023). Bandicoot: C++ Library for GPU Linear Algebra and Scientific Computing. *arXiv:2308.03120v1*. http://arxiv.org/abs/2308.03120v1

Hennessy, J. L., & Patterson, D. A. (2017). *Computer architecture, sixth edition: A quantitative approach* (6th ed., pp. 5–6). Morgan Kaufmann Publishers Inc. ISBN: 0128119055

Landa-Marbán, D., & Sandve, T. H. (2025). pyopmspe11: A Python framework using OPM Flow for the SPE11 benchmark project. *Journal of Open Source Software*. https://doi.org/10.21105/joss.07357

Lye, K. O., Andersen, T. M., Torben, J., & Rasmussen, A. (2025). *Gpu-ISTL - extending OPM flow with GPU linear solvers* (Version 2024.10). Zenodo. https://doi.org/10.5281/zenodo.15259506

Mills, R. T., Adams, M. F., Balay, S., Brown, J., & Dener, A. (2021). Toward performance-portable PETSc for GPU-based exascale systems. *Parallel Computing*, *108*, 102831. https://doi.org/10.1016/j.parco.2021.102831

Nordbotten, J. M., Ferno, M. A., Flemisch, B., Kovscek, A. R., & Lie, K.-A. (2024). The 11th society of petroleum engineers comparative solution project: Problem definition. *SPE Journal*, *29*(05), 2507–2524. https://doi.org/10.2118/218015-pa

NVIDIA, Vingelmann, P., & Fitzek, F. H. P. (2023). *CUDA, release: 12.2.r12.2*. https://developer.nvidia.com/cuda-toolkit

Rasmussen, A. F., Sandve, T. H., Bao, K., Lauser, A., Hove, J., Skaflestad, B., Klöfkorn, R., Blatt, M., Rustad, A. B., Sævareid, O., Lie, K.-A., & Thune, A. (2021). The Open Porous Media Flow reservoir simulator. *Computers & Mathematics with Applications*, *81*, 159–185. https://doi.org/10.1016/j.camwa.2020.05.014

Rupp, K., Tillet, P., Rudolf, F., Weinbub, J., Morhammer, A., Grasser, T., Jüngel, A., & Selberherr, S. (2016). ViennaCL—linear algebra library for multi- and many-core architectures. *SIAM Journal on Scientific Computing*, *38*(5), S412–S439. https://doi.org/10.1137/15M1026419

Sanderson, C., & Curtin, R. (2019). Practical sparse matrices in c++ with hybrid storage and template-based expression optimisation. *Mathematical and Computational Applications*, *24*(3), 70. https://doi.org/10.3390/mca24030070