Dyablo : A simulation code for astrophysics fluids with adaptive mesh refinement in the exascale era

Maxime Delorme, Arnaud Durocher, Allan Sacha Brun, Adam J. Finley, Olivier Marchal, Dominique Aubert

Dyablo : A simulation code for astrophysics fluids with adaptive mesh refinement in the exascale era

Journal of Physics: Conference Series **2997** (2025) 012014 doi:10.1088/1742-6596/2997/1/012014

Dyablo : A simulation code for astrophysics fluids with adaptive mesh refinement in the exascale era

Maxime Delorme[1], Arnaud Durocher[2], Allan Sacha Brun[2], Adam J. Finley[2], Olivier Marchal[3], Dominique Aubert[3]

[1]IRFU, CEA, Université Paris-Saclay, F-91191 Gif-Sur-Yvette, France
[2]Université Paris-Diderot, AIM, Sorbonne Paris Cité, CEA, CNRS, F-91191 Gif-Sur-Yvette, France
[3]Observatoire Astronomique de Strasbourg, Université de Strasbourg, CNRS UMR 7550, 11 rue de l'Université, 67000 Strasbourg, France

E-mail: maxime.delorme@cea.fr

Abstract. Dyablo is a new code for the simulation of astrophysical plasmas on exascale architectures using Adaptive Mesh Refinement. The code targets new and upcoming Exascale supercomputers with a new codebase in C++ using the Kokkos portability performance library. Kokkos allows us to write a unique code to target multiple hardware architectures including GPUs. Dyablo is based on modern software engineering principles to allow "separation of concerns" and collaboration between HPC engineers and physicists through abstract interfaces and modularity. Although Dyablo is still in development, the code is now deployable and usable. In this communication, we present the code, its philosophy, three example runs, and scalability results.

1 Introduction

With the upcoming launch of the German and French exascale supercomputers, European scientists have been developing and migrating HPC codes to fully support graphics processing units (GPUs). Due to the variety of constructors (Nvidia, AMD, Intel) and programming models (CUDA, HIP, SYCL), new codes tend to rely on performance portability libraries to outsource the specific tasks of memory allocation and parallel dispatch. Such libraries allow developers and physicists to implement computing kernels in an agnostic way that will be compiled and optimized on different backends for CPUs or GPUs. At the forefront of these libraries, Kokkos [10] [4] has now become a very relevant choice for the future exascale codes. In 2024, Kokkos became part of the newly founded High Performance Software Foundation[1], a linux foundation dedicated to maintaining an open-source high-performance ecosystem on contemporary and future supercomputing architectures.

For a few years now, European teams have been benefitting from portability performance libraries to develop new codes in astrophysics. In France, the Idefix code [8] has been the first to use Kokkos for astrophysics and to have reached a production stage for the simulation of MHD astrophysical fluids on fixed grids. Some simulations, however, require large ranges in temporal and spatial scales which cannot be reached with fixed grids. The use of adaptive mesh refinement (AMR) is thus a very promising solution. However, parallelism of AMR on GPU is a notoriously difficult problem due to the non-regular memory access and potentially heavy data structures.

In this communication, we present Dyablo, a code developed at CEA for the simulation of astrophysical fluids on AMR grids using Kokkos. The next section presents the key-concepts behind Dyablo, while section 3 presents two sample simulations and scalability plots.

[1] `https://hpsf.io`

2 The Dyablo Code

Dyablo is a finite-volume code aimed at leveraging performance from modern hardware and modern concepts in software engineering. The code is written in C++(17) and relies on Kokkos for intra-node performance portability and MPI for inter-node parallelism.

At its core, Dyablo relies on three pillars: separation of concerns, modularity and abstraction. *Separation of concerns* dictates that the code should be compartmentalized in such a way that physicists and mathematicians can implement modules without being able to impact the high-performance sections of the code. On the other hand, computer scientists should be able to change the various backends used by the code (e.g. the way the AMR is managed) without having an impact on the physics/mathematics side of the code. The second pillar, *modularity*, implies that the code should be versatile enough so that we can activate or deactivate certain treatments without having to recompile the code and that the implementation of new physics models, solvers, or backends should be made easy by the code. Finally, *abstraction* is a way to provide the users high-level methods that will hide the complexity of the underlying data models, to allow mathematicians and physicists to have a code that is closer to their field of expertise.

These three pillars leads to the key concept of plugins in Dyablo. A plugin is an interchangeable piece of code that can be parametrized at runtime. Each aspect of Dyablo is a plugin: hyperbolic and parabolic solvers, time-step calculation, initialization, refinement or IOs are examples of different types of plugins in Dyablo. This allows us to implement a variety of solutions for each plugin, and let the user choose the combination at runtime without needing to recompile the code.

In order to streamline as much as possible calculations on GPU and avoid superfluous transfers and memory usage, we use a linear octree (see e.g. [3]) stored on device for the AMR tree. Dyablo avoids having too many non-conformal interfaces by implementing block-based AMR where each entry of the linear octree stores a regular Cartesian subgrid. Emphasis has been made on providing high-level abstractions to access the AMR grid. Hence, tree-traversal, neighbour finding and memory accessing are done via high-level methods.

The AMR backend itself is modular and Dyablo integrates at the moment its own AMR solution and the possibility to use the PABLO[2] AMR library. It is not excluded that more backends will be supported in the future.

As of today, Dyablo is able to solve a range of physical problems. The base of the code uses the Euler equations for hydrodynamics, magnetohydrodynamics or radiative hydrodynamics. These equations can be supplemented with additional parabolic term for the viscosity and the thermal conduction, as well as source terms for the radiation/matter coupling.

Finally, all developments are made on a internal CEA gitlab with a custom continuous integration system that tests and validates each merge request on the code. A public mirror of the current development branch is pushed on github[3] and is widely accessible.

3 Simulations and Scaling

Simulations with Dyablo are now running on all French major super computers, including the Irene, Jean-Zay and Adastra computers, supporting both their CPU and GPU partitions. We now present three sample simulations as well as some scalability results.

3.1 Radiative cosmology

A first simulation has been done in the context of the GINEA [1] research group and can be seen on Fig 1. This run simulates the evolution of matter in a ΛCDM universe with radiation. Dyablo solves the hydrodynamics equations, the Poisson equation for gravity using a preconditioned conjugate gradient solver, as well as radiation using an explicit M1 solver [2]. The code is evolved using a SSP-RK2 scheme[7], with the piecewise linear method and a minmod slope limiter for reconstruction. Interfaces fluxes are retrieved using a HLLC Riemann solver [9] for the hydro part and a Lax-Friedrichs solver for the radiative part [2]. The simulation has been computed on the AMD Rome CPU partition of the Irene supercomputer on 16 MPI ranks using the OpenMP backend of Kokkos. The simulation is started with a base resolution of 512^3 grid cells and 512^3 particles for a $128 Mpc/h$ comoving boxsize. The run allows for 3 active refinement levels, pushing the maximum resolution to 2048^3. The refinement is triggered on the total mass inside a cell. The box is periodic along each direction.

[2] `https://optimad.github.io/bitpit/modules/overview.html`

[3] `https://github.com/Dyablo-HPC/Dyablo`

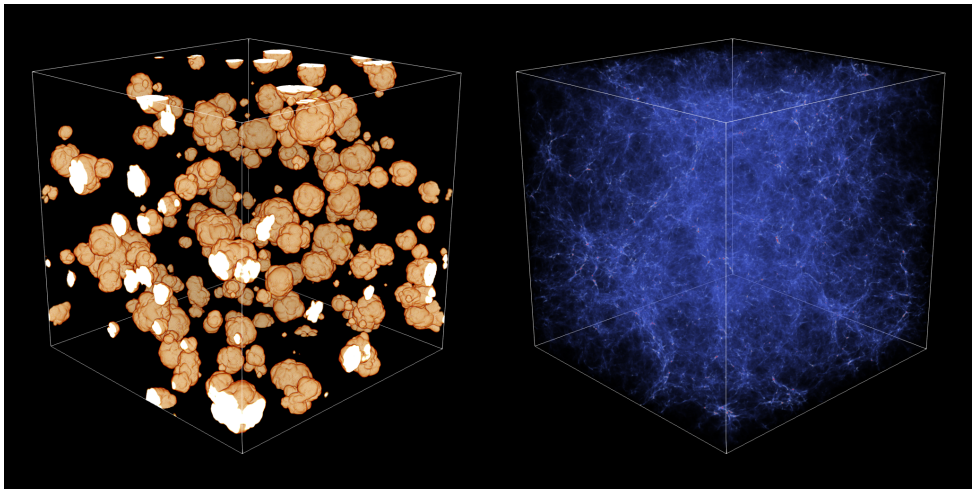

Figure 1: Radiative cosmological simulation at half reionization. Refinement is triggered on the mass of the cells. **Left**: The ionization fraction is represented by the orange bubbles. **Right**: The distribution of dark matter particles in the box.

3.2 Standalone solar convection

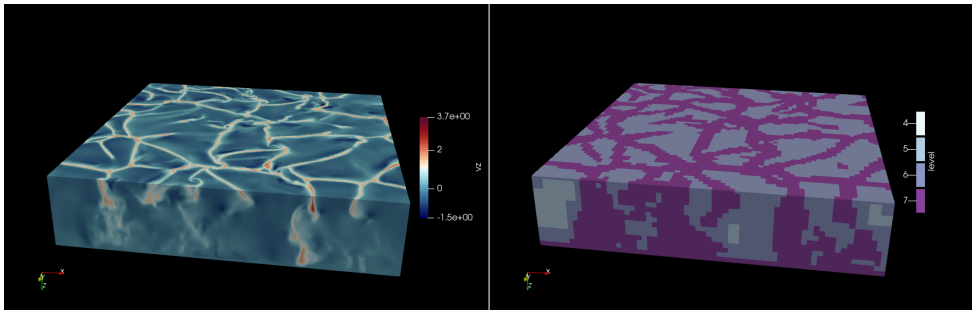

Figure 2: Solar convection simulation. **Left**: Vertical velocity. We see in blue the hot matter rising through the box and in red the rapid downflows of cold matter. **Right**: Refinement level of the box. The low-resolution regions are in blue while high-resolution are in purple. The refinement is triggered on the strong downflows, allowing the precise resolution of the edges of the convection cells.

The second simulation we present is one of solar convection and can be seen on Fig 2. We initialize a Cartesian slab in hydrostatic equilibrium using a polytropic profile akin to the one defined in [5]. The precise setup can be found in [6]. In this simulation, Dyablo solves the hydrodynamics equations with constant in time and space gravity and explicit thermal conduction and viscosity. For this case, the hydro equations are evolved also using a SSP-RK2 timestepping with linear reconstruction and a minmod slope limiter. The interface fluxes are calculated using the HLLC solver. Parabolic terms are discretized as a cell-boundary flux are evolved explicitly using a first order euler timestepping scheme. The simulation ran on four Nvidia a100 on the Storm cluster at CEA. The initial resolution is set to $64 \times 64 \times 16$ and allows for 4 active refinement levels, meaning the maximum resolution would reach $1024 \times 1024 \times 256$. Boundary conditions are periodic horizontally and impenetrable, stress-free walls in hydrostatic equilibrium for the vertical boundaries.

3.3 Triple layer solar convection

The third and final simulation we present is one of solar convection in a triple layer configuration and can be seen in Fig 3. In this simulation, we initialize a Cartesian box with three polytropic profiles (as in the previous setup). The equations solved and boundary conditions are exactly the same. The top and

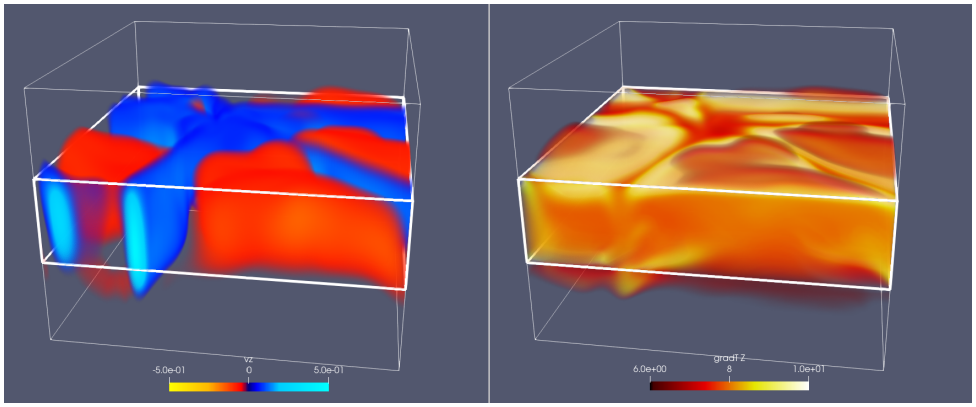

Figure 3: Solar convection simulation. **Left**: Vertical velocity. We see in blue the hot matter rising through the box and in red the rapid downflows of cold matter. **Right**: Refinement level of the box. The low-resolution regions are in blue while high-resolution are in purple. The refinement is triggered on the strong downflows, allowing the precise resolution of the edges of the convection cells.

bottom layers are initialized with a polytropic index $m1 = m3 = 3 > 3/2$, and are hence made stable, while the central layer is made convectively unstable with $m2 = 1$. The thermal diffusion coefficient is fixed by layer to ensure that initially, the vertical thermal flux is constant over the box. In this setup, convection quickly develops as with the previous run. However, the triple layer setup allows the convective plumes to overshoot and to penetrate above and below the convective zone exciting gravity waves in these layers. This run is done in fixed-grid mode with 128 points horizontally and 256 vertically and was run on two Nvidia a100 on the Storm cluster at CEA.

3.4 Weak-Scaling

To evaluate scaling of Dyablo on super-computers and in particular on GPUs we use the solar-convection setup described in the previous section. Knowing the box is periodic along the horizontal direction we build a weak-scaling test where we take a single realization of a convection run, and we tile it horizontally having one domain per MPI process. This allows us to grow the size of the problem perfectly with the number of processes which is perfectly suited for load-balancing. To evaluate the performance of the computation kernels we deactivate IOs. Since all domains are tiled, the box is perfectly load-balanced at all times, and thus we also deactivate the well-balancing process. We proceed by having Dyablo compute a hundred iterations, with an AMR cycle every iteration.

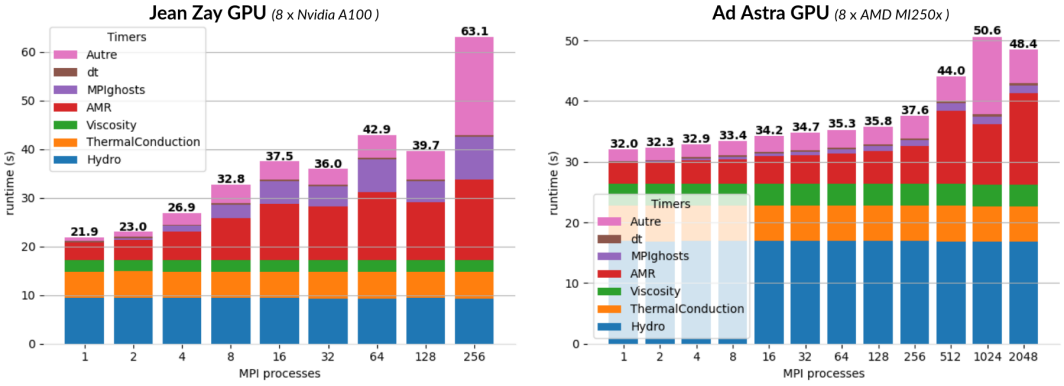

Figure 4: Weak scalability of the convection run on GPUs for French Supercomputers. **Left**: Scalability on Jean-Zay. Each node has 8 Nvidia a100 GPUs. **Right**: Scalability on Adastra. Each node has 4 AMD MI250 GPUs. Each process is bound to one MI250 GCD, so one GPU holds 2 MPI-processes and one node 8.

We ran that scalability test on CPUs and GPUs on the Jean-Zay and Adastra French supercomputers. The results can be seen on Fig 4 for the GPUs. Scalability is indeed good, and in the case of Adastra we reach 2048 GPUs, having a total domain with approximately 62 billion cells. The computational kernels in blue, orange and green scale perfectly while the AMR cycle tends to increase time consumption slightly due to an increase of usage in communication bandwidth. Optimization is currently ongoing to improve the timing of the "other" ("Autre") bars that also tend to grow when increasing the number of processes. Finally we can see a discrepancy between the times-to-solution on a100 and mi250 that should be roughly the same. We believe this discrepancy comes from a difficulty for the ROCm compiler to optimize the HIP code generated by Kokkos due to the high level of abstractions in Dyablo.

4 Conclusions

Dyablo is nearing its first production runs. We have shown that the methodology employed for the development of the code allowed physicists from very different communities to implement their schemes and models easily. Thanks to the use of Kokkos, Dyablo is natively compatible with all French Tier-0 and Tier-1 supercomputers. The code is now able to run and scale on a large number of GPUs on the largest French supercomputer.

Dyablo is in constant evolution with more and more physicists and numericists joining the development team. Ongoing developments for the next milestone includes :

- simulations using curvilinear coordinates,
- more efficient linear solvers for gravity,
- 3rd and 4th order spatial and temporal schemes for the simulation of highly stratified atmospheres,
- non-ideal MHD including resistivity, hall-effect and ambipolar diffusion,
- profiling and optimization on large clusters,
- the inclusion of cosmic-rays and dust,
- the development of dedicated file formats for block-based AMR.

In the coming year, we expect to run production grade simulations of MHD convective dynamo and magnetized multi-layered solar simulation. New benchmarks focused cosmology and MHD simulations will be added to the test suite as well to validate the performances of different setups of Dyablo on most available hardware in European supercomputers.

Acknowledgements

We acknowledge the financial support of the ERC Whole Sun Synergy grant #810218 and CNES Solar Orbiter supports. We are thankful to GENCI via project 1623. We also acknowledge the support of INSU and its programs PNST, PNCG, PNHE, PNPS and PCMI. This project was provided with computing HPC and storage resources by GENCI at TGCC thanks to the grant 2024-A0150411049 on the supercomputer Joliot Curie's ROME partition.

References

[1] D. Aubert and A. Durocher. Ginea and dyablo. In A. Siebert, K. Baillié, E. Lagadec, N. Lagarde, J. Malzac, J. B. Marquette, M. N'Diaye, J. Richard, and O. Venot, editors, *SF2A-2021: Proceedings of the Annual meeting of the French Society of Astronomy and Astrophysics*, pages 473–475, Dec. 2021. URL `https://ui.adsabs.harvard.edu/abs/2021sf2a.conf..473A`.

[2] D. Aubert and R. Teyssier. A radiative transfer scheme for cosmological reionization based on a local eddington tensor. *Monthly Notices of the Royal Astronomical Society*, 387(1):295–307, June 2008. ISSN 1365-2966. doi: 10.1111/j.1365-2966.2008.13223.x.

[3] C. Burstedde. Parallel tree algorithms for amr and non-standard data access. *ACM Transactions on Mathematical Software*, 46(4):1–31, Nov. 2020. ISSN 1557-7295. doi: 10.1145/3401990.

[4] H. Carter Edwards, C. R. Trott, and S. Daniel. Kokkos: Enabling manycore performance portability throughpolymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, 74: 3202–3216, 2014.

[5] F. Cattaneo, N. H. Brummell, J. Toomre, A. Malagoli, and N. E. Hurlburt. Turbulent compressible convection. , 370:282, Mar. 1991. doi: 10.1086/169814. URL https://ui.adsabs.harvard.edu/abs/1991ApJ...370..282C.

[6] M. Delorme, A. Durocher, A. S. Brun, and A. Strugarek. Novel numerical methods for solar convection: The dyablo whole-sun adaptative mesh refinement code. In J. Richard, A. Siebert, E. Lagadec, N. Lagarde, O. Venot, J. Malzac, J. B. Marquette, M. N'Diaye, and B. Briot, editors, *SF2A-2022: Proceedings of the Annual meeting of the French Society of Astronomy and Astrophysics*, pages 209–213, Dec. 2022. URL https://ui.adsabs.harvard.edu/abs/2022sf2a.conf..209D.

[7] S. Gottlieb, C.-W. Shu, and E. Tadmor. Strong stability-preserving high-order time discretization methods. *SIAM Review*, 43(1):89–112, Jan. 2001. ISSN 1095-7200. doi: 10.1137/s003614450036757x.

[8] G. R. J. Lesur, S. Baghdadi, G. Wafflard-Fernandez, J. Mauxion, C. M. T. Robert, and M. Van den Bossche. Idefix: A versatile performance-portable godunov code for astrophysical flows. , 677: A9, Sept. 2023. doi: 10.1051/0004-6361/202346005. URL https://ui.adsabs.harvard.edu/abs/2023A&A...677A...9L.

[9] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. Springer Berlin Heidelberg, 2009. ISBN 9783540498346. doi: 10.1007/b79761.

[10] C. R. Trott, D. Lebrun-Grandie, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D. S. Hollman, D. Ibanez, N. Liber, J. Madsen, J. Miles, D. Poliakoff, A. Powell, S. Rajamanickam, M. Simberg, D. Sunderland, B. Turcksin, and J. Wilke. Kokkos 3: Programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems*, 33(4):805–817, Apr. 2022. ISSN 2161-9883. doi: 10.1109/tpds.2021.3097283.