The Journal of Open Source Software

dNami: a framework for solving systems of balance laws using explicit numerical schemes on structured meshes

Nicolas Alferez[*1], **Emile Touber**[†‡2, 3], **Stephen D. Winn**[2, 3], **and Yussuf Ali**[2]

1 Laboratoire DynFluid, Conservatoire National des Arts et Métiers, Paris, France **2** Okinawa Institute of Science and Technology, Okinawa, Japan **3** Department of Mechanical Engineering, Imperial College London, London, UK

DOI: 10.21105/joss.04186

Software

- Review
- Repository
- Archive

Editor: Kristen Thyng

Reviewers:

- @JamieJQuinn
- @mancellin

Submitted: 10 February 2022
Published: 22 February 2022

Summary

The time evolution of a variety of physical and biological processes may be described by systems of balance laws, which, if given appropriate initial and boundary conditions, dictate the future states of the systems. For instance, systems of balance laws invoking mass, momentum and energy have been incredibly successful at providing meaningful insights to the future states of realistic systems in physics (e.g. fluid dynamics). Yet, experimenting numerically with such systems still requires much implementation time. `dNami` (di:na:mi:) was created so that more research time is spent exploring the dynamical properties of the system of balance laws of interest to the user, and less time is wasted on its numerical implementation across the whole computational spectrum, from the initial small-scale exploratory work on a workstation to the final large-scale computations on national clusters. Thus, `dNami` is a computational framework to study problems of the form:

$$\frac{\partial \mathbf{q}}{\partial t} = f(\mathbf{q}) \; + \; \text{initial/boundary conditions}, \tag{1}$$

in a flexible and efficient manner, where $\mathbf{q} \in \mathbb{R}^n$ is a vector of n real-valued unknowns, t is time, and $f(\mathbf{q})$ is a generic function of $\mathbf{q}$ which may include differential and algebraic operators.

The ability of `dNami` to clearly separate the problem statement from its numerical implementation (often a major time sink in research laboratories) is rooted in the flexibility of the Python language so as to let the user define her/his own system of balance laws in the most natural way (i.e. using a human-readable syntax), which is then interpreted in Fortran to build a computationally-efficient library of Equation 1 which is callable from Python. Users can then easily interact with their own system of balance laws, including at runtime, thereby making it possible to integrate solutions to Equation 1 with other tools and libraries (e.g. optimisation and stability tools) to fully explore the properties of the system, seamlessly from small to large-scale calculations.

State of the field

Code developments aimed at producing numerical solvers for Equation 1 typically follow two different strategies which usually involve two different communities.

[*] co-first author
[†] co-first author
[‡] corresponding author

The Journal of Open Source Software

For small-enough problems of the type of Equation 1, generic solutions provided by high-level languages may be used. Notable examples may be found in the vast offer provided by the scientific Python community. One of the reasons for the Python language success in computational science is its versatility to researchers' computational needs (e.g. diverse object and data-type structures, various interactions with data at runtime using the large offer of tools from the community). However, a common major drawback is the inability to easily tackle problems involving a large number of degrees of freedom, which typically require highly efficient parallel capabilities. Note that solutions involving pre-compiled Python modules, see SciPy project (Virtanen et al., 2020), or just-in-time compilation, see Numba project (Lam et al., 2015) exist but are most of the time restricted to workstation workflow and incompatible with large-scale computing of complex problems.

Thus, researchers targeting Equation 1 and requiring large-scale computing capabilities typically have to tackle the tedious problem of High-Performance Computing (HPC) development on their own. This is time consuming for a non-specialist and often results in conservative technical solutions that do not comply with the rapid evolution of hardware architecture that comes with continuous change of parallelisation paradigms. Alternatively, such researchers can collaborate with HPC specialists for the HPC-layer of the solver. Several examples of such fruitful joint efforts are available in the literature, see for instance Bernardini et al. (2021) and Krishnan et al. (2017) in Computational Fluid Dynamics (CFD). However, this approach often results in highly technical source codes, targeting specific problems that are difficult to modify in time (e.g. new $f(\mathbf{q})$, new choice of boundary conditions).

A relatively recent trend aimed at maintaining HPC capabilities whilst providing user and problem-specific flexibilities is the use of Domain-Specific-Languages (DSL) libraries developed by HPC specialists to tackle Equation 1. Examples of such approaches can be found in CFD, a notoriously HPC-intensive domain of computational physics (Di Renzo & Pirozzoli, 2021; Lusher et al., 2021; Witherden et al., 2014). Other DSL-based solvers directly target Equation 1 (Burns et al., 2020; Miquel, 2021). Although DSL approaches provide the versatility of the physical problem to solve and the efficient adaptability to modern hardware architectures, they do require users to learn the new DSL and drastically change paradigm in their developing approach. In addition, most DSL-based solutions rely on compiled binary executables produced from a low-level programming language (typically C or Fortran) to achieve their performance. They do not provide the flexibility of pre-compiled Python modules from the user point of view (especially at runtime). `dNami` aims at conciliating the DSL-based approach with all the advantages of a Python pre-complied module so as to solve general problems like Equation 1 on both small and large scales.

Features

At the core of `dNami` is the translation of symbolic expressions written in high-level Python language to discretise equations in low-level Fortran language. `dNami` employs explicit schemes to discretise differential operators. For the temporal derivative of Equation 1, a low-storage 3^{rd} order Runge–Kutta (RK) scheme is used (other explicit schemes may easily be implemented). Spatial derivatives are discretised using finite differences of arbitrary orders provided by the user. A choice between standard and optimised schemes (such as those in Bogey & Bailly, 2004) is available. Both the governing equation in the form of "$\partial\mathbf{q}/\partial t = f(\mathbf{q})$" and the boundary conditions are specified symbolically. `dNami` automatically deals with stencil-size and order reduction close to boundaries using the user-specified symbolic equations, which removes the need for time-consuming and often problem-specific code development.

The source-to-source translation is performed by a set of Python functions using regular expressions (see `genKer.py`). The produced discretised version of Equation 1 is then inserted into appropriate do-loops included in Fortran template files by pre-processing techniques. This

simple yet effective strategy makes it possible for the HPC-layer to be tailored at the template-file level independently of Equation 1. Finally, the resulting Fortran source code is compiled as a shared library and optimised through the auto-optimisation process of modern Fortran compilers. A Python module is then created with a high-level interface of the shared library functions using F2PY (Peterson, 2009). It is important to stress that researchers can still follow a traditional development workflow, in a pure Fortran environment, either at the shared library level or inside the high-level interface.

`dNami` solves Equation 1 on structured grids. Parallelisation is then ensured via classical domain decomposition techniques through point-to-point MPI communications using `mpi4py` (Dalcin & Fang, 2021). Efficient vectorisation of intense stencil-based computations are achieved via manual loop unrolling (yet automatically done by `genKer.py`) and cache-blocking techniques following recommendations from Andreolli et al. (2015).

Python is used at runtime to set up the run parameters and initial conditions. More importantly the time loop is exposed to Python, giving the user the freedom to interact with the computation inside the RK steps at run-time and to plug-in external libraries and/or output custom values with in-place data read and write between Python and Fortran. `dNami`'s Python interface thus allows easy integration of pre-processing, co-processing and post-processing tools.

Current `dNami` applications

`dNami` is currently being used by researchers to study linear and nonlinear wave phenomena in various systems, from engineering to biology and geo/astro-physical flows. For example, we solve the Navier–Stokes equations to study shock waves and compressible turbulence, reaction-diffusion equations to study dissipative solitons in biological networks, the magneto-hydrodynamic equations to study shock-entropy interactions in plasmas, and the shallow-water equations to study geophysical flows (e.g. internal gravity waves, tsunamis and meteotsunamis, two-dimensional compressible turbulence).

To illustrate the ability of `dNami` to quickly adapt to new systems of balance laws on supercomputers, we used the eruption of the Tonga volcano on 15 January 2022, which made tsunami warnings fail in Japan (Kataoka et al., 2022). A new system of balance laws to couple gravity-driven waves (shallow-water equations) in the ocean and the primary Lamb wave (compressible Euler equations) in the atmosphere following the eruption was created and computed on a global scale using available bathymetry data. The arrival times of the meteotsunami waves are found to be in good agreement with tide gages around the globe. A snapshot of the resulting animation is shown in Figure 1. Starting from an empty project, the global-scale meteotsunami was set and run in just a day owing to the flexibility with which `dNami` could handle a new set of equations, a choice of spherical coordinates, and importing geotiff files for the bathymetry data.

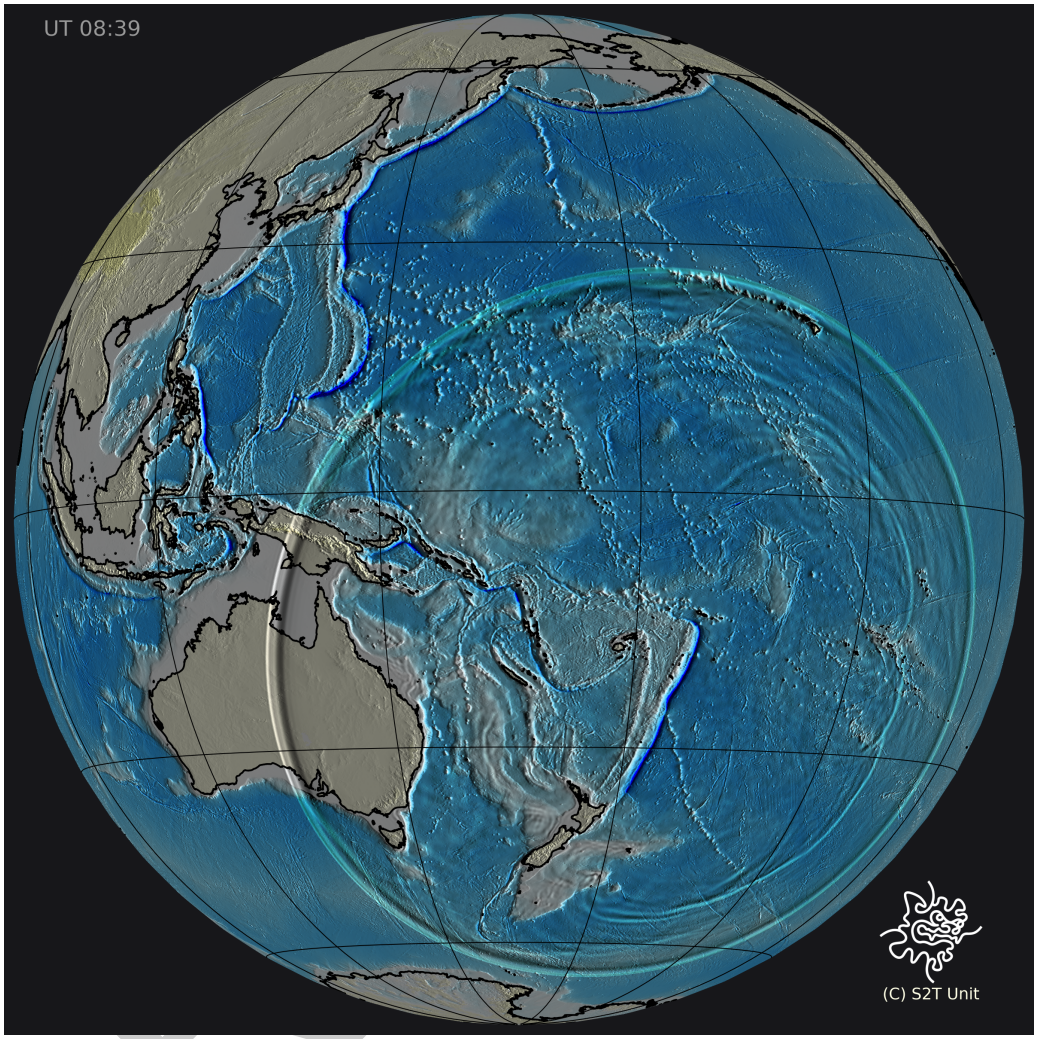

Figure 1: Example computation of a global simulation of the atmopsheric and water-height disturbance due to the January 2022 Tonga volcano explosion. Time is shown in UT starting on January 15th.

Acknowledgements

We are grateful for the help and support provided by the Scientific Computing and Data Analysis section of Research Support Division at OIST. This work used computational resources of the supercomputer Fugaku provided by RIKEN through the HPCI System Research Project (Project ID: hp200198 and hp210186).

References

Andreolli, C., Thierry, P., Borges, L., Skinner, G., & Yount, C. (2015). Chapter 23 - characterization and optimization methodology applied to stencil computations. In J. Reinders & J. Jeffers (Eds.), *High performance parallelism pearls* (pp. 377–396). Morgan Kaufmann. https://doi.org/10.1016/B978-0-12-802118-7.00023-6

Bernardini, M., Modesti, D., Salvadore, F., & Pirozzoli, S. (2021). STREAmS: A high-fidelity accelerated solver for direct numerical simulation of compressible turbulent flows. *Computer Physics Communications*, *263*, 107906. https://doi.org/10.1016/j.cpc.2021.107906

Bogey, C., & Bailly, C. (2004). A family of low dispersive and low dissipative explicit schemes for flow and noise computations. *Journal of Computational Physics*, *194*(1), 194–214. https://doi.org/10.1016/j.jcp.2003.09.003

Burns, K. J., Vasil, G. M., Oishi, J. S., Lecoanet, D., & Brown, B. P. (2020). Dedalus: A flexible framework for numerical simulations with spectral methods. *Physical Review Research*, *2*(2), 023068. https://doi.org/10.1103/PhysRevResearch.2.023068

Dalcin, L., & Fang, Y.-L. L. (2021). mpi4py: Status update after 12 years of development. *Computing in Science Engineering*, *23*(4), 47–54. https://doi.org/10.1109/MCSE.2021.3083216

Di Renzo, M., & Pirozzoli, S. (2021). HTR-1.2 solver: Hypersonic Task-based Research solver version 1.2. *Computer Physics Communications*, *261*, 107733. https://doi.org/10.1016/j.cpc.2020.107733

Kataoka, R., Winn, S. D., & Touber, E. (2022). Meteotsunamis in Japan associated with the Tonga eruption in January 2022. *EarthArXiv*. https://doi.org/10.31223/X55K8V

Krishnan, A., Mesnard, O., & Barba, L. A. (2017). cuIBM: A GPU-based immersed boundary method code. *Journal of Open Source Software*, *2*(15), 301. https://doi.org/10.21105/joss.00301

Lam, S. K., Pitrou, A., & Seibert, S. (2015). Numba: A llvm-based python jit compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 1–6. https://doi.org/10.1145/2833157.2833162

Lusher, D. J., Jammy, S. P., & Sandham, N. D. (2021). OpenSBLI: Automated code-generation for heterogeneous computing architectures applied to compressible fluid dynamics on structured grids. *Computer Physics Communications*, *267*, 108063. https://doi.org/10.1016/j.cpc.2021.108063

Miquel, B. (2021). Coral: A parallel spectral solver for fluid dynamics and partial differential equations. *Journal of Open Source Software*, *6*(65), 2978. https://doi.org/10.21105/joss.02978

Peterson, P. (2009). F2PY: A tool for connecting fortran and python programs. *International Journal of Computational Science and Engineering*, *4*(4), 296–305. https://doi.org/10.1504/IJCSE.2009.029165

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Witherden, F. D., Farrington, A. M., & Vincent, P. E. (2014). PyFR: An open source framework for solving advection-diffusion type problems on streaming architectures using the flux reconstruction approach. *Computer Physics Communications*, *185*(11), 3028–3040. https://doi.org/10.1016/j.cpc.2014.07.011