The Journal of Open Source Software

GridapSolvers.jl: Scalable multiphysics finite element solvers in Julia

Jordi Manyer [1*¶], **Alberto F. Martín** [2*], **and Santiago Badia** [1]

1 School of Mathematics, Monash University, Clayton, Victoria, 3800, Australia. **2** School of Computing, Australian National University, Canberra, ACT, 2600, Australia ¶ Corresponding author * These authors contributed equally.

DOI: 10.21105/joss.07162

Software

- Review
- Repository
- Archive

Editor: Patrick Diehl

Reviewers:

- @PetrKryslUCSD
- @Leticia-maria

Submitted: 21 August 2024
Published: 05 October 2024

Summary and statement of need

The ever-increasing demand for resolution and accuracy in mathematical models of physical processes governed by systems of Partial Differential Equations (PDEs) can only be addressed using fully-parallel advanced numerical discretization methods and scalable solvers, able to exploit the vast amount of computational resources in state-of-the-art supercomputers.

One of the biggest scalability bottlenecks within Finite Element (FE) parallel codes is the solution of linear systems arising from the discretization of PDEs. The implementation of exact factorization-based solvers in parallel environments is an extremely challenging task, and even state-of-the-art libraries such as MUMPS (Amestoy et al., 2001, 2019) and PARDISO (Schenk & Gärtner, 2011) have severe limitations in terms of scalability and memory consumption above a certain number of CPU cores. Hence, the use of iterative methods is crucial to maintain scalability of FE codes. Unfortunately, the convergence of iterative methods is not guaranteed and rapidly deteriorates as the size of the linear system increases. To retain performance, the use of highly scalable preconditioners is mandatory. For simple problems, algebraic solvers and preconditioners (i.e., those based solely on the algebraic system) are enough to obtain robust convergence. Many well-known libraries providing algebraic solvers already exist, such as PETSc (Balay et al., 2021), Trilinos (Trilinos Project Team, 2020), and Hypre (Hypre, n.d.). However, algebraic solvers are not always suited to deal with more challenging problems.

In these cases, solvers that exploit the physics and mathematical discretization of the particular problem are required. This is the case of many multiphysics problems involving differential operators with a large kernel such as the divergence (Arnold et al., 1997) and the curl (Arnold et al., 2000). Examples can be found amongst highly relevant problems such as Navier-Stokes, Maxwell, and Darcy. Scalable solvers for this type of multiphysics problems rely on exploiting the block structure of such systems to find a spectrally equivalent block-preconditioner, and are often tied to a specific discretization of the underlying equations.

As a consequence, high-quality open-source parallel finite element packages like FEniCS (Logg et al., 2012) and deal.II (Arndt et al., 2021) already provide implementations of several state-of-the-art physics-informed solvers (Cui et al., 2024; Farrell et al., 2021). The Gridap ecosystem (Badia & Verdugo, 2020) aims to provide a similar level of functionality within the Julia programming language (Bezanson et al., 2017).

To this end, GridapSolvers is a registered Julia software package which provides highly scalable physics-informed solvers tailored for the FE numerical solution of PDEs on parallel computers within the Gridap ecosystem of packages. Emphasis is put on the modular design of the library, which easily allows new preconditioners to be designed from the user's specific problem.

Building blocks and composability

Figure 1 depicts the relation among GridapDistributed and other packages in the Julia package ecosystem.

The core library Gridap (Badia & Verdugo, 2020) provides all necessary abstraction and interfaces needed for the FE solution of PDEs (Verdugo & Badia, 2022) for serial computing. GridapDistributed (Badia et al., 2022) provides distributed-memory counterparts for these abstractions, while leveraging the serial implementations in Gridap to handle the local portion on each parallel task. GridapDistributed relies on PartitionedArrays (Verdugo, 2021) in order to handle the parallel execution model (e.g., message-passing via the Message Passing Interface (MPI) (Message Passing Interface Forum, 2021)), global data distribution layout, and communication among tasks. PartitionedArrays also provides a parallel implementation of partitioned global linear systems (i.e., linear algebra vectors and sparse matrices) as needed in grid-based numerical simulations. This parallel framework does however not include any performant solver for the resulting linear systems. This was delegated to GridapPETSc (Verdugo et al., 2021), which provides a plethora of highly-scalable and efficient algebraic solvers through a high-level interface to the Portable, Extensible Toolkit for Scientific Computation (PETSc) (Balay et al., 2021).

GridapSolvers complements GridapPETSc with a modular and extensible interface for the design of physics-informed solvers. Some of the highlights of the library are:

- A set of HPC-first implementations for popular Krylov-based iterative solvers. These solvers extend Gridap's API and are fully compatible with PartitionedArrays.
- A modular, high-level interface for designing block-based preconditioners for multiphysics problems. These preconditioners can be used together with any solver compliant with Gridap's API, including those provided by GridapPETSc.
- A generic interface to handle multi-level distributed meshes, with full support for Adaptative Mesh Refinement (AMR) using p4est (Burstedde et al., 2011) through GridapP4est (Martin, 2021).
- A modular implementation of Geometric MultiGrid (GMG) solvers (Briggs et al., 2000), allowing different types of smoothers and restriction/prolongation operators.
- A generic interface for patch-based subdomain decomposition methods, and an implementation of patch-based smoothers for GMG solvers. Here the term "patch-based" refers to the use of local overlapping subdomains (patches) built by aggregation of cells around a given vertex, face or cell. See Farrell et al. (2021) and Cui et al. (2024) for more details.

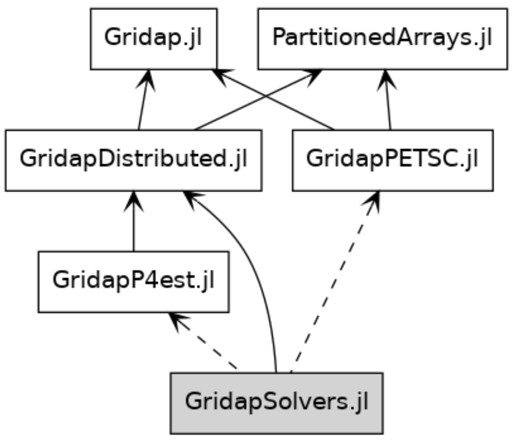

Figure 1: GridapSolvers and its relation to other packages in the Julia package ecosystem. In this diagram, each node represents a Julia package, while the (directed) arrows represent relations (dependencies) among packages. Dashed arrows mean the package can be used, but is not required.

Demo

The following code snippet shows how to solve a 2D incompressible Stokes cavity problem in a Cartesian domain $\Omega = [0,1]^2$. We discretize the velocity and pressure in $H^1(\Omega)$ and $L^2(\Omega)$ respectively, and use the well known stable element pair $Q_k \times P_{k-1}^{-}$ with $k = 2$. For the cavity problem, we fix the velocity to $u_t = \hat{x}$ on the top boundary $\Gamma_t = (0,1) \times \{1\}$, and homogeneous Dirichlet boundary conditions elsewhere. We impose a zero-mean pressure constraint to have a solvable system of equations. Given discrete spaces $V \times Q_0$, we find $(u,p) \in V \times Q_0$ such that

$$\int_\Omega \nabla v : \nabla u - (\nabla \cdot v)p - (\nabla \cdot u)q = \int_\Omega v \cdot f \quad \forall v \in V_0, q \in Q_0$$

where V_0 is the space of velocity functions with homogeneous boundary conditions everywhere.

The system is block-assembled and solved using a flexible Generalised Minimum Residual (F-GMRES) solver, together with a block-triangular Schur-complement-based preconditioner. We eliminate the velocity block and approximate the resulting Schur complement by a pressure mass matrix. A more detailed overview of this preconditioner as well as its spectral analysis can be found in Elman et al. (2014). The resulting block structure for the system matrix $\mathcal{A}$ and our preconditioner $\mathcal{P}$ is

$$\mathcal{A} = \begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix}, \quad \mathcal{P} = \begin{bmatrix} A & B^T \\ 0 & -M \end{bmatrix}$$

with A the velocity Laplacian block, and M a pressure mass matrix.

Application of the above block-preconditioner requires both diagonal sub-matrices to be solved. The pressure block M is solved using a Conjugate Gradient (CG) solver with Jacobi preconditioner. The velocity block A is solved by a 2-level V-cycle GMG solver, where the coarsest level is solved exactly in a single processor. The code for this example can be found here. It is set up to run in parallel with 4 MPI tasks and can be executed with the following command: `mpiexec -n 4 julia --project=. demo.jl`.

Parallel scaling benchmark

The following section shows scalability results for the demo problem discussed above. We run our code on the Gadi supercomputer, which is part of the Australian National Computational Infrastructure (NCI). We use Intel's Cascade Lake 2x24-core Xeon Platinum 8274 nodes. Scalability is shown for up to 64 nodes, for a fixed local problem size of 48x64 quadrangle cells per processor. This amounts to a maximum size of approximately 37M cells and 415M degrees of freedom distributed amongst 3072 processors. Within the GMG solver, the number of coarsening levels is progressively increased to keep the global size of the coarsest solve (approximately) constant. The coarsest solve is then performed by a CG solver preconditioned by an Algebraic MultiGrid (AMG) solver, provided by PETSc (Balay et al., 2021) through the package GridapPETSc.jl.

The results in Figure 2 show that the code scales relatively well up to 3072 processors, with loss in performance mostly tied to the number of GMG levels used for the velocity solver. The number of F-GMRES iterations required for convergence is also shown to be relatively constant (and even decreasing for bigger problem sizes), indicating that the preconditioner is robust with respect to the problem size.

The code used to create these results can be found here. The exact releases for the packages used are provided by Julia's `Manifest.toml` file.

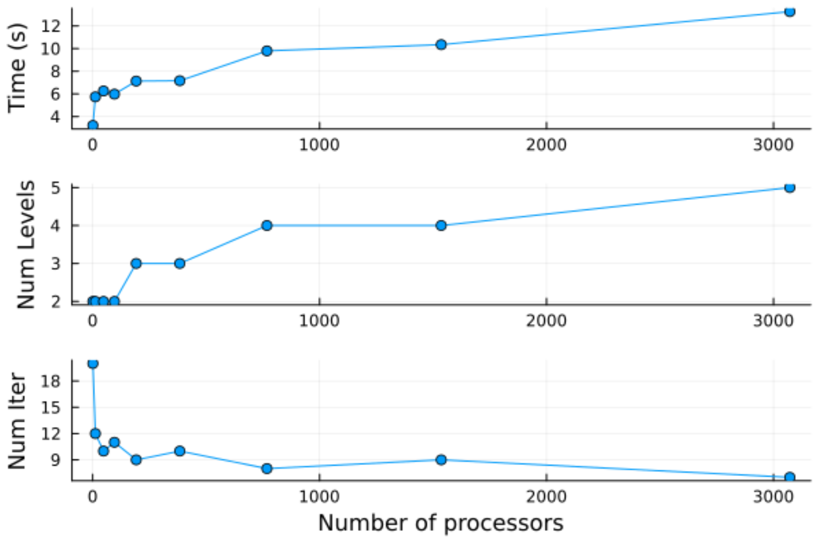

Figure 2: Top: Weak scalability for a Stokes problem in 2D. Time is given per F-GMRES iteration, as a function of the number of processors. **Middle**: Number of coarsening levels for the GMG solver, as a function of the number of processors. **Bottom**: Number of F-GMRES iterations required for convergence.

Acknowledgements

This research was partially funded by the Australian Government through the Australian Research Council (project number DP210103092). This work was also supported by computational resources provided by the Australian Government through NCI under the National Computational Merit Allocation Scheme (NCMAS), the Monash-NCI partnership scheme and the ANU Merit Allocation Scheme (ANUMAS).

References

Amestoy, P. R., Buttari, A., L'Excellent, J.-Y., & Mary, T. (2019). Performance and scalability of the block low-rank multifrontal factorization on multicore architectures. *ACM Transactions on Mathematical Software*, *45*, 2:1–2:26. https://doi.org/10.1145/3242094

Amestoy, P. R., Duff, I. S., Koster, J., & L'Excellent, J.-Y. (2001). A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, *23*(1), 15–41. https://doi.org/10.1137/s0895479899358194

Arndt, D., Bangerth, W., Blais, B., Fehling, M., Gassmöller, R., Heister, T., Heltai, L., Köcher, U., Kronbichler, M., Maier, M., Munch, P., Pelteret, J.-P., Proell, S., Simon, K., Turcksin, B., Wells, D., & Zhang, J. (2021). The `deal.II` library, version 9.3. *Journal of Numerical Mathematics*, *29*(3), 171–186. https://doi.org/10.1515/jnma-2021-0081

Arnold, D. N., Falk, R. S., & Winther, R. (1997). *Preconditing in H(div) and applications*. https://doi.org/10.1090/S0025-5718-97-00826-0

Arnold, D. N., Falk, R. S., & Winther, R. (2000). Multigrid in H(div) and H(curl). *Numerische Mathematik*, *85*, 197–217. https://doi.org/10.1007/PL00005386

Badia, S., Martín, A. F., & Verdugo, F. (2022). GridapDistributed: A massively parallel finite element toolbox in Julia. *Journal of Open Source Software*, *7*(74), 4157. https://doi.org/10.21105/joss.04157

Badia, S., & Verdugo, F. (2020). Gridap: An extensible finite element toolbox in Julia. *Journal of Open Source Software*, *5*(52), 2520. https://doi.org/10.21105/JOSS.02520

Balay, S., Abhyankar, S., Adams, M. F., Benson, S., Brown, J., Brune, P., Buschelman, K., Constantinescu, E., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W. D., Hapla, V., Isaac, T., Jolivet, P., Karpeev, D., Kaushik, D., Knepley, M. G., Kong, F., … Zhang, J. (2021). *PETSc/TAO users manual* (ANL-21/39 - Revision 3.16). Argonne National Laboratory.

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98. https://doi.org/10.1137/141000671

Briggs, W. L., Henson, V. E., & McCormick, S. F. (2000). *A multigrid tutorial, second edition* (Second). Society for Industrial; Applied Mathematics. https://doi.org/10.1137/1.9780898719505

Burstedde, C., Wilcox, L. C., & Ghattas, O. (2011). p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, *33*(3), 1103–1133. https://doi.org/10.1137/100791634

Cui, C., Grosse-Bley, P., Kanschat, G., & Strzodka, R. (2024). *An implementation of tensor product patch smoothers on GPU*. https://doi.org/10.48550/arXiv.2405.19004

Elman, H., Silvester, D., & Wathen, A. (2014). *Finite elements and fast iterative solvers: With applications in incompressible fluid dynamics*. Oxford University Press. https://doi.org/10.1093/acprof:oso/9780199678792.001.0001

Farrell, P. E., Knepley, M. G., Mitchell, L., & Wechsung, F. (2021). PCPATCH: Software for the topological construction of multigrid relaxation methods. *ACM Transactions on Mathematical Software*, *47*(3), 1–22. https://doi.org/10.1145/3445791

hypre*: High performance preconditioners*. (n.d.).

Logg, A., Mardal, K.-A., & Wells, G. (Eds.). (2012). *Automated solution of differential equations by the finite element method*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-23099-8

Martin, A. F. (2021). GridapP4est. In *GitHub repository*. GitHub. https://github.com/gridap/GridapP4est.jl

Message Passing Interface Forum. (2021). *MPI: A message-passing interface standard version 4.0*. https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf

Schenk, O., & Gärtner, K. (2011). PARDISO. In D. Padua (Ed.), *Encyclopedia of parallel computing* (pp. 1458–1464). Springer US. https://doi.org/10.1007/978-0-387-09766-4_90

Trilinos Project Team. (2020). *The Trilinos project website*. https://trilinos.github.io

Verdugo, F. (2021). PartitionedArrays. In *GitHub repository*. GitHub. https://github.com/fverdugo/PartitionedArrays.jl

Verdugo, F., & Badia, S. (2022). The software design of Gridap: A finite element package based on the Julia JIT compiler. *Computer Physics Communications*, *276*, 108341. https://doi.org/10.1016/j.cpc.2022.108341

Verdugo, F., Sande, V., & Martin, A. F. (2021). GridapPETSc. In *GitHub repository*. GitHub. https://github.com/gridap/GridapPETSc.jl