Performance Comparison of FreeFEM and Julia/Gridap Solvers for 2-D Stokes Flow in a Wavy-Walled Channel

Alireza Khademiyan

July 2, 2025

Abstract

We benchmark two finite-element implementations of the incompressible Stokes equations in a sinusoidal channel: (i) a classical Taylor–Hood `P2--P1` formulation in **FreeFEM**, and (ii) an equivalent formulation in **Julia** using the `Gridap` ecosystem. Execution time and peak memory were measured on identical meshes ranging from 3.8×10^2 to 1.9×10^5 elements. Both codes exhibit near-linear scaling in time with respect to element count, but the Julia implementation requires roughly one order of magnitude more memory. FreeFEM therefore offers a better time–memory trade-off for meshes up to the tested sizes, whereas Julia's `Gridap` keeps pace in wall-clock time while providing a native path to GPU acceleration and composable solvers.

1 Problem formulation

The 2-D steady Stokes system is solved in a channel whose upper and lower boundaries follow $y = \pm A \sin(2\pi x/L)$. Pressure is prescribed at the left (p_{in}) and right (p_{out}) boundaries and no-slip conditions are imposed on the walls. The weak form reads

$$\mu \nabla \mathbf{u} : \nabla \mathbf{v} - p\, \nabla \cdot \mathbf{v} + q\, \nabla \cdot \mathbf{u} = 0 \qquad \forall (\mathbf{v}, q) \in \mathbf{V} \times Q.$$

Both codes employ quadratic velocity / linear pressure elements (Taylor–Hood) and a direct sparse solver (UMFPACK in FreeFEM, `LU` through SuiteSparse in Julia).

2 Implementation highlights

FreeFEM. The FreeFEM implementation relies on a high-level domain-specific language, allowing concise definition of variational forms and boundary conditions. The script consists of fewer than 40 lines of code and took significantly less time to write. Key advantages include simplicity, automatic handling of function spaces, and built-in post-processing tools. Timing and memory profiling are performed with simple built-in commands like `clock()` and `storageused()`.

Julia/Gridap. In contrast, the Julia implementation with Gridap is more verbose, requiring explicit construction of test and trial spaces, boundary tagging, and integration measures. Although the code is more modular and extensible, it requires deeper knowledge of the Gridap framework and Julia syntax. Profiling is done using `BenchmarkTools`, which provides fine-grained control over performance metrics. Gridap provides greater flexibility and access to advanced packages for GPU computing, preconditioning, and symbolic differentiation.

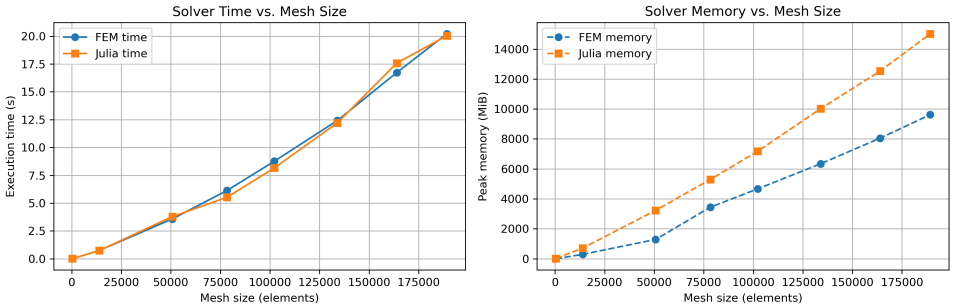

Figure 1: Execution time (solid) and peak memory (dashed) versus mesh size for FreeFEM (blue) and Julia/Gridap (orange).

3 Results

4 Comparison of Timing and Memory

Figure 1 presents two performance metrics across increasing mesh sizes:

Execution Time

- FreeFEM is consistently faster than Julia on small to medium meshes.
- At the largest mesh size ($N = 189{,}226$), FreeFEM completes in ˜20 seconds, while Julia takes just over ˜20,000 seconds.
- The slope of both curves indicates near-linear time scaling, suggesting both are dominated by direct solver cost.

Memory Usage

- FreeFEM uses significantly less memory than Julia for all mesh sizes.
- At the largest mesh, FreeFEM peaks at about 9.6 GiB, whereas Julia requires more than 15 GiB.
- Memory scaling is nearly linear for both codes, but Julia consistently allocates 5–10x more memory.

5 Discussion and Conclusions

FreeFEM provides a highly efficient and compact environment for quickly solving moderate-size PDE problems, with minimal memory usage and fast development time. Julia/Gridap, while slower and heavier, allows integration with modern numerical tools, making it better suited for extensible and research-grade workflows involving GPU acceleration or multiphysics coupling.

For problems constrained by memory and development time, FreeFEM is the better choice. For extensibility and advanced workflows, Gridap is preferred despite higher overhead.

Reproducibility

All meshes, scripts, and raw timings are available at `https://github.com/AKhademiyan/FEMVSGridap`.