Domain Decomposition in Firedrake

Malachi Phillips

Abstract:

The numerical solution of partial differential equations (PDEs) is becoming more ubiquitous in science and engineering applications. The clock speed of modern computers, meanwhile, has not increased. Therefore, there is a need for performant, parallel PDE solution methods. One such class of methods, known as Schwarz methods, offers a method of decomposing a problem into several smaller computations, each of which may be done independent of the other. Firedrake is a performant implementation of the unified form language (UFL) domain-specific language (DSL) for solving finite element (FE) problems of the FEniCS project. However, Firedrake has little support in expressing domain decomposition techniques, especially given that subdomain implementation remains an open issue within Firedrake. Herein, a miniature DSL for expressing domain decomposition methods directly into Firedrake is proposed. Although this proves possible, there is an extensive amount of modification still required in order to make this abstraction performant.

Methods and Code Complexity:

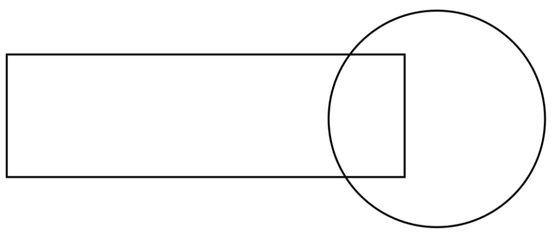


Figure 1: Complex domain comprised of a disk and a rectangle

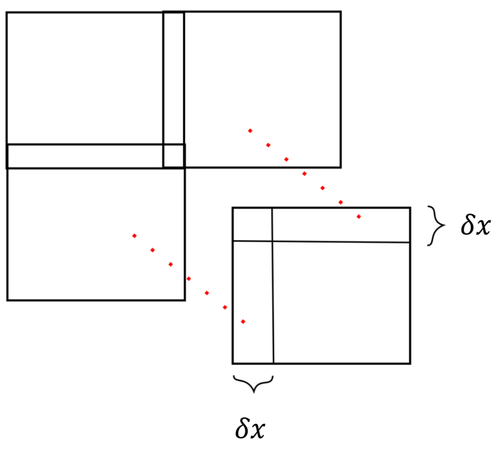


Figure 2: Four subdomains with constant amount of overlap

```
from firedrake import *
mesh=Mesh(...)
V=FunctionSpace(mesh, "CG", 1)
u=TrialFunction(V)
v=TestFunction(V)
f=Function(V)
x,y=SpatialCoordinate(mesh)
f.interpolate(sin(2*pi*x)*sin(2*pi*y))
par={'ksp_type':'preonly','pc_type':'lu'}
myBCSurfaces=[...]
myBCs=[]
for surface in myBCSurfaces:
    myBCs.append(DirichletBC(V,0,surface))
BEGIN PROGRAM HERE
mesh_name ~ mesh
form ~ (dot(grad(u), grad(v))) * dx
rhs ~ f * v * dx
space_variable ~ coord
functionSpace ~ V
dirichletBCs={...}
# domain number to id mapping
domains={...}
# d_Oi_n_Oj->interface id mapping
interfaces={...}
solution~u
solver_settings ~ solver_parameters=par
END PROGRAM HERE
```

Figure 3: Sample code for expressing domain decomposition. User specifies the domain and interface mappings from domain name and numbering in the mesh file.

Complex Firedrake Code

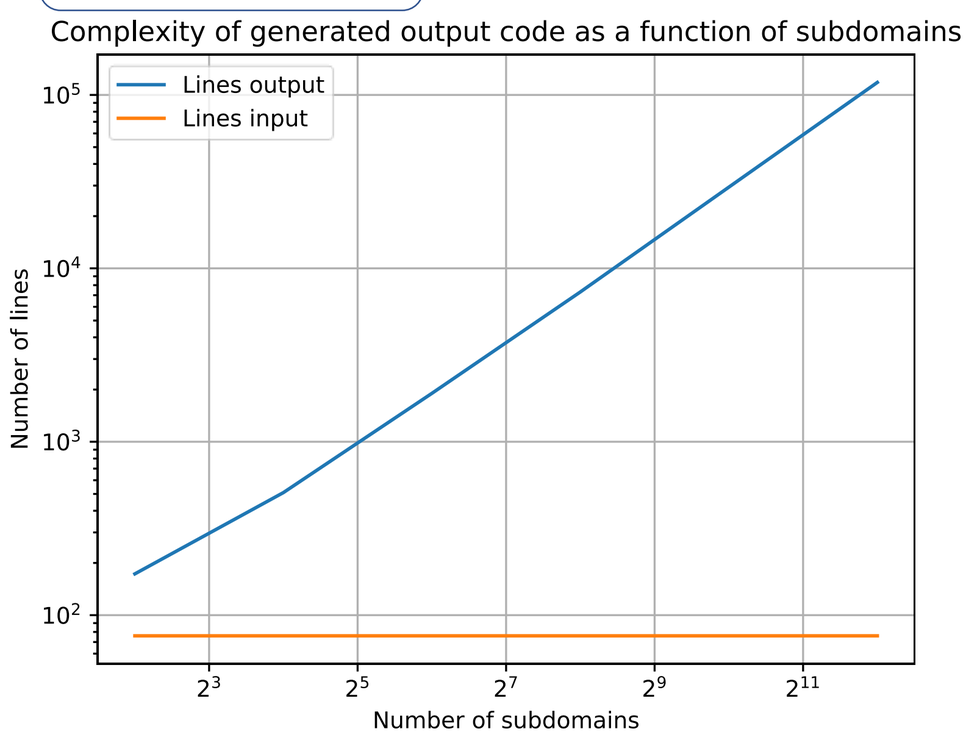
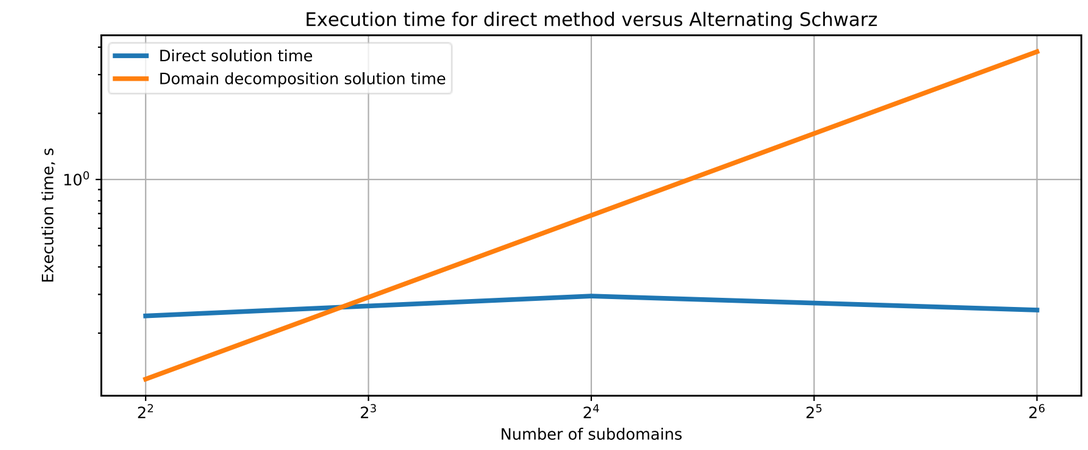


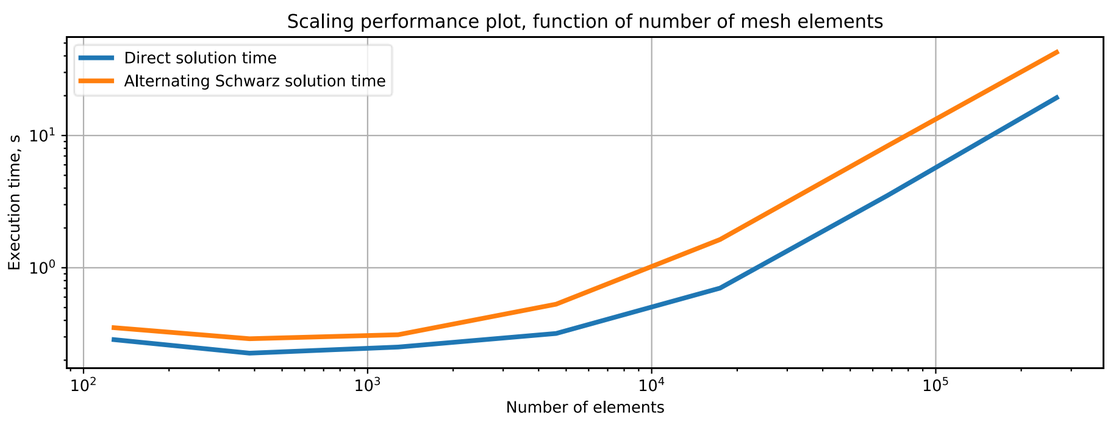
Figure 4: Complexity of output code (blue line) as a function of the number of subdomains compared with the complexity of the input code (orange line), which is not a function of the number of subdomains, modulo specifying the subdomain mappings.

Results:



(Above) Figure 5: Scaling with respect to the number of subdomains for the direct solution versus the alternating Schwarz algorithm.

(Bottom) Figure 6: Scaling with respect to the number of elements for the four-subdomain case (see Fig2). Comparison between direct solution time and alternating Schwarz algorithm.



Conclusions and Future work:

- Extending Unified Form Language (UFL) too difficult and error prone, therefore specifying a translation script is much simpler.
- Implemented source-to-source program translation from description of domain decomposition (given as subdomain and interface mappings plus coupling at the boundary condition) to appropriate Firedrake syntax.
- Resulting output in Firedrake is too slow for production use – Firedrake by itself is already performant.
- Domain decomposition techniques offer a naïve way of parallelizing FE codes by simply giving a collection of subdomains to a processor.
- Domain decomposition method is more performant than Firedrake for a single iteration, however, several iterations are needed for convergence.

Future Work:

- Implement true subdomain support in Firedrake
- Run performance scaling benchmarks on more general interface conditions between subdomains, e.g. Robin boundary conditions

References:

[1] Rathgeber, Florian, David A. Ham, Lawrence Mitchell, Michael Lange, Fabio Luporini, Andrew T. T. McRae, Gheorghe-Teodor Bercea, Graham R. Markall, and Paul H. J. Kelly. 2016. “Firedrake: Automating the Finite Element Method by Composing Abstractions.” ACM Trans. Math. Softw. 43 (3): 24:1–24:27. https://doi.org/10.1145/2998441.