The Twisted Road to Parallelizing Elliptic PDEs

by Jonathan Serencsa

Some background info

The general form of a symmetric linear second-order elliptic PDE is:

$$\sum_{i,j=1}^{d} \frac{\partial}{\partial x_i}(a_{ij}(x)\frac{\partial u}{\partial x_j}(x)) + c(x)u(x) = f(x)\ in\ \Omega \subset \mathbb{R}^d$$

$$u(x) = g(x)\ on\ \partial\Omega$$

which is a generalized form of Poisson's Equation

$$\Delta u(x) := \sum_{j=1}^{d} \frac{\partial^2 u}{\partial x_j^2}(x) = f(x)\ in\ \Omega$$

$$u(x) = g(x)\ on\ \partial\Omega$$

we will only deal with solving Poisson's equation

We discretize using some numerical method to get the linear system of equations,

$$A_\Omega u_\Omega = F - A_{\partial\Omega} u_{\partial\Omega}$$

$$u_{\partial\Omega} = G$$

or by eliminating the second equation, grouping the RHS and simplifying notation we get the linear system

$$Au = F$$

where A is an N x N matrix, u and F are vectors of size N. The vector u represents the approximate solution to our PDE.

Before we look to store A and F into memory and numerically solve the linear system we must first answer some preliminary questions:

- What is the structure of A? That is, is it symmetric? Invertible? Positive definite? Sparsity structure?

- How large is N for interesting 2D and 3D problems?

Answers:

- Our type of PDE limits us to the case where A is symmetric positive definite and thus invertible. If we use any reasonable approximation method (finite differences, finite element etc) we get $O(1)$ non-zeros per row. Thus requiring $O(N)$ storage to hold A.

- Typical problems in physics and weather forecasting can require grids which use thousands of grid points in each direction. For 3D problems, if A were dense, just storing our matrix could yield memory requirements in the exabyte range. But because of the sparsity we are only looking at storage requirements in the gigabyte range.

Now assuming we've constructed our linear system in memory we examine some methods for finding our solution u.

Direct Factorization Methods: Gaussian Elimination, LU Factorization, Cholesky	$O(N^3)$ complexity Factors require $O(N^2)$ storage, leading to memory overflow.
Specialized Factorizations: Banded and Sparse GE	$O(N^{2.33})$ & $O(N^2)$ complexity resp. Less storage requirement than above, but still not linear storage.
Classical Iterative Methods: Jacobi, Gauss-Seidel, Successive Over Relaxation	$O(N^{1.67} \ln N)$ for Jacobi and GS, $O(N^{1.33} \ln N)$ for SOR. Linear storage requirement
Conjugate Gradient Methods: Standalone or Preconditioned	$O(N^{1.33} \ln N)$ as standalone, but can get as good as $O(N \ln N)$ for complicated preconditioners
Multigrid Methods: V- or W- cycle and Full Multigrid	$O(N \ln N)$ for V- and W-cycle while $O(N)$ for Full Multigrid

Note: The listed complexities are for 3D domains, Banded and Sparse GE are better for 2D while Classical iterative methods and standalone CG gets a little worse.

It would appear that Full Multigrid is our best option, but it has limitations.

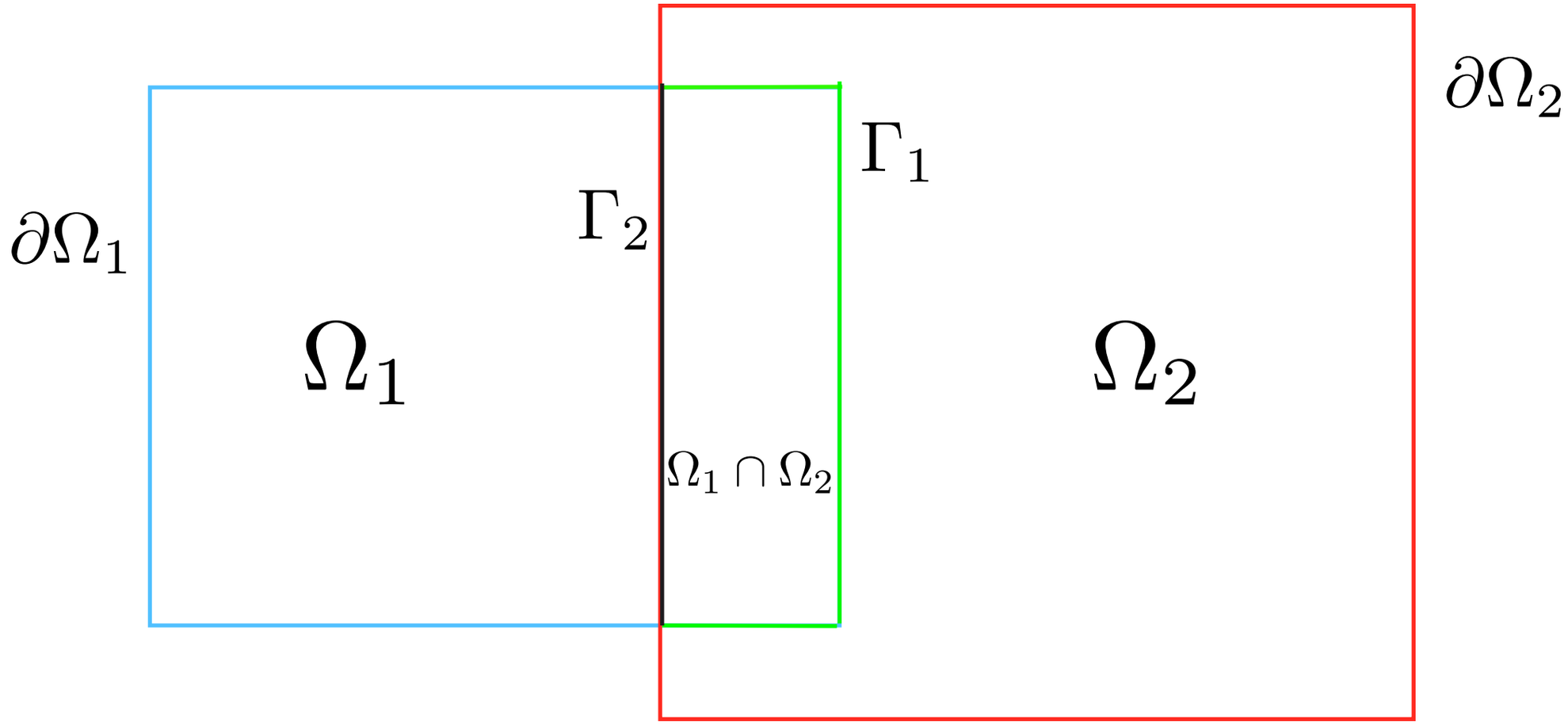
- Eventually we will run out of memory as problem size increases.
- Not easily parallelizable.
- Even if we did parallelize, the method to do so generates a lot of bottlenecks.

So we use a combination of Full Multigrid and PCG to achieve an algorithm which is naturally parallelizable...

Domain Decomposition Methods

General Idea

We subdivide the original domain Ω into M overlapping, smaller subdomains $\{\Omega_k\}_{k=1}^{M}$ where M may or may not be the number of processors.



For simplicity, we only draw the two subdomain case.

We then pose new subproblems on each of the subdomains where the boundary conditions are interpolated from the adjacent domain(s).

$$\Delta u_1(x) = f(x)\ in\ \Omega_1 \qquad \Delta u_2(x) = f(x)\ in\ \Omega_2$$

$$u_1(x) = g(x)\ on\ \partial\Omega_1 \setminus \Gamma_1 \qquad u_2(x) = g(x)\ on\ \partial\Omega_2 \setminus \Gamma_2$$

$$u_1(x) = u_2(x)|_{\Gamma_1}\ on\ \Gamma_1 \qquad u_2(x) = u_1(x)|_{\Gamma_2}\ on\ \Gamma_2$$

Optimally, we wish to solve these two problems simultaneously, but this is just as difficult as solving the large problem, so we must propose and iterative approach.

Given some initial guess for the solution u^0 we construct an iteration that solves each subproblem simultaneously, using the previous solution to set boundary conditions,

Set initial guess u^0, $u_1^0 = u^0|_{\Omega_1}$, $u_2^0 = u^0|_{\Omega_2}$,

$$\Delta u_1^k(x) = f(x)\ in\ \Omega_1 \qquad \Delta u_2^k(x) = f(x)\ in\ \Omega_1$$

$$u_1^k(x) = g(x)\ on\ \partial\Omega_1 \setminus \Gamma_1 \qquad u_2^k(x) = g(x)\ on\ \partial\Omega_2 \setminus \Gamma_2$$

$$u_1^k(x) = u_2^{k-1}(x)|_{\Gamma_1}\ on\ \Gamma_1 \qquad u_2^k(x) = u_1^{k-1}(x)|_{\Gamma_2}\ on\ \Gamma_2$$

and hopefully each sub-solution converges to its corresponding piece of the global solution. This method is known as Additive Schwarz.

The good news,

- Since each subdomain solve uses boundary info from previous step, each subdomain solve is completely independent from one another and can be solved in parallel

- No matter how much overlap we use, we only need to communicate a small fraction of information between each iteration.

- We can solve all the subproblems using Full Multigrid which has linear complexity.

and the bad news

- As a standalone method, the proposed iterative method will not in general converge.
- Even if it does converge, it does so very slowly, and the speed depends on the number of subdomains.
- Each iteration requires communication, which adds up in the long run.

Solutions to the bad news ... (sort of)

- If we use Additive Schwarz with Conjugate Gradients, the combined method will converge to our desired solution.

- Via a more precise initial guess obtained from solving the global problem on a coarse mesh, we get a bound on the number of iterations, independent of the number of subdomains.

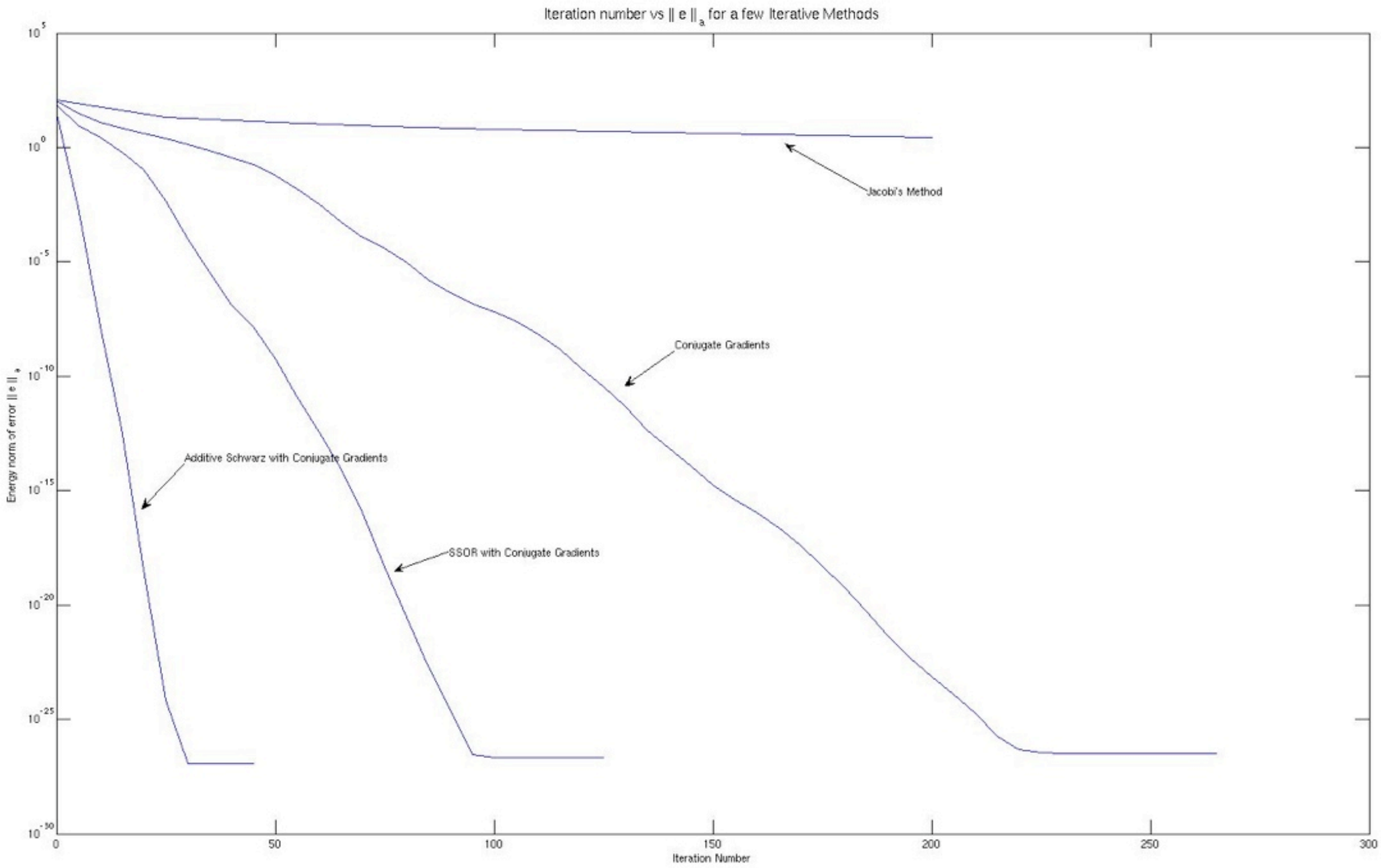
- To avoid high communication costs, each subdomain must minimize the surface to volume ratio.

But our solutions still generate more problems,

- Implementation of CG is much more complicated if we are unable to explicitly construct a global solution, and preliminary attempts to do so destroys prerequisites for CG (symmetry).

- Because of a $\ln N$ term which is a result of using an $O(N \ln N)$ method to parallelize, efficiency will always tend to zero as N increases.

Some preliminary results,



References

- B. Smith, P. Bjorstad, W. Gropp, *Domain Decomposition*. Cambridge University Press 1996

- A. Grama, A. Gupta, G. Karypis, V. Kumar, *Introduction to Parallel Computing*. Pearson Education Limited 2003

- M. Holst, Numerical Methods: Adaptive Methods for nonlinear PDE with Application to Poisson-Botlzman Eqn. *IPAM Workshop* 2005

- M. Holst, S. Vandewalle, Schwarz Methods: To symmetrize or not to symmetrize. *SIAM J. Numer. Anal.*, 34:699-722, 1997