Asynchronous Iterative Solvers for Extreme-Scale Computing

Edmond Chow, Georgia Institute of Technology
Erik Boman, Sandia National Laboratories
Jack J. Dongarra, University of Tennessee
Daniel B. Szyld, Temple University

SIAM Annual Meeting, Pittsburgh, PA
July 10-14, 2017

Overview

- Motivation for asynchronous methods; simple examples
- Optimized Schwarz method
- Parallel and distributed Southwell methods

Fixed-point iteration for solving $x = G(x)$

$$x^{(k+1)} = G(x^{(k)}), \quad k = 0, 1, 2, \ldots$$

Written explicitly:

$$\begin{aligned}
x_1 &= g_1(x_1, x_2, x_3, x_4, \cdots, x_m) \\
x_2 &= g_2(x_1, x_2, x_3, x_4, \cdots, x_m) \\
x_3 &= g_3(x_1, x_2, x_3, x_4, \cdots, x_m) \\
x_4 &= g_4(x_1, x_2, x_3, x_4, \cdots, x_m) \\
&\vdots \\
x_m &= g_m(x_1, x_2, x_3, x_4, \cdots, x_m)
\end{aligned}$$

Synchronous: all updates use components of x at the same iteration

Asynchronous: updates are assigned to processors statically or dynamically; updates use latest components of x that are available

Standard references on asynchronous iterative methods

Chazan, Miranker, “Chaotic relaxation,” 1969

Baudet, “Asynchronous iterative methods for multiprocessors,” 1978

Bertsekas, Tsitsiklis, “Parallel and Distributed Computation: Numerical Methods,” 1989

Frommer, Szyld, “On asynchronous iterations,” 2000 (includes nonlinear iterations)

Bahi, Contassot-Vivier, Couturier, “Parallel Iterative Algorithms: from Sequential to Grid Computing,” 2007

Recht, Re, Wright, Niu, “Hogwild: A lock-free approach to parallelizing stochastic gradient descent,” 2011 (for optimization)

Are asynchronous methods faster than synchronous methods?

Are asynchronous methods faster than synchronous methods?

If processors perform updates at exactly the same rate:

- Asynchronous and synchronous methods have the same speed

Are asynchronous methods faster than synchronous methods?

If processors perform updates at exactly the same rate:

- Asynchronous and synchronous methods have the same speed

If some processors are slower (e.g., load imbalance):

Are asynchronous methods faster than synchronous methods?

If processors perform updates at exactly the same rate:

- Asynchronous and synchronous methods have the same speed

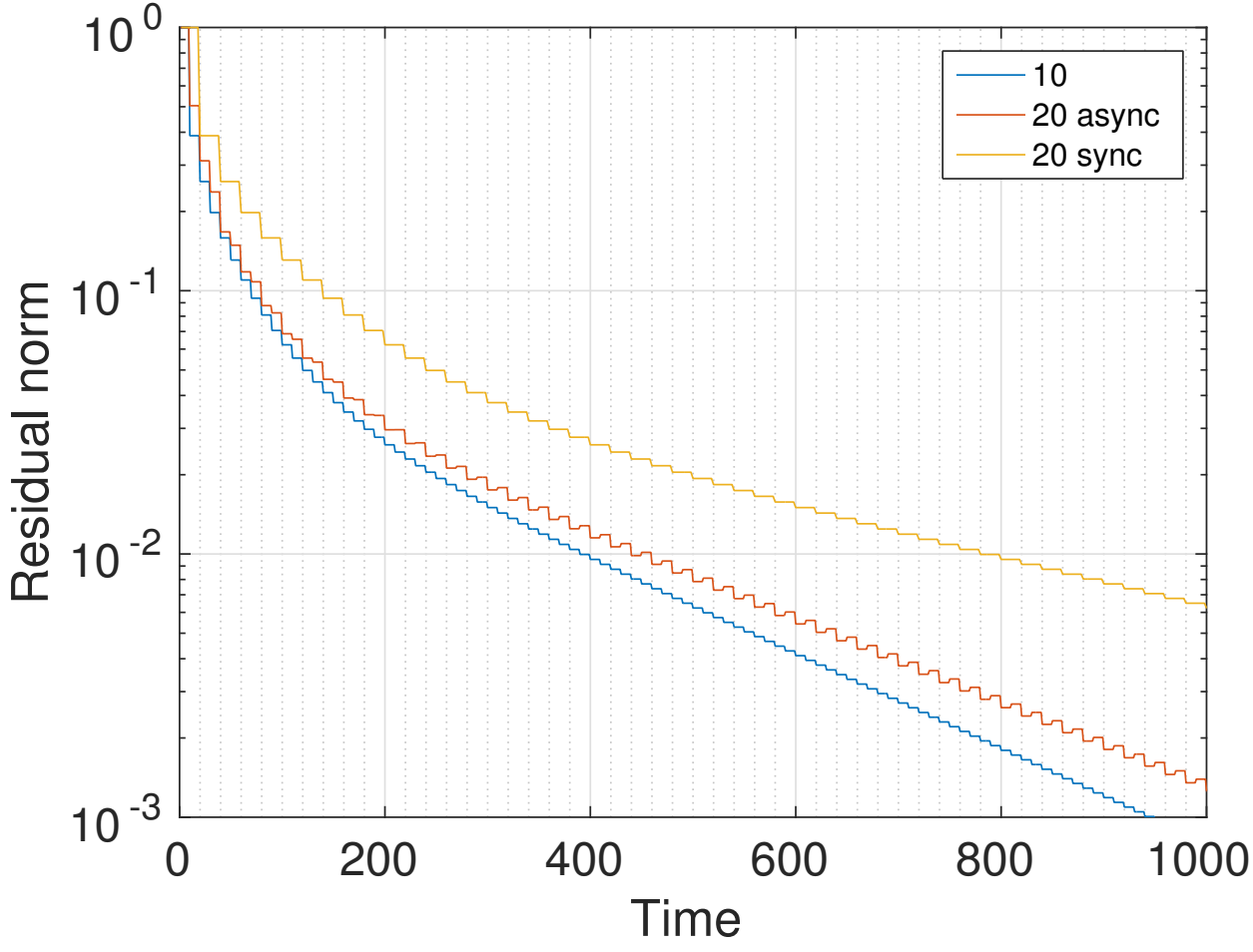
If some processors are slower (e.g., load imbalance):

- Asynchronous method is *faster* than synchronous method

Asynchronous vs. synchronous: 1 slow processor

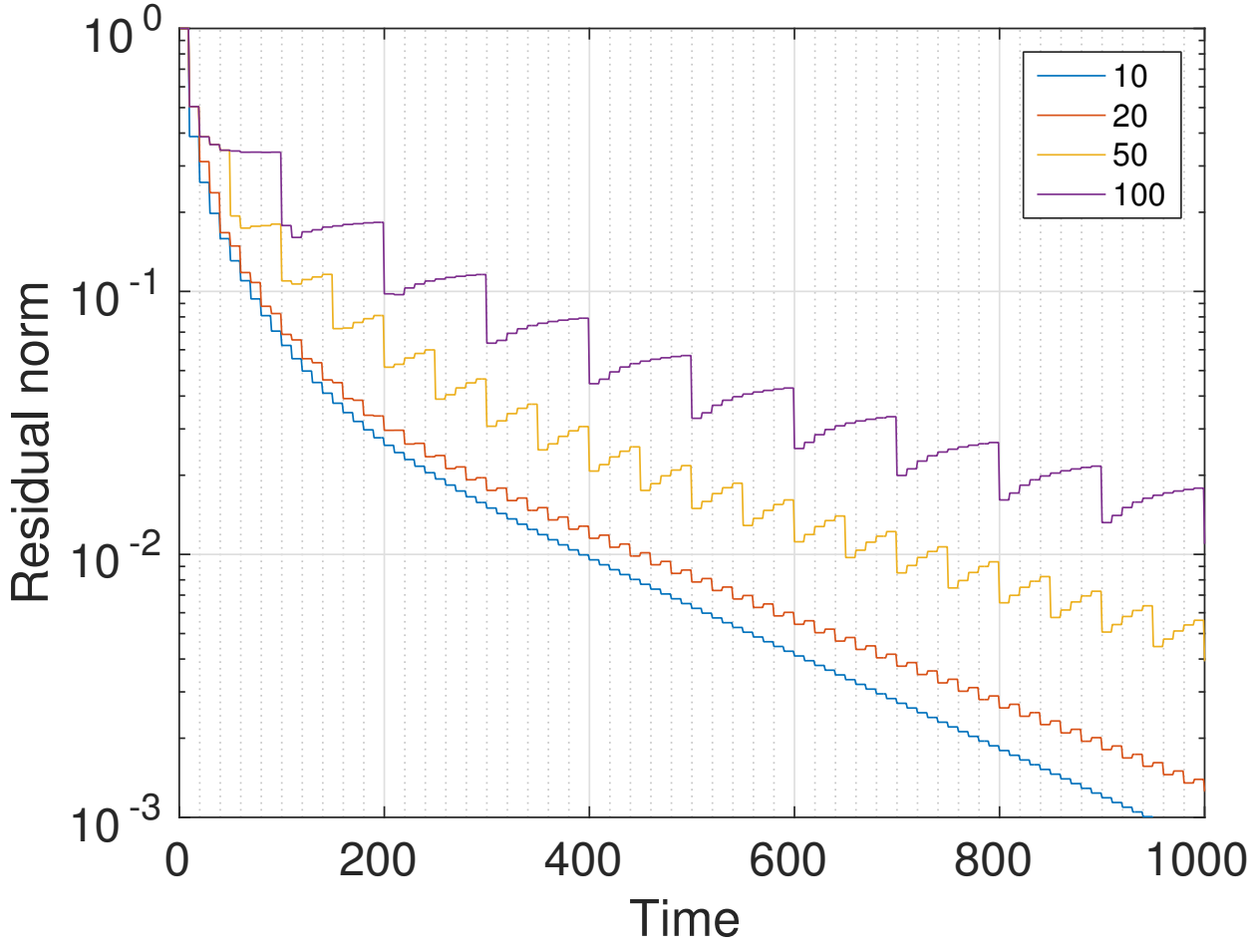
- 100 equations (finite difference Laplacian on 10×10 grid)
- 10 processors
- 9 processors each update 10 equations in 10 units of time
  (write to other processors after 10 units of time)
- 1 processor updates 10 equations in s units of time, e.g., $s = 20$
- tests are Matlab simulations

Asynchronous vs. synchronous: 1 slow processor

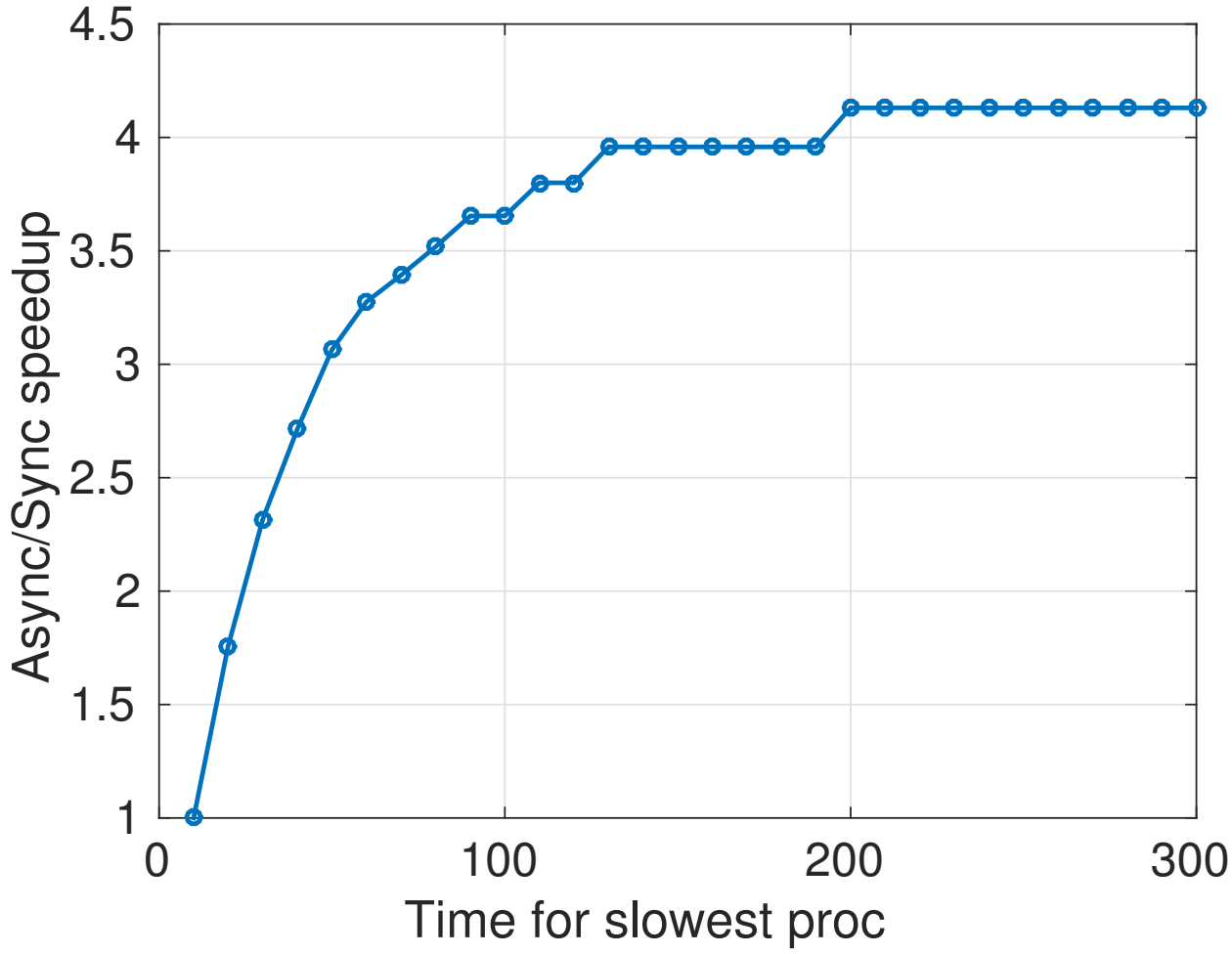


If, on the other hand, one processor is faster, the asynchronous algorithm is even faster than “10 sync”

Asynchronous time for different delays by one processor



Asynchronous vs. synchronous: speedup (fixed accuracy)

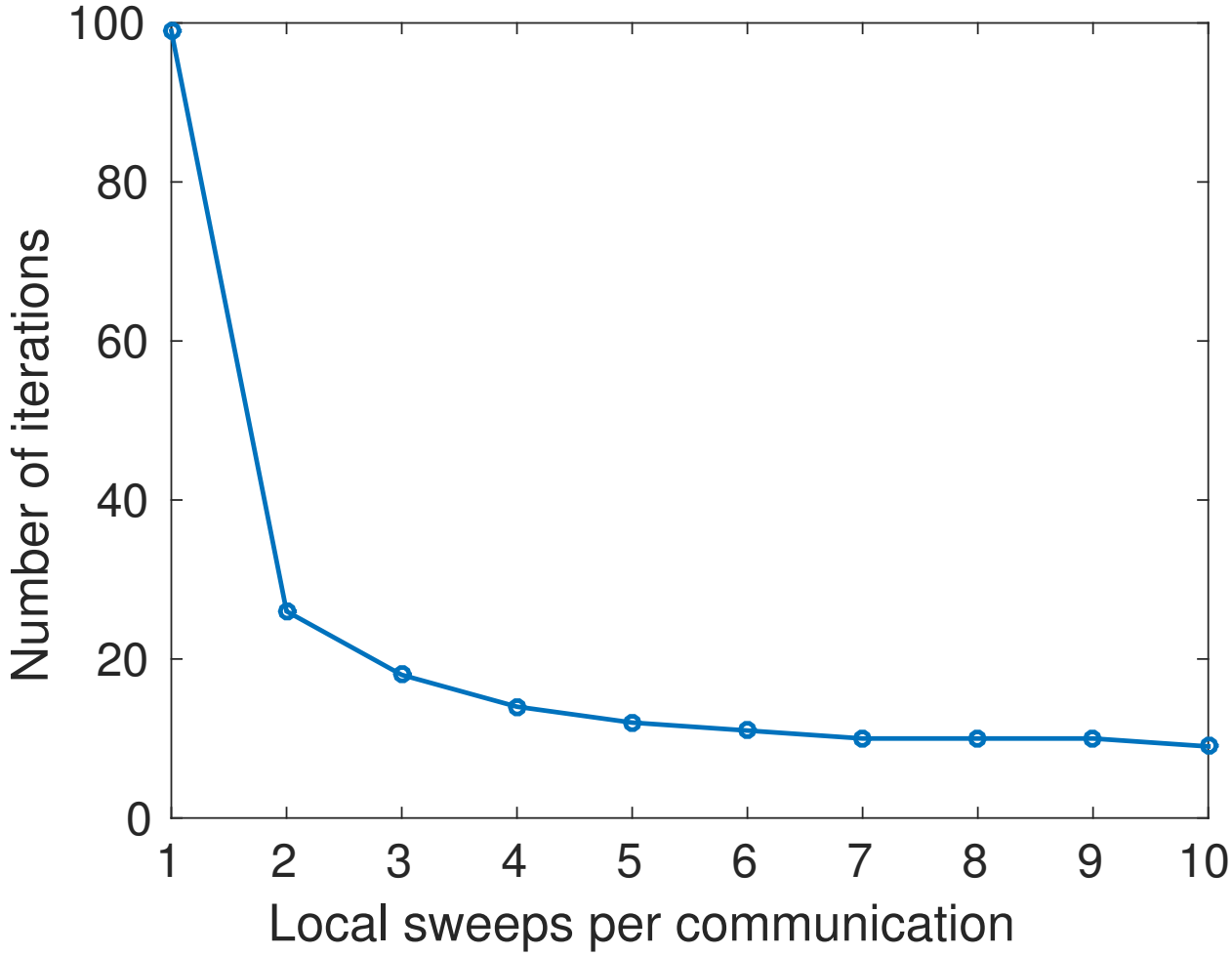


Ultimate speedup is eventually limited by the convergence on the slowest processor.

If a processor does not receive new data, does it waste computation by continuing to iterate on its local equations?

- 4096 equations (finite difference Laplacian on 64×64 grid)
- 64 processors
- Each processor performs k local sweeps before writing updated values to shared memory
- Count iterations to reach $\|r_n\|/|r_0\| < 0.05$

Additional local iterations before communicating



Additional sweeps improve convergence, but may increase total computation.

Overdecomposition for asynchronous methods

So far, we have p blocks of equations and p processors.

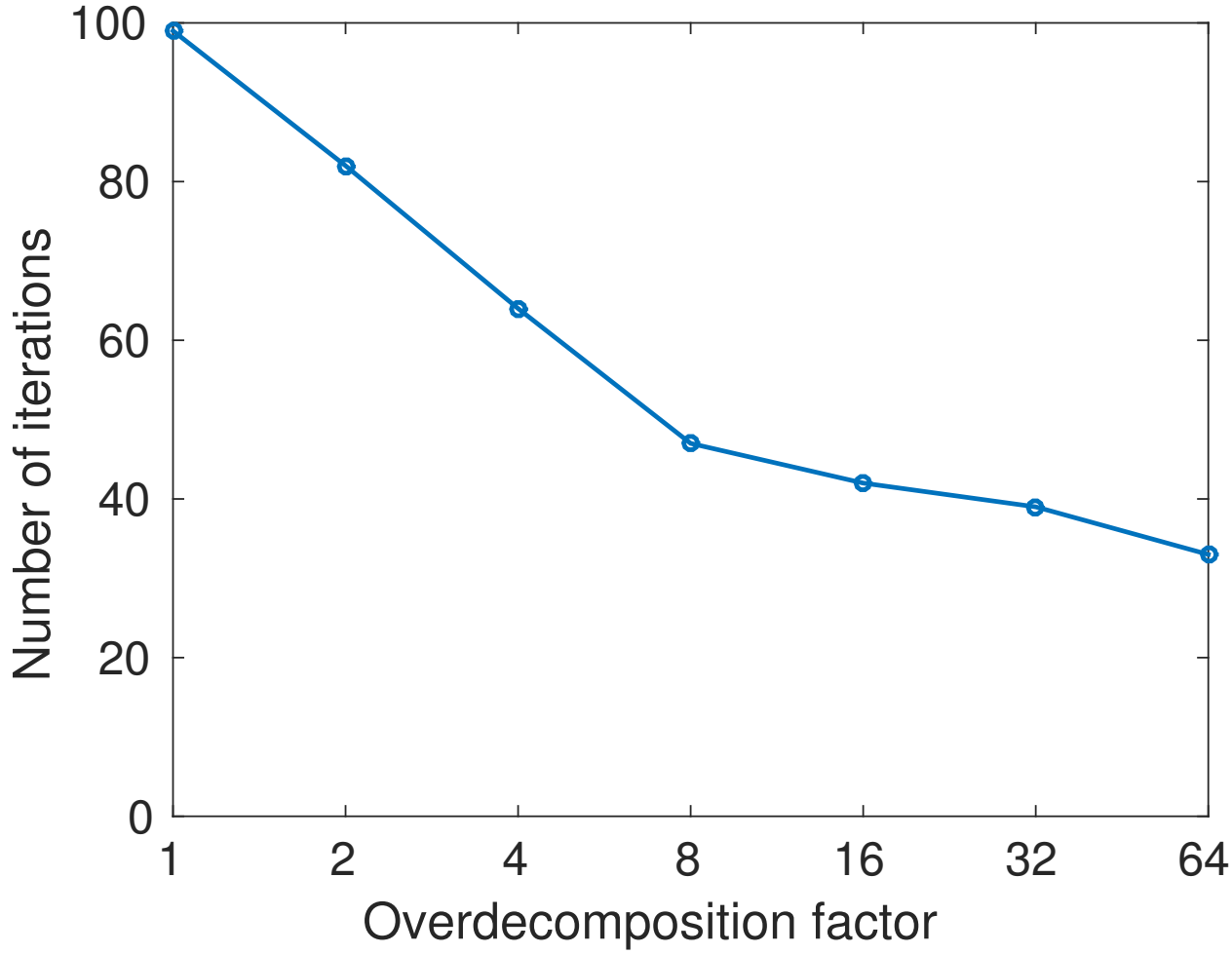
Overdecompose the problem: m blocks and p processors, with $m > p$

- Not all blocks are being updated at the same time
- Processors use latest data that is available
- Updates tend to use fresher information (like Gauss-Seidel)

Note: on GPUs, we always decompose the problem; need more thread blocks than multiprocessors (to hide memory latency).

Example: 4096 equations, 64 processors
64 blocks $\Rightarrow$ overdecomposition factor 1
128 blocks $\Rightarrow$ overdecomposition factor 2
$\vdots$
4096 blocks $\Rightarrow$ overdecomposition factor 64

Overdecomposition



Distributed parallel implementation issues

Need two features for efficiency:

- Remote memory access (RMA): process can get/put data on a target process without interaction from the target
- Asynchronous completion: not all RMA functions (e.g., remote accumulate functions) may be supported by the hardware, and software support may be needed

Gerstenberger, Besta, Hoefler, “Enabling highly-scalable remote memory access programming with MPI-3 one sided,” 2013

Si, Peña, Hammond, Balaji, Takagi, Ishikawa, “Casper: an asynchronous progress model for MPI RMA on many-core architectures,” 2015

Magoulès, Gbikpi-Benissan, Venet, “JACK: an asynchronous communication kernel library for iterative algorithms,” 2016

Optimized Schwarz method

Interlude: Krylov subspace methods

Krylov subspace methods are effective and widely-used for solving large-scale linear systems

Dot-products (global synchronizations) in Krylov methods are very costly at high levels of parallelism

Some possible remedies:

- *s*-step and communication avoiding Krylov subspace methods: Chronopoulos, Gear 1989; Hoemmen, Demmel, 2010
- Pipelined Krylov subspace methods: Ghysels, Ashby, Meerbergen, Vanroose, 2013; Gropp $\sim$2010

Effect of hiding collective communication by `MPI_Iallreduce` is significant, especially for strong scaling (Nakajima, 2017)

Backward error is increased compared to "standard" method (Carson, Rozložník, Strakoš, et al., 2017), i.e., higher precision computation is needed

Motivation for asynchronous optimized Schwarz

Schwarz methods do not require global synchronizations, but are slow compared to Krylov subspace methods

Optimized Schwarz methods (e.g., Gander, Nataf) converge rapidly

Optimized Schwarz methods are fixed-point iterations and asynchronous versions are possible (Magoulès, Szyld, Venet, 2017)

Use Krylov subspace methods for subdomain solves

The latter is related to: McInnes, Smith, Zhang, Mills, 2014: Reduce number of outer Krylov iterations with inner iterations also using Krylov methods

Test problem

2-D isotropic diffusion problem

$$\nabla^2 u = f \quad \text{in } \Omega = (0,1) \times (0,1)$$
$$u = 0 \quad \text{on } \partial\Omega$$

discretized with (1) centered finite differences or (2) linear triangular finite elements (irregular mesh).

Discrete f chosen with random components uniformly in a range with mean zero.

Optimized Schwarz

Subdomain problems are solved with transmission conditions that involve a parameter, e.g., OO0 transmission conditions:

$$\frac{\partial u^{(1)}}{\partial n_1} + \alpha u^{(1)} = \frac{\partial u^{(2)}}{\partial n_1} + \alpha u^{(2)}$$

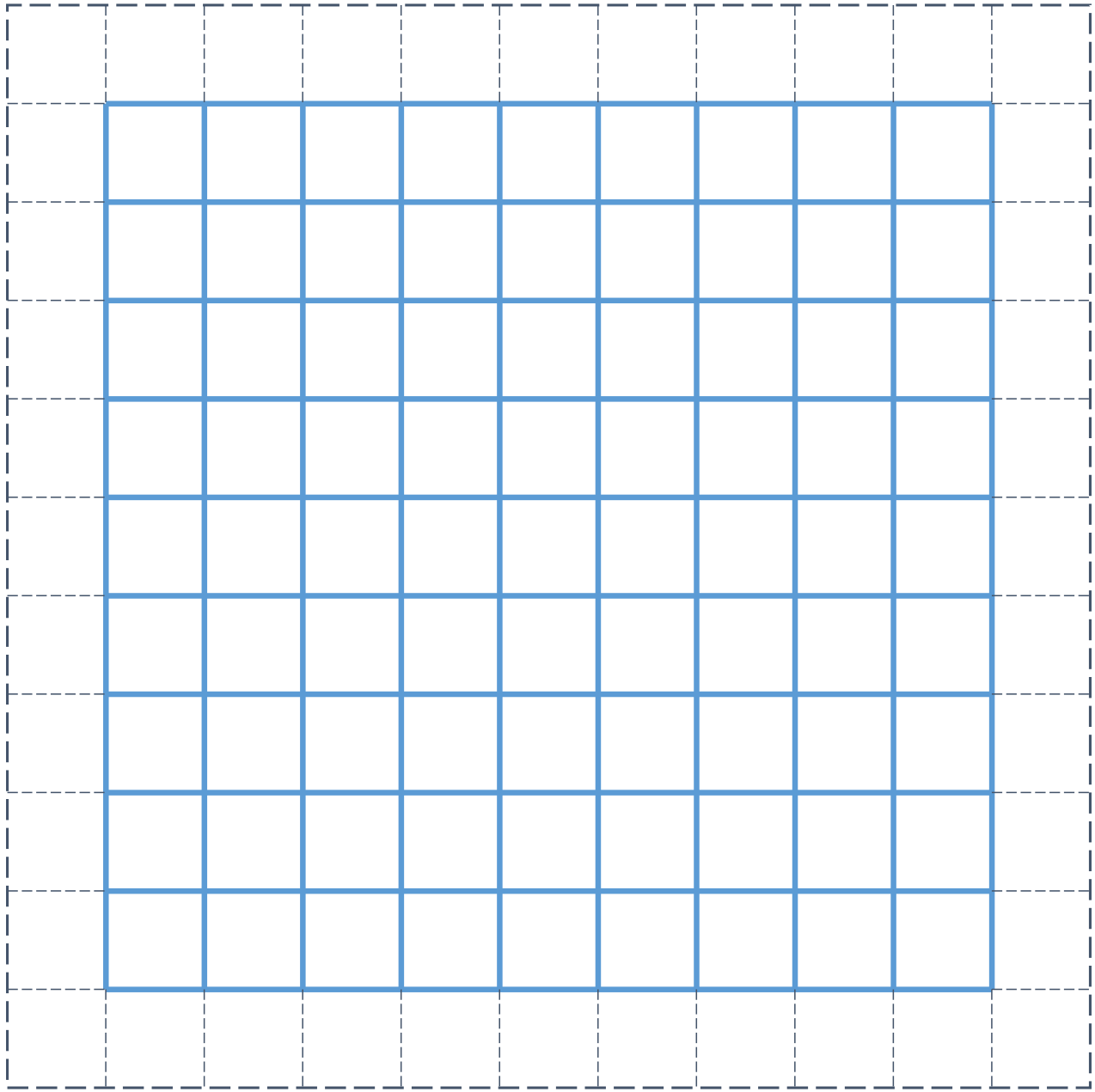
for subdomains (1) and (2).

Parameter α is to be chosen to optimize convergence.

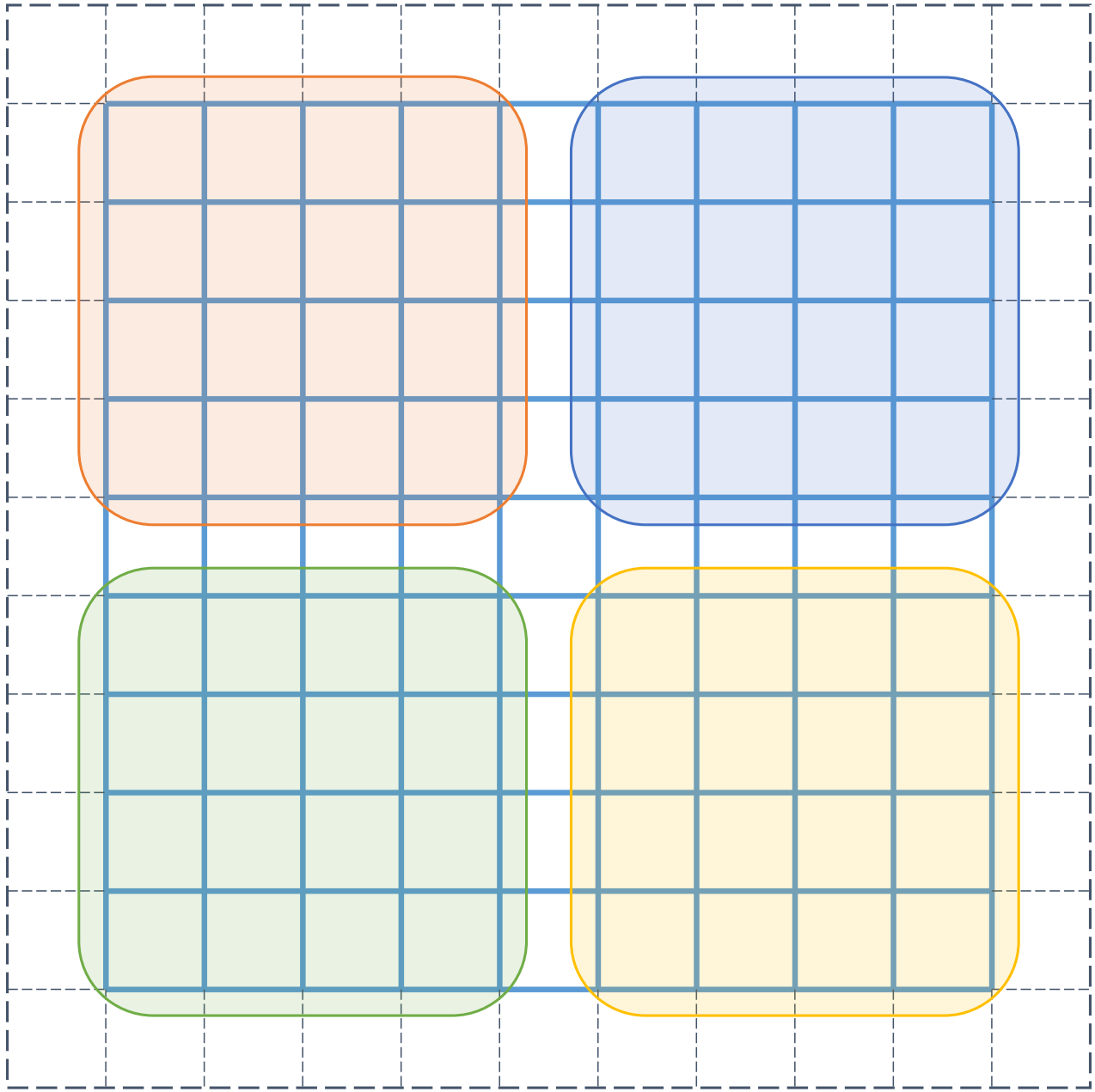
We use optimized restricted additive Schwarz.

See references in:
Dolean, Jolivet, Nataf 2015

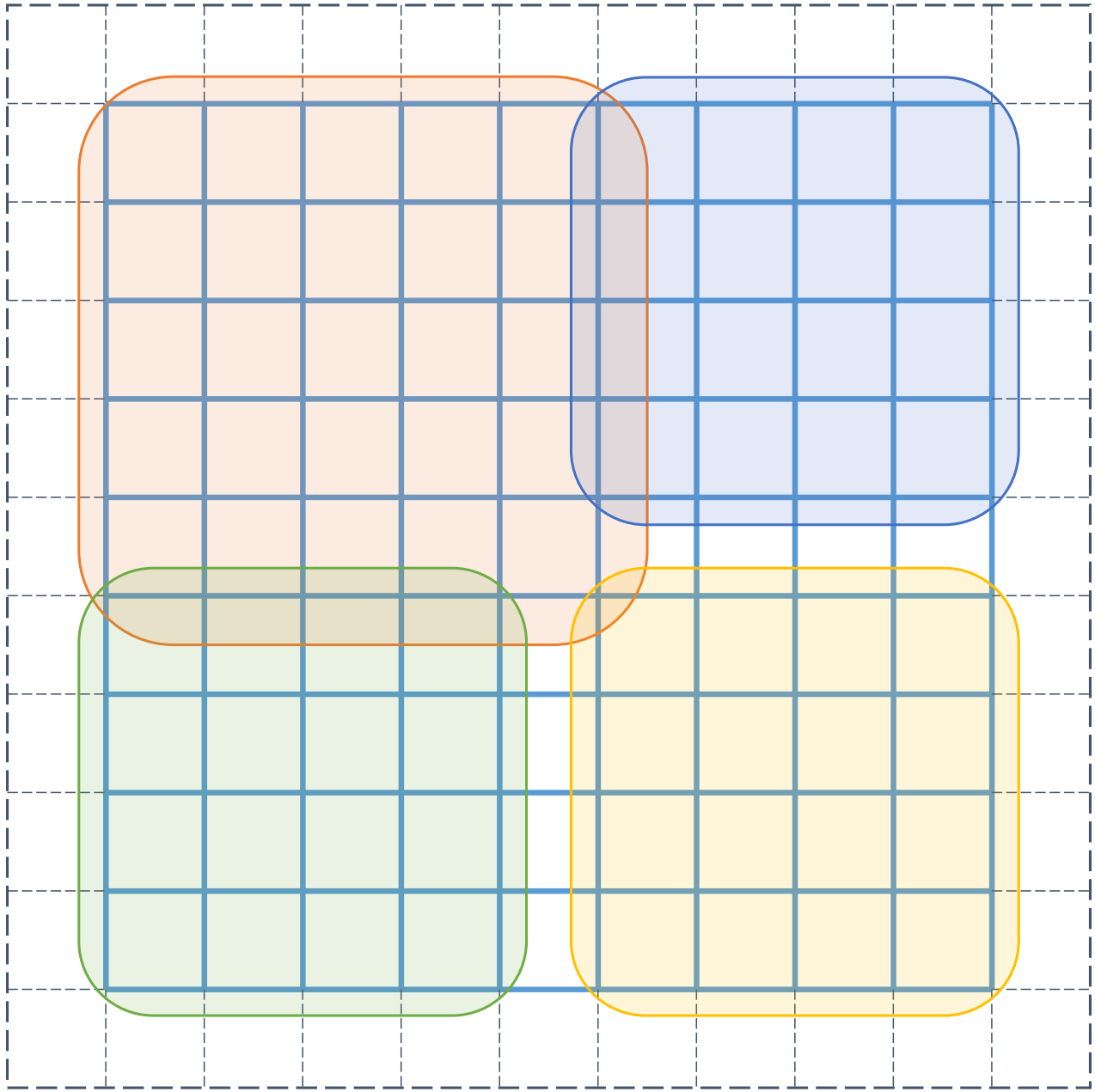
Finite difference mesh



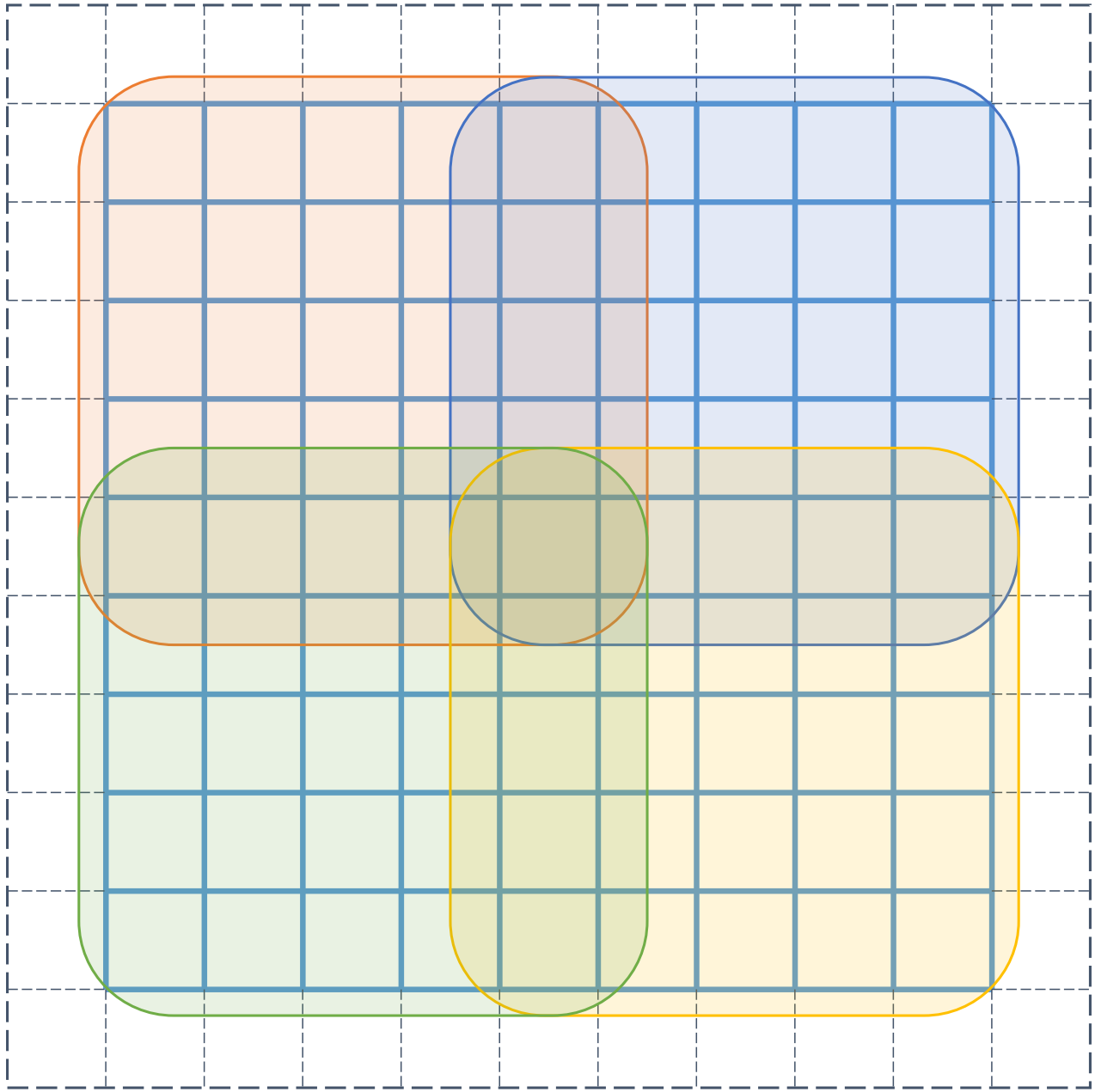
Unknowns partitioned without overlap



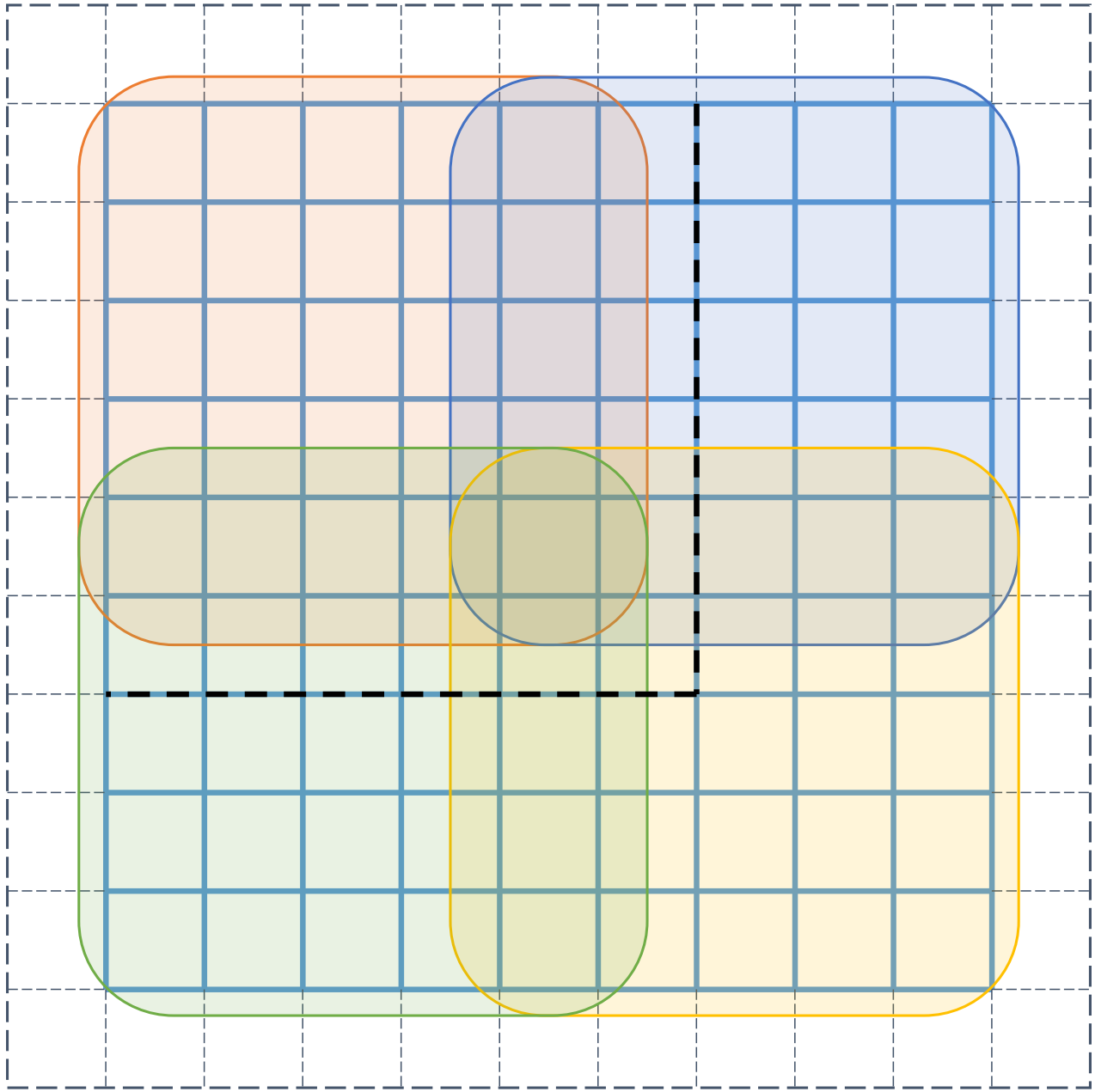
First subdomain is extended



All subdomains extended



Interior boundary for first subdomain



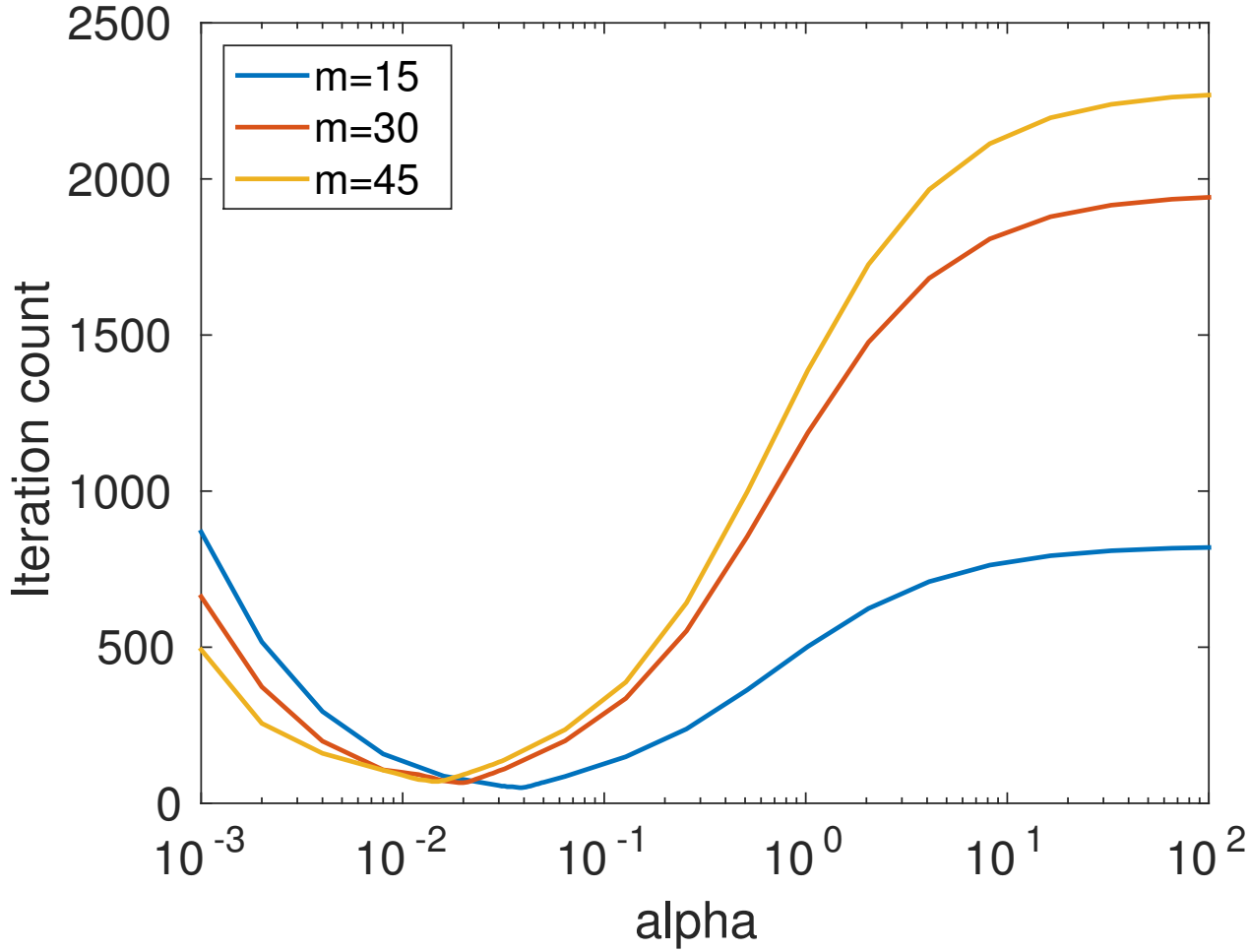
Optimized Schwarz results

2-D FD Laplacian on $p \times p$ processor grid
($p = 10$ in most experiments).

Local problem size is nominally $m \times m$ (unextended dimensions of interior subdomain).

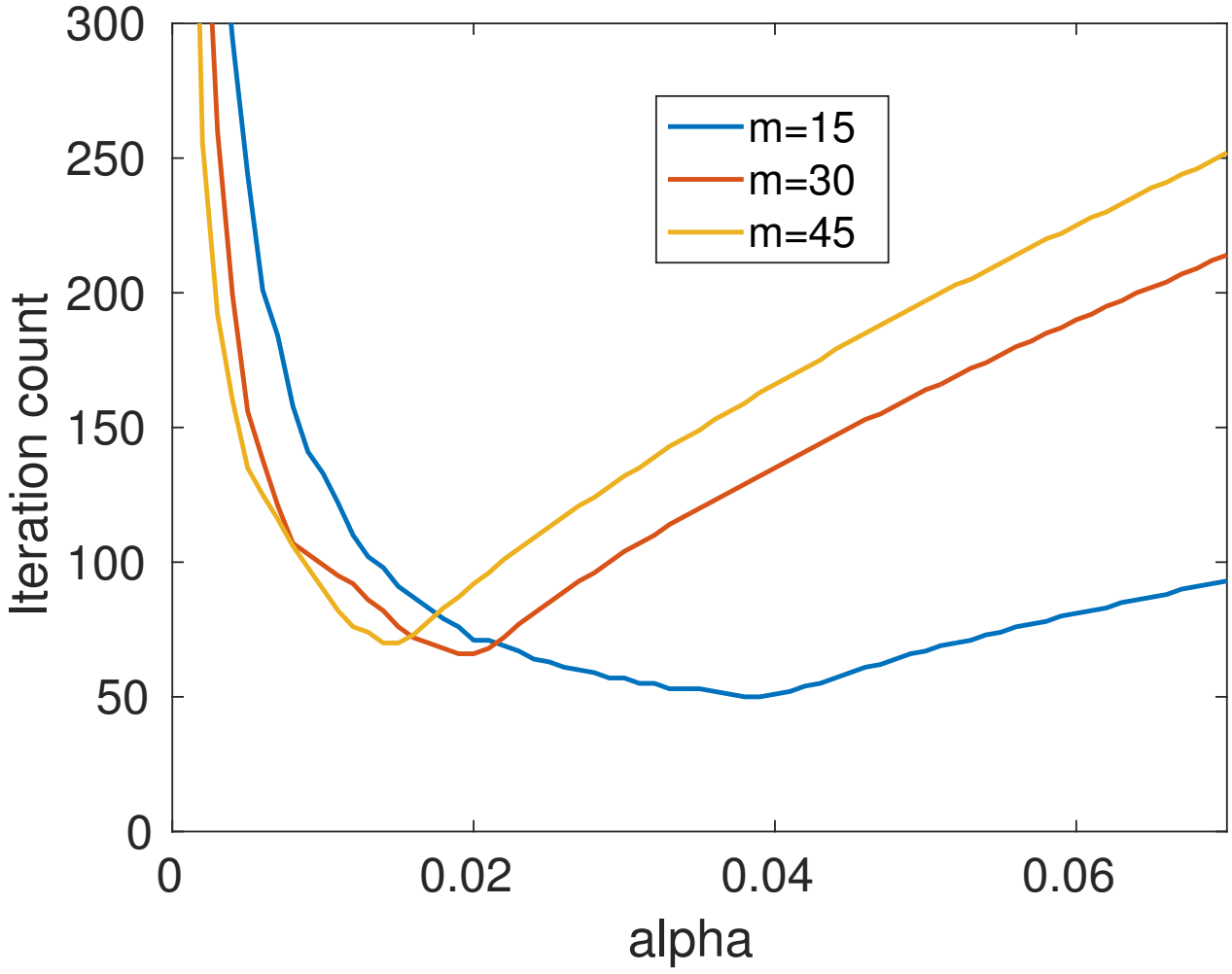
“Minimum overlap” used unless otherwise indicated.

Optimized Schwarz



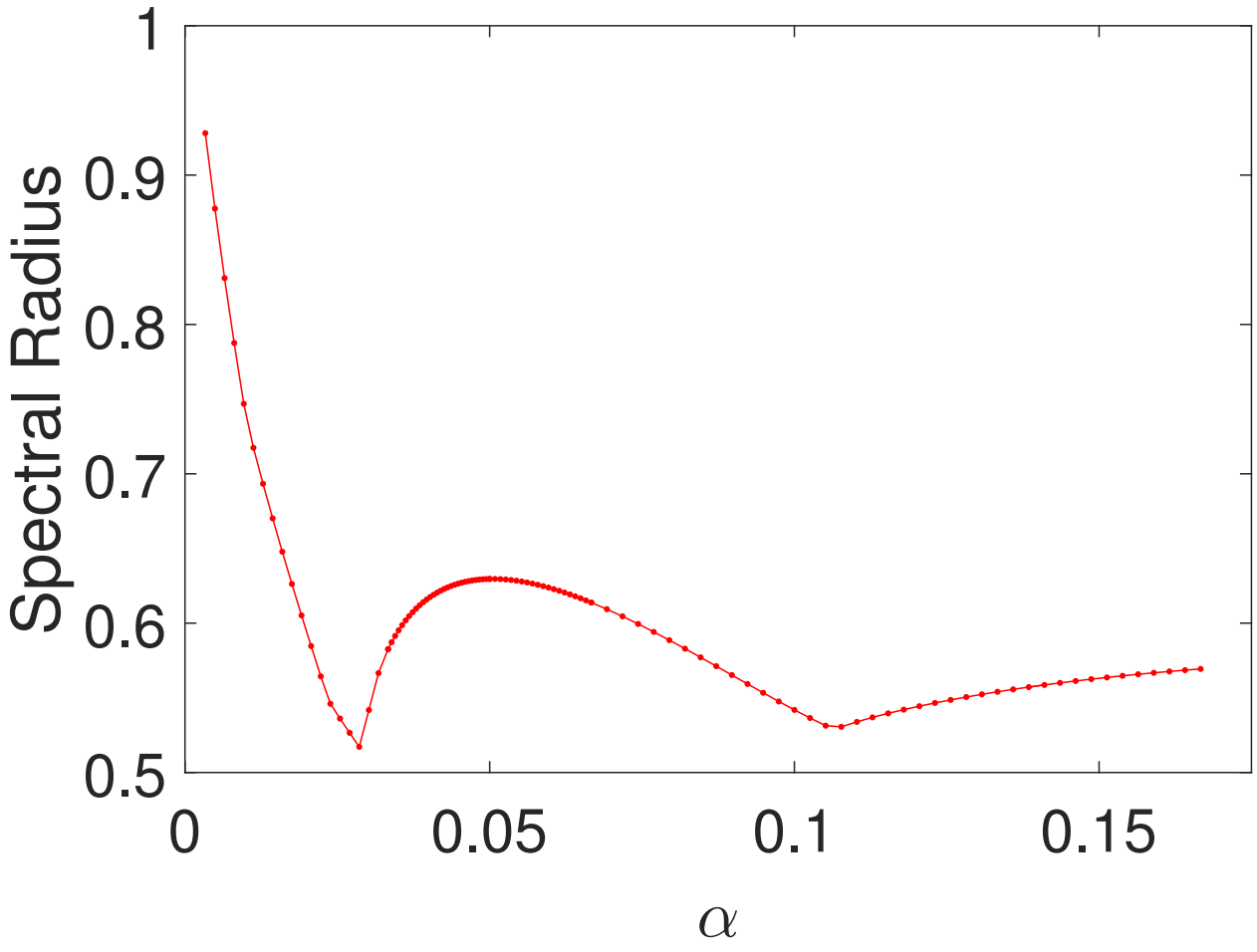
Optimal iteration counts much smaller than for classical Schwarz and are many times smaller than those of IC(0)-PCG.

Optimized Schwarz



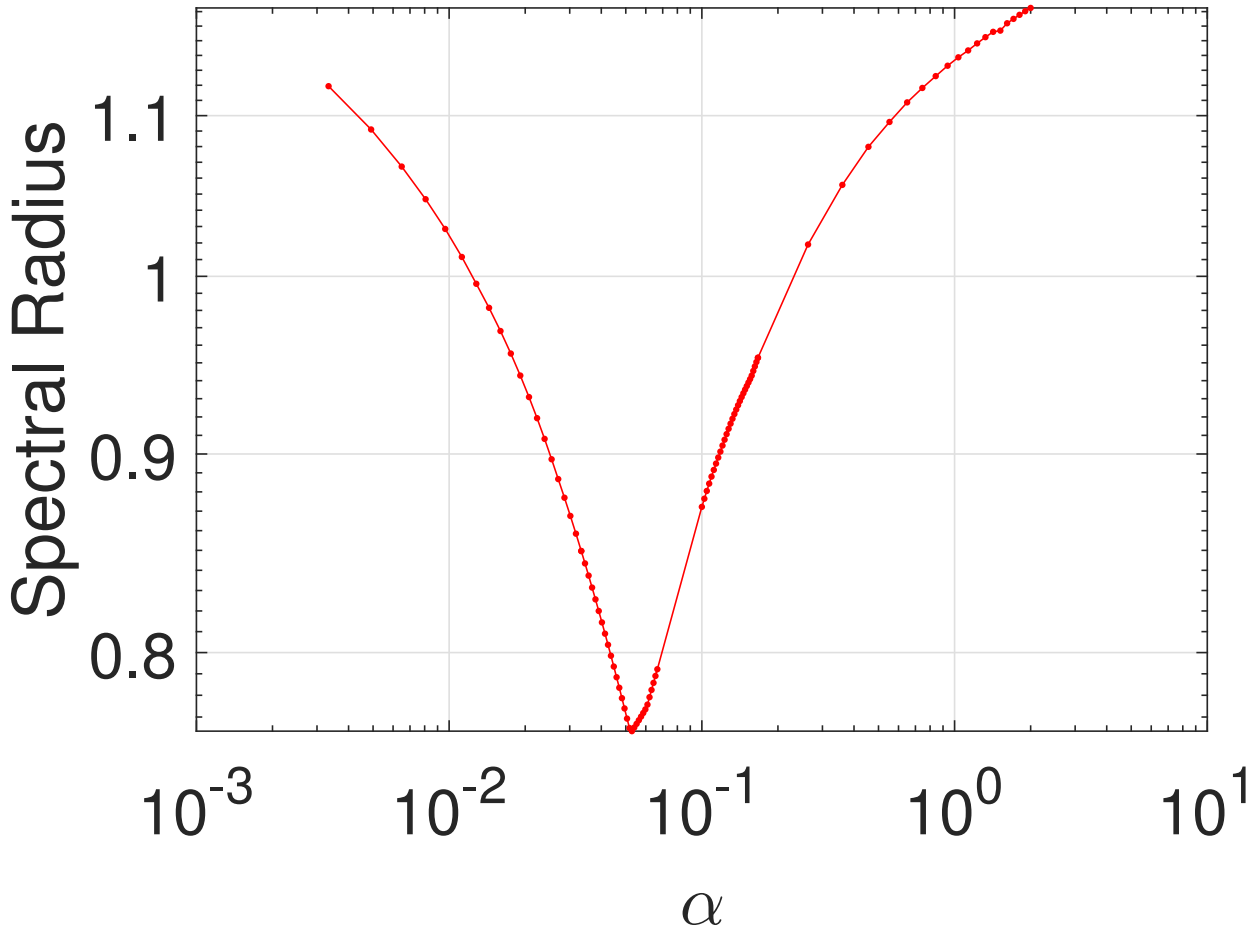
Relatively slow increase in optimal iteration count with problem size.

Analysis of spectral radius: synchronous case, $m = 30$



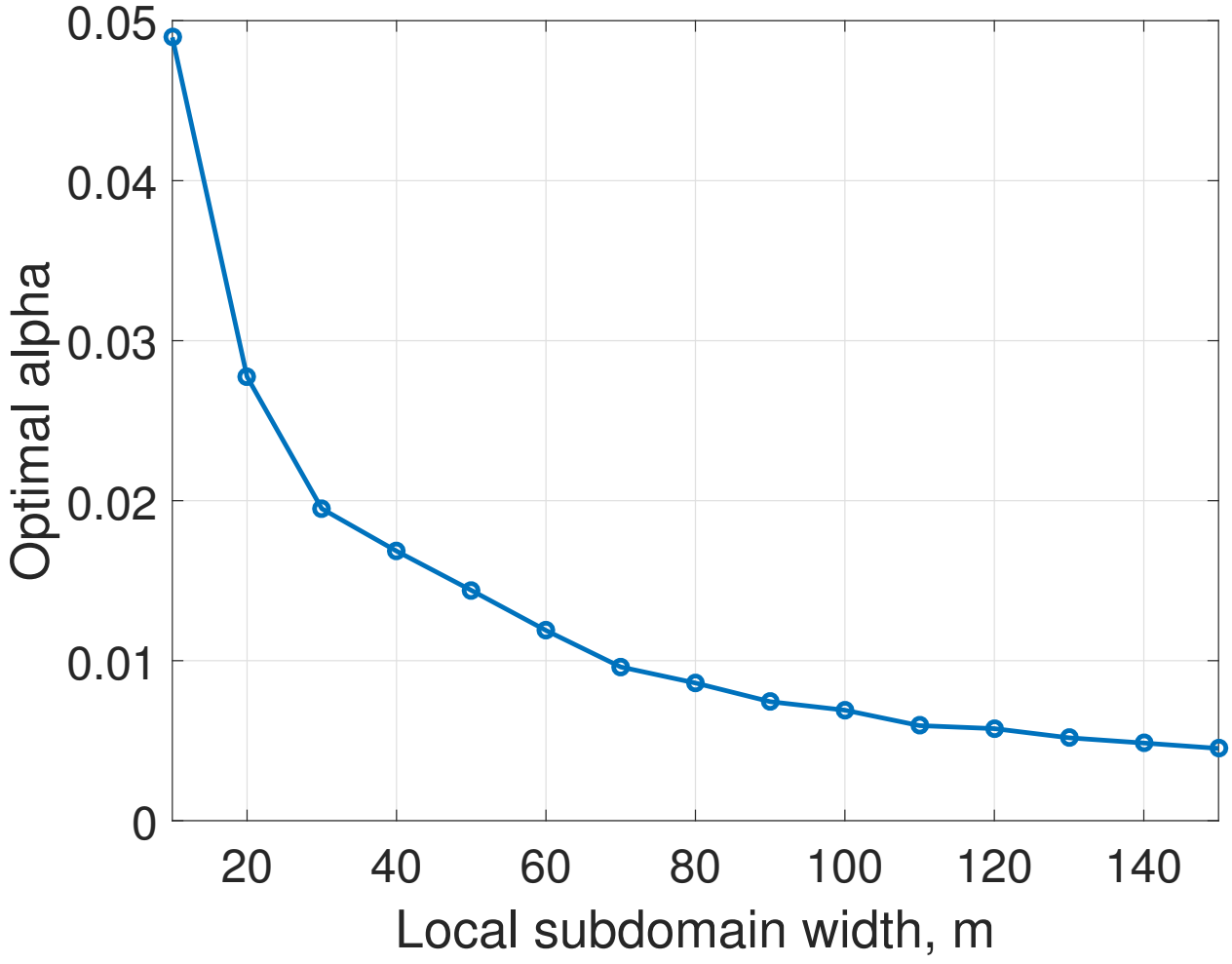
(Garay, Szyld, Magoulès, 2017)

Analysis of spectral radius: asynchronous case, $m = 30$

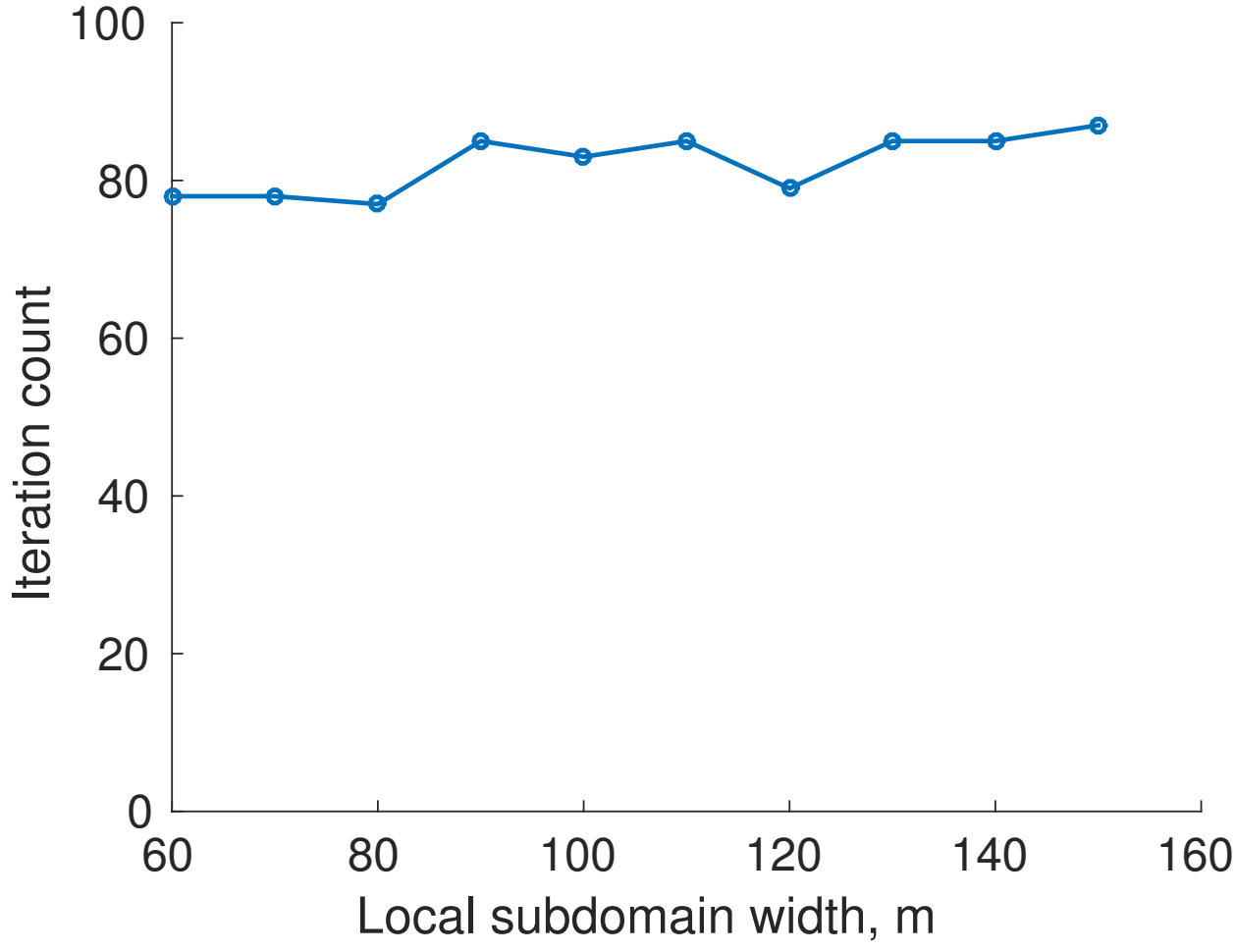


(Garay, Szyld, Magoulès, 2017)

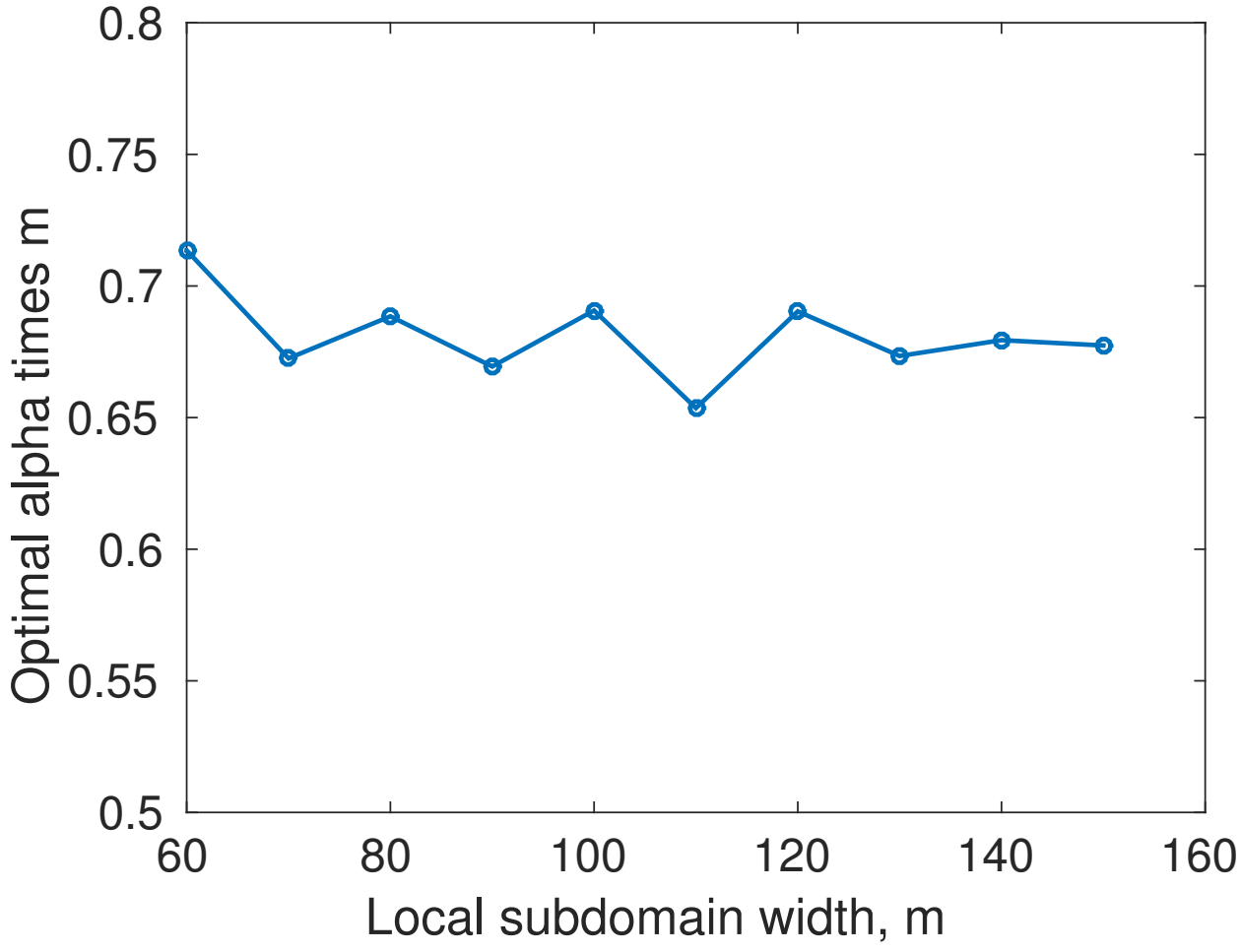
Optimized Schwarz: increasing mesh resolution



Optimized Schwarz: strong scaling iteration counts



Relation between α_{opt} and m



$\alpha \times m$ is dimensionless; similar relation between α_{opt} and p

Challenges

- Optimizing the α parameters for each interface for complicated problems
- Application codes typically to not provide derivative information to solvers
- Incorporating a coarse grid solve in asynchronous fashion

Parallel and distributed Southwell methods

Gauss-Seidel and Southwell Relaxation

For solving $Au = f$

Gauss-Seidel Relaxation

At each step, choose the "next" equation to relax using a pre-specified ordering of all the equations.

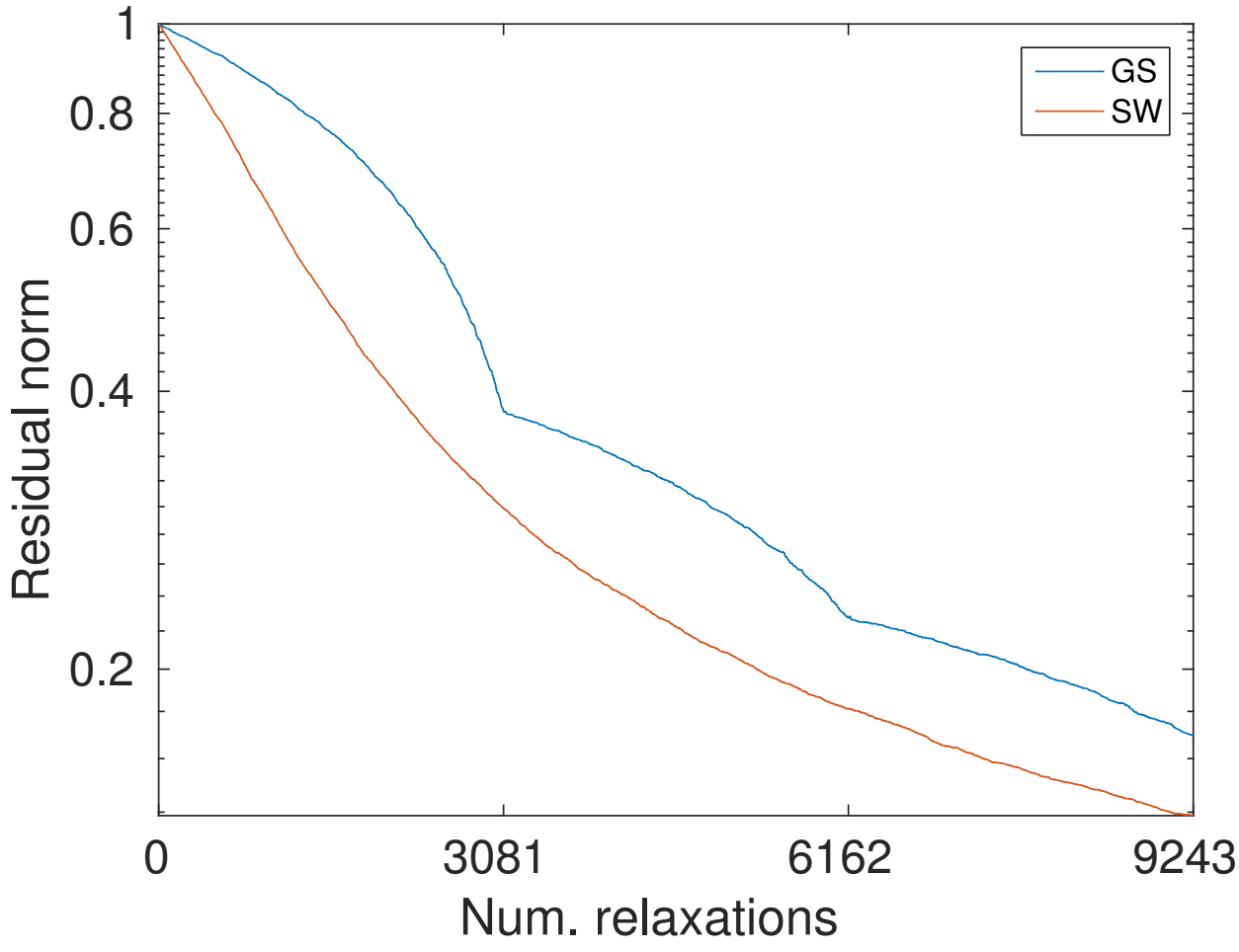
Southwell Relaxation

At each step, relax the equation with the largest residual; equations can be relaxed multiple times before others are relaxed. Sequential algorithm by definition.

R. V. Southwell, *Relaxation Methods in Engineering Science, a Treatise on Approximate Computation*, Oxford Univ. Press, 1940

R. V. Southwell, *Relaxation Methods in Theoretical Physics*, Oxford Univ. Press, 1946

Southwell vs. Gauss-Seidel (FEM problem)



Southwell uses half as many relaxations when low accuracy needed

Parallel Southwell

For simplicity, assume one-to-one assignment of equations to threads or compute nodes.

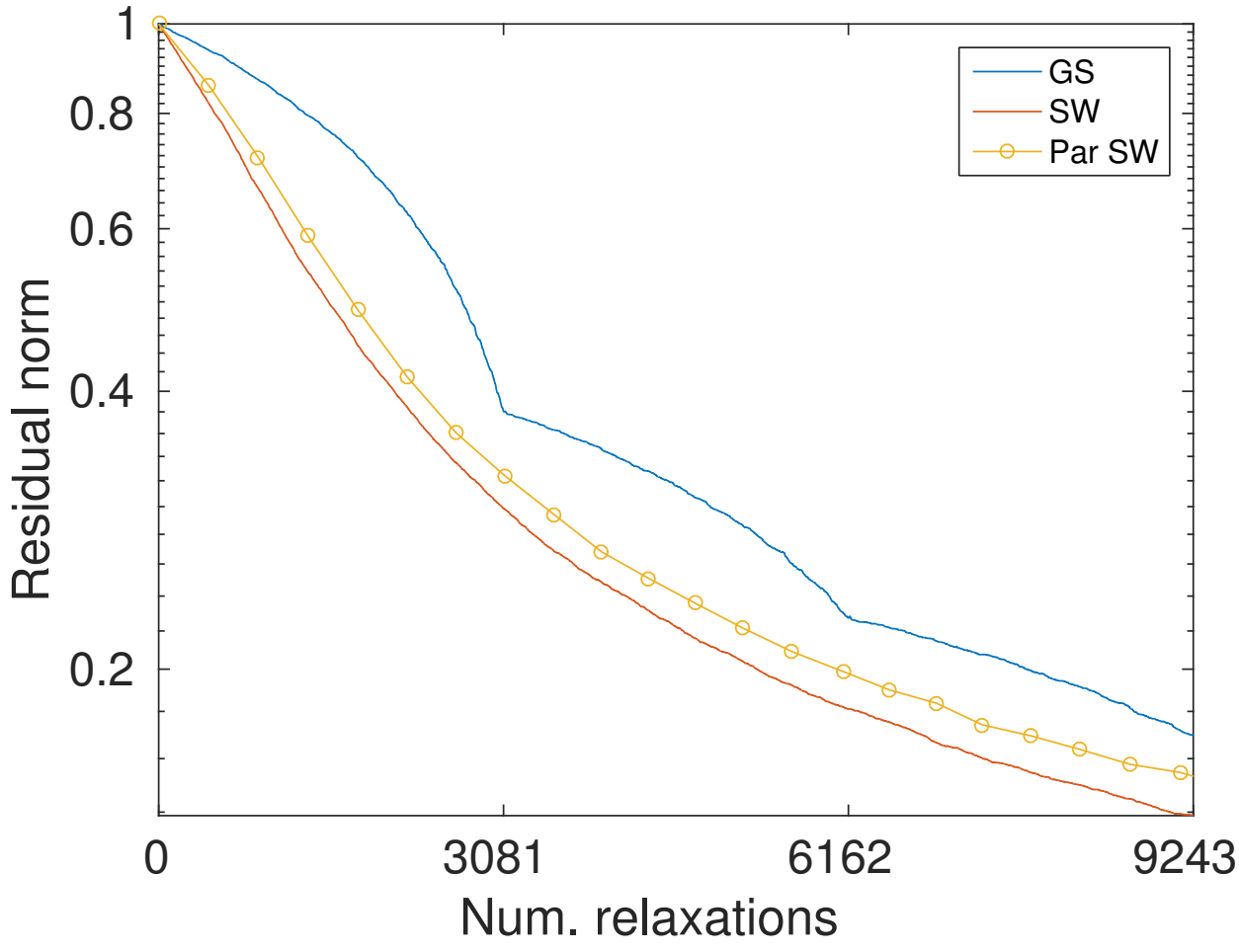
Goals:
More than one equation can be relaxed simultaneously.
Avoid global communication to determine which equations to relax.

Main idea:
An equation is relaxed if its residual is larger in size than the residuals of all its neighbors.

Neighbor = neighbor in the PDE mesh, or an equation that is directly coupled through a common variable.

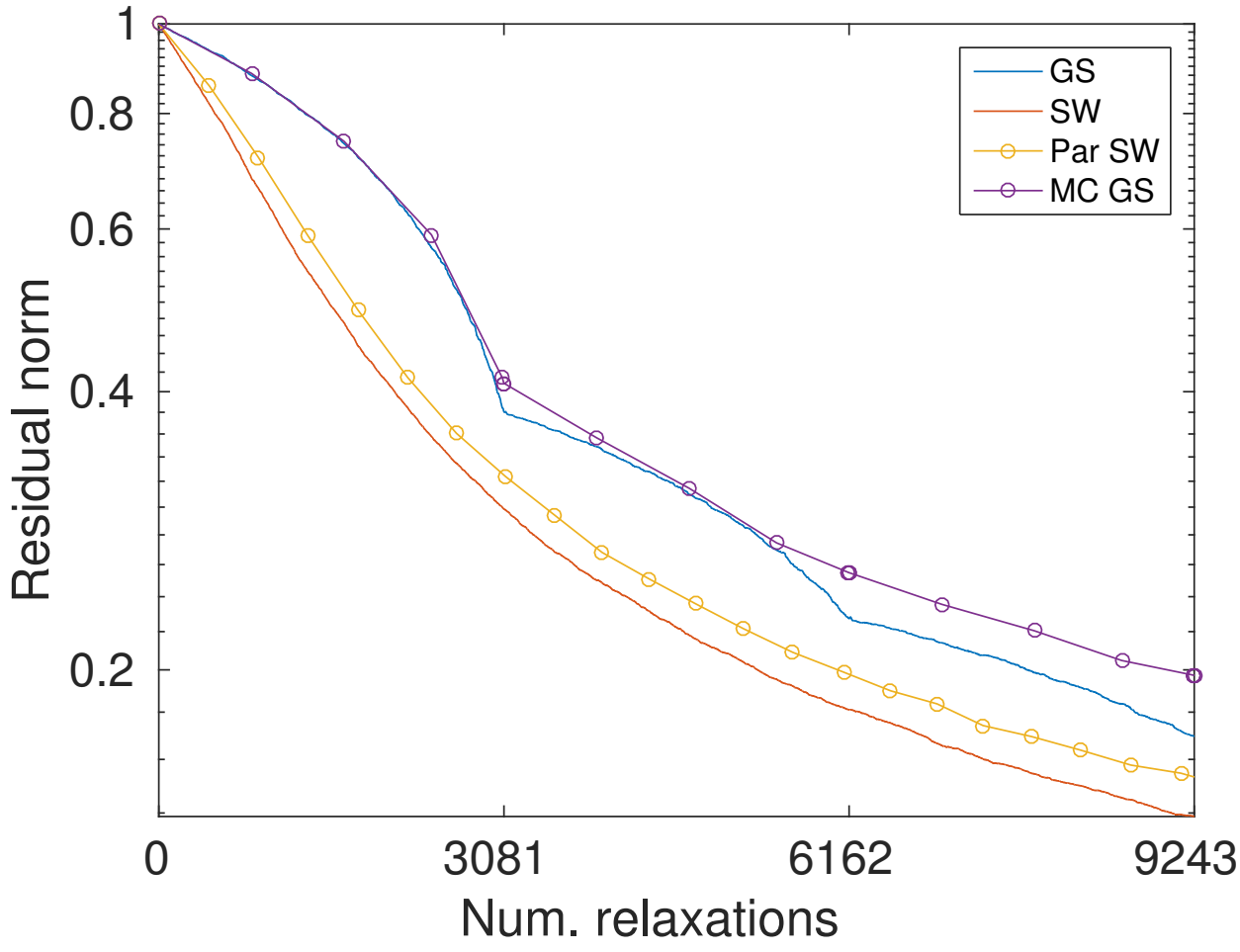
Note: Assumes the residual of neighbors can be accessed cheaply.

Parallel Southwell



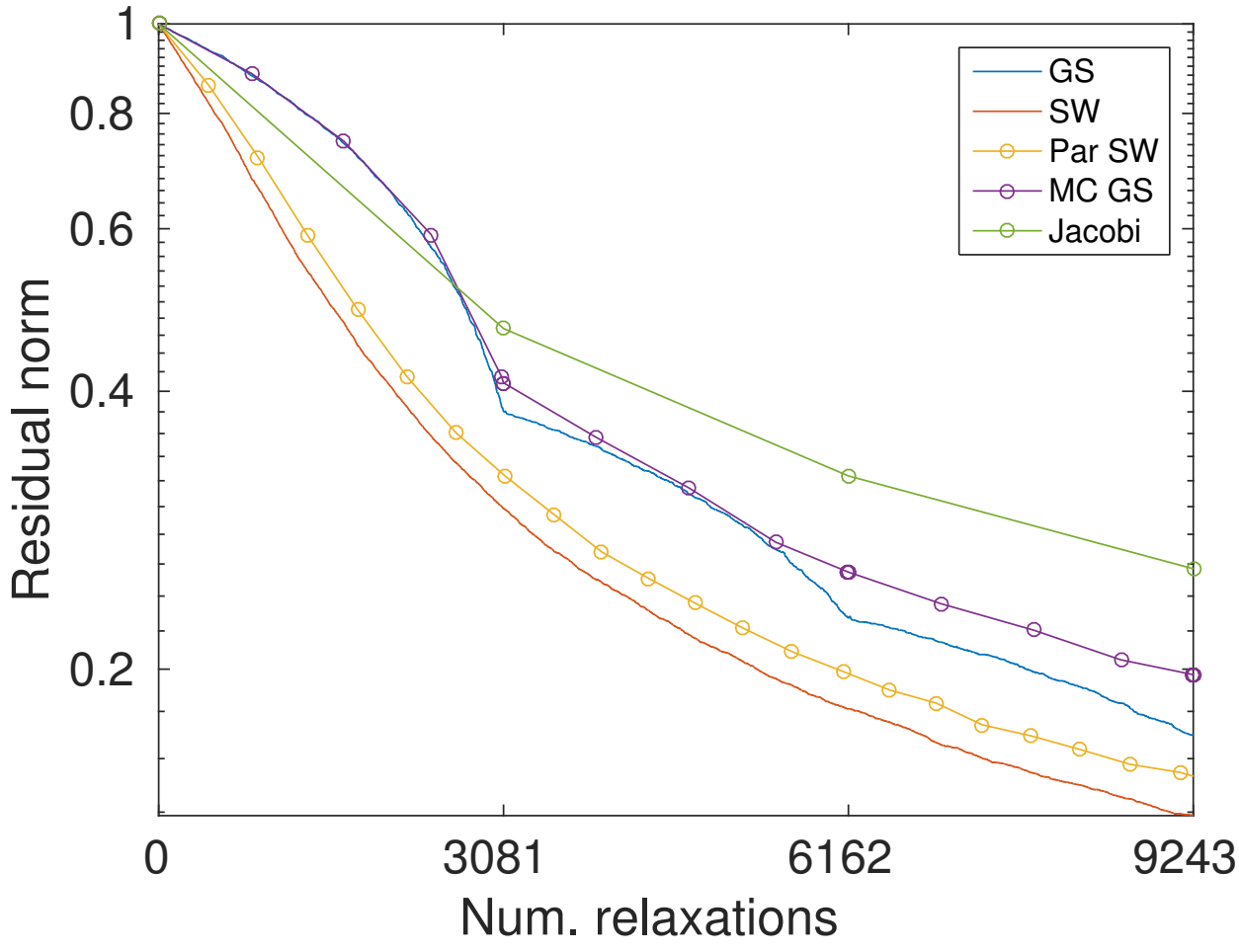
Parallel Southwell: similar number of relaxations as Southwell, but is parallel.

Parallel Southwell vs. multicolor GS



Relaxation is associated with communication; therefore Parallel Southwell does less communication than multicolor GS.

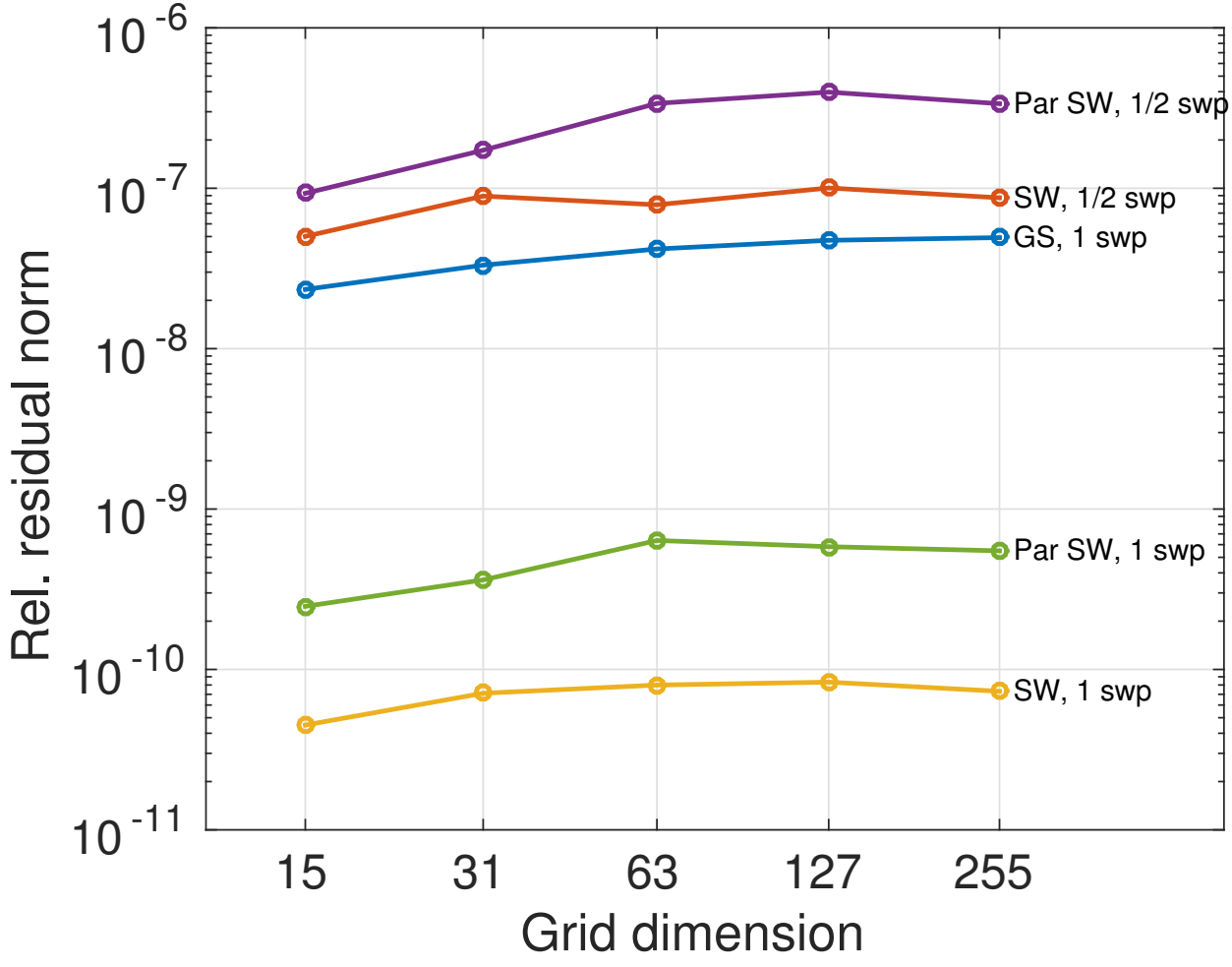
Parallel Southwell vs. Jacobi



Unlike Jacobi, Parallel Southwell is guaranteed to converge for any SPD system.

Parallel Southwell smoother

FD Laplacian problem. Residual norm after 9 V(1,1) cycles.



Related work: (sequential) Southwell method as a smoother

- Rüde, “Fully adaptive multigrid methods,” 1993.
- Griebel and Oswald, “Greedy and randomized versions of the multiplicative Schwarz method,” 2012.

Distributed Southwell

- Remove assumption that the residual on neighbor processor is cheap to access
- Use an estimate of the exact residual on neighbor processors
- Estimate of residual on j can be updated by i without communication when relaxation on i is performed

$$i \Longleftrightarrow j \Longleftrightarrow k$$

but estimates get out of date when other relaxations affect j

Distributed Southwell

- Remove assumption that the residual on neighbor processor is cheap to access
- Use an estimate of the exact residual on neighbor processors
- Estimate of residual on j can be updated by i without communication when relaxation on i is performed

$$i \Longleftrightarrow j \Longleftrightarrow k$$

  but estimates get out of date when other relaxations affect j
- Deadlock can occur when all processors think that its residual is smaller than one of its neighbors

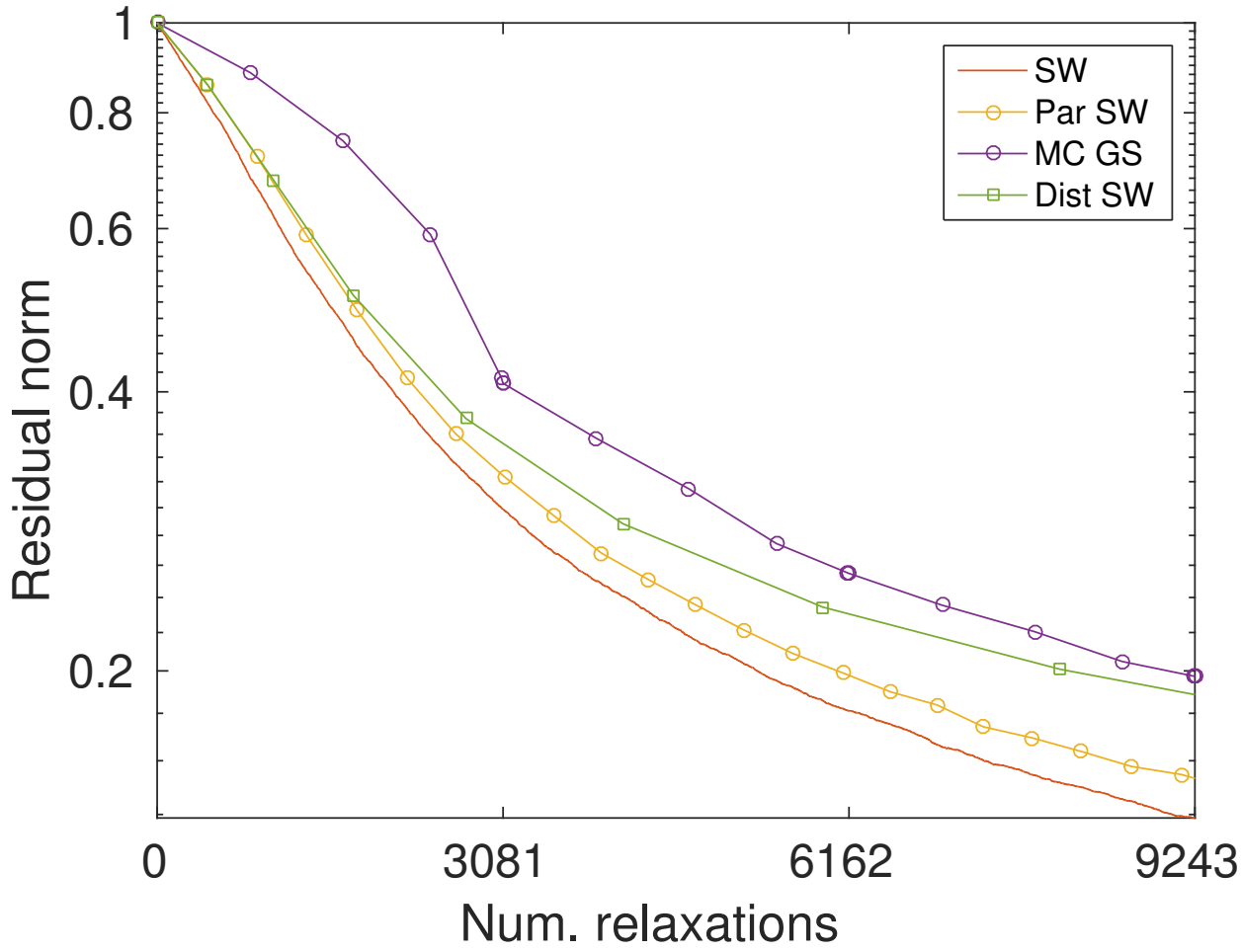
Distributed Southwell

- Remove assumption that the residual on neighbor processor is cheap to access
- Use an estimate of the exact residual on neighbor processors
- Estimate of residual on j can be updated by i without communication when relaxation on i is performed

$$i \Longleftrightarrow j \Longleftrightarrow k$$

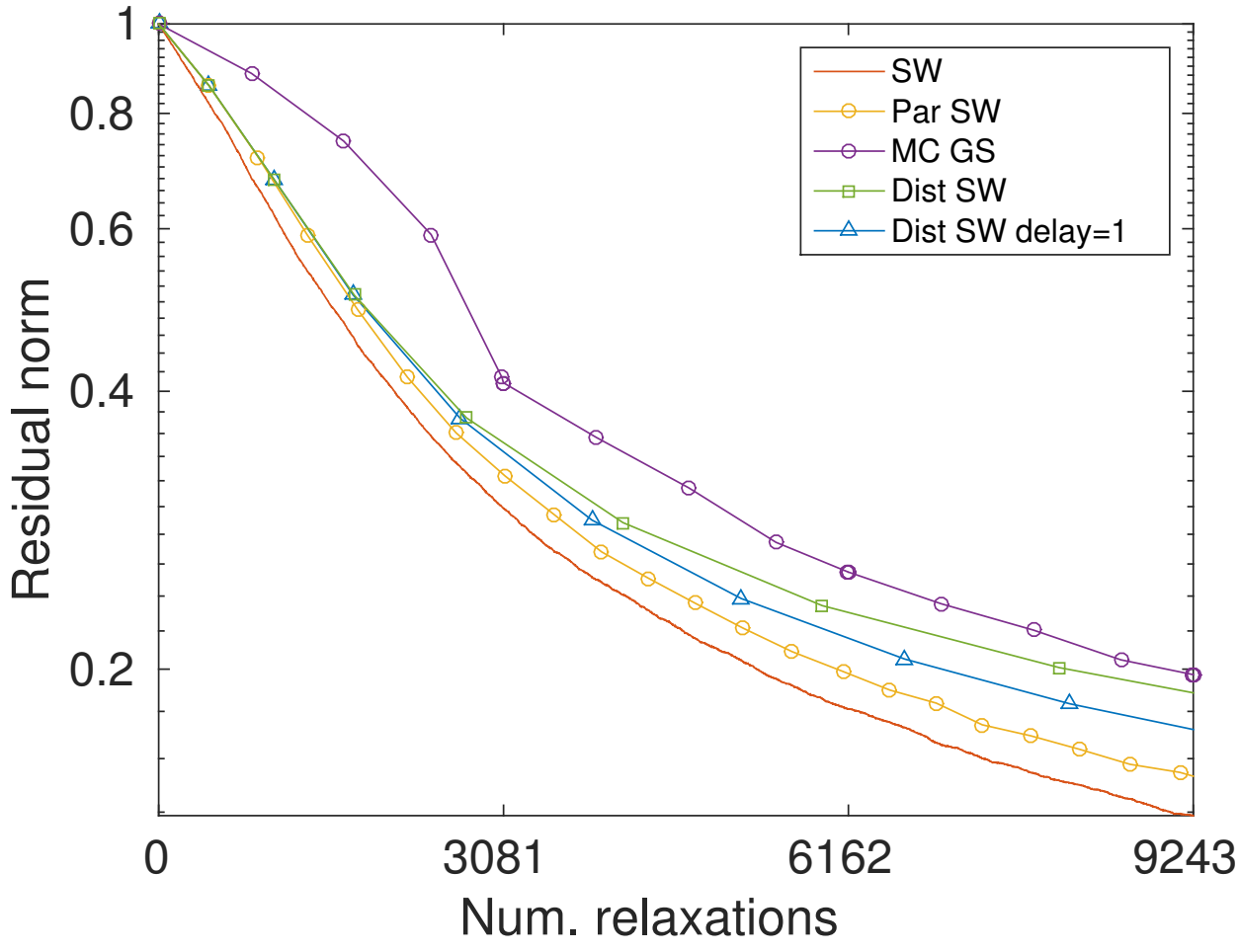
  but estimates get out of date when other relaxations affect j
- Deadlock can occur when all processors think that its residual is smaller than one of its neighbors
- To avoid deadlock, processors keep track of the estimates other processors have of its residual. If the estimate is higher than the true residual, then use communication to update the estimate
- Asynchronous implementation: Wolfson-Pou, Chow 2017

Distributed Southwell



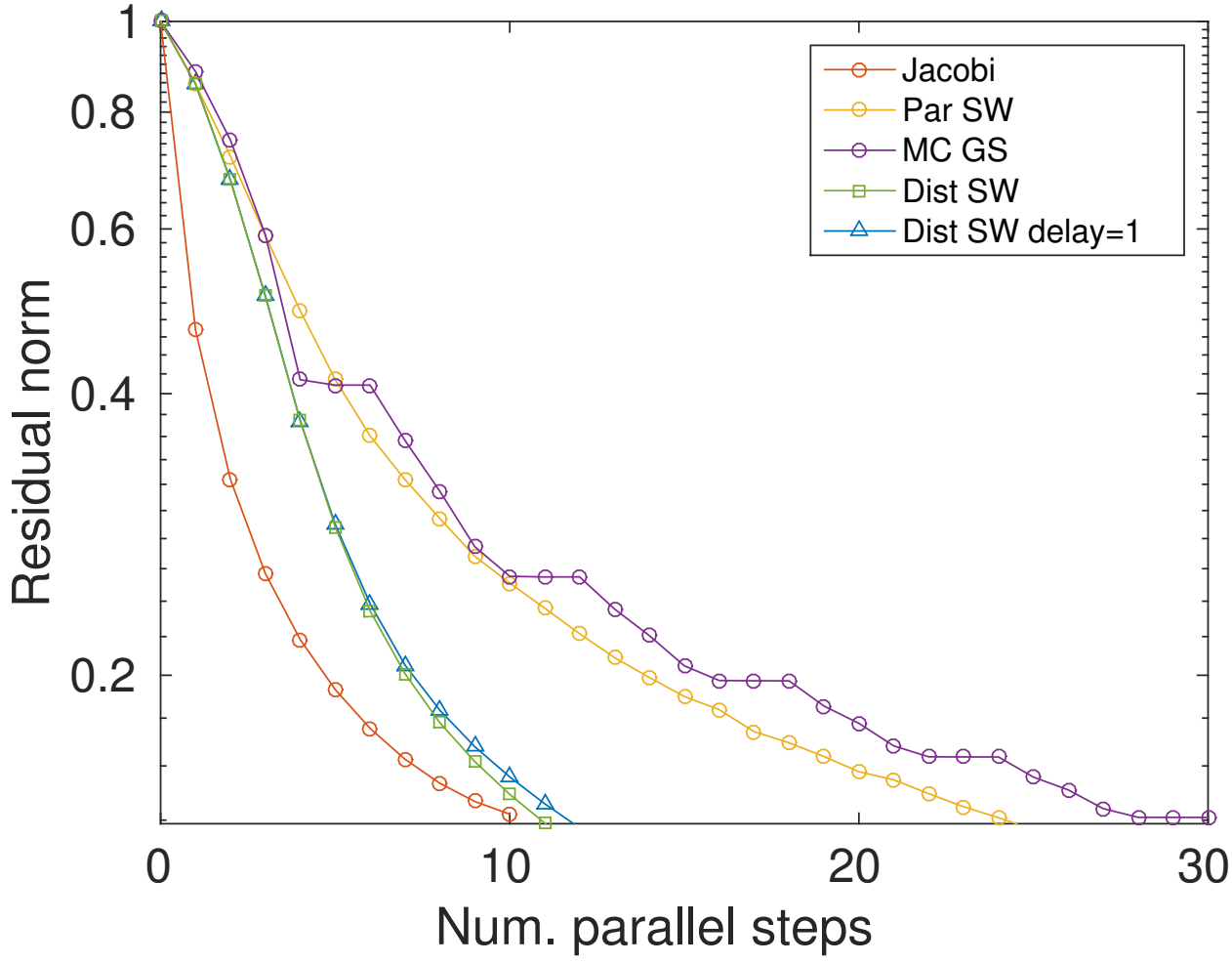
Distributed Southwell relaxes more equations per step than Parallel Southwell but converges more slowly.

Distributed Southwell with delay



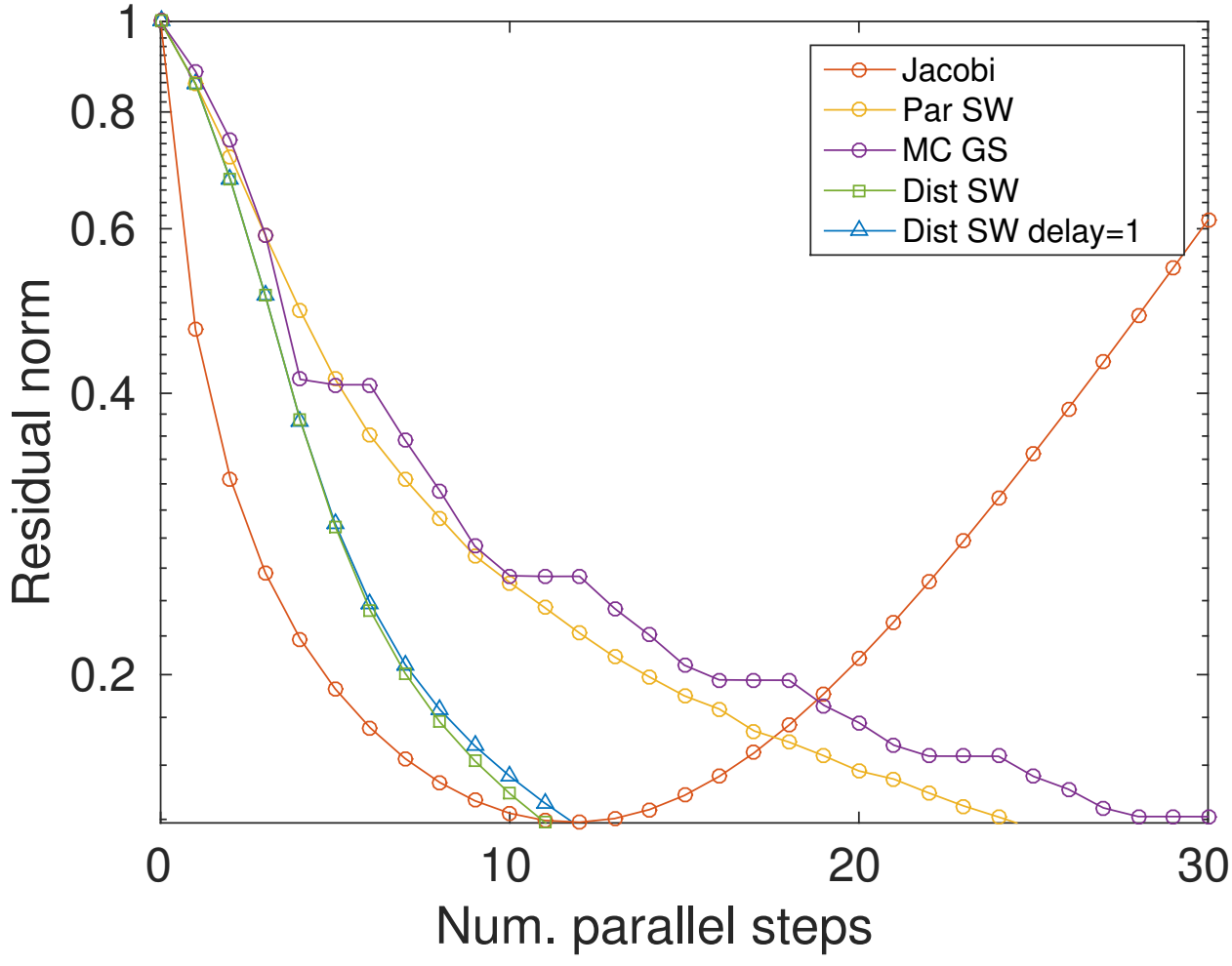
Delay improves convergence but reduces parallelism (equations that are relaxed cannot relax at the next iteration).

Comparison by parallel steps



Distributed Southwell is better than Parallel Southwell per parallel step.

Comparison by parallel steps



Cannot take too many Jacobi steps!

Challenges

- Incorporating multiple levels (coarse grid solves) asynchronously
- Distributed termination (detecting and signaling convergence)

Conclusions

Asynchronous methods

- Can be faster than synchronous methods when there is load imbalance or machine non-uniformities

Optimized Schwarz

- Converges rapidly and can be implemented asynchronously
- A challenge is to determine optimization parameters

Parallel and distributed Southwell

- Potentially useful if communication cost is a bottleneck
- Challenge is to develop these ideas into an asynchronous multilevel solver

Team

Georgia Tech
Edmond Chow
Jordi Wolfson-Pou
Paritosh Ramanan
Fan Zhou

Sandia Labs
Erik Boman
Siva Rajamanickam
Christian Glusa

University of Tennessee
Jack Dongarra
Hartwig Anzt
Ichitaro Yamazaki

Temple University
Daniel Szyld
José Garay
Mireille El-Haddad

This material is based upon work supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Number DE-SC-0016564.