Asynchronous Iterative Methods

Edmond Chow and Jordi Wolfson-Pou
School of Computational Science and Engineering
Georgia Institute of Technology

Workshop IV: New Architectures and Algorithms
Institute for Pure and Applied Mathematics
Los Angeles, CA
November 26-30, 2018

Motivation for asynchronous iterative methods

Iterative methods for solving large systems of linear and nonlinear equations

- naturally implemented using a bulk synchronous parallel (BSP) programming model

Motivation for asynchronous iterative methods

Iterative methods for solving large systems of linear and nonlinear equations

- naturally implemented using a bulk synchronous parallel (BSP) programming model

Potentially high cost of synchronization due to:

- exponentially increasing parallelism in hardware
- increasing heterogeneity in compute and memory access hardware

Motivation for asynchronous iterative methods

Iterative methods for solving large systems of linear and nonlinear equations

- naturally implemented using a bulk synchronous parallel (BSP) programming model

Potentially high cost of synchronization due to:

- exponentially increasing parallelism in hardware
- increasing heterogeneity in compute and memory access hardware

Asynchronous iterative methods first proposed in 1969

- literature predominantly on theoretical asymptotic convergence
- recent interest on practical behavior

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

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

Written explicitly:

$$\begin{aligned} x_1 &= g_1(x_1, x_2, x_3, x_4, \cdots, x_n) \\ x_2 &= g_2(x_1, x_2, x_3, x_4, \cdots, x_n) \\ &\vdots \\ x_n &= g_n(x_1, x_2, x_3, x_4, \cdots, x_n) \end{aligned}$$

Consider a distributed parallel computation: x is partitioned among the processes and the process that "owns" x_i will update x_i using the formula g_i.

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

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

Written explicitly:

$$\begin{aligned} x_1 &= g_1(x_1, x_2, x_3, x_4, \cdots, x_n) \\ x_2 &= g_2(x_1, x_2, x_3, x_4, \cdots, x_n) \\ &\vdots \\ x_n &= g_n(x_1, x_2, x_3, x_4, \cdots, x_n) \end{aligned}$$

Consider a distributed parallel computation: x is partitioned among the processes and the process that "owns" x_i will update x_i using the formula g_i.

Synchronous: processes wait for other processes to finish an iteration and make updated values available before starting a new iteration; iteration $j+1$ uses values at iteration j.

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

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

Written explicitly:

$$\begin{aligned} x_1 &= g_1(x_1, x_2, x_3, x_4, \cdots, x_n) \\ x_2 &= g_2(x_1, x_2, x_3, x_4, \cdots, x_n) \\ &\vdots \\ x_n &= g_n(x_1, x_2, x_3, x_4, \cdots, x_n) \end{aligned}$$

Consider a distributed parallel computation: x is partitioned among the processes and the process that "owns" x_i will update x_i using the formula g_i.

Synchronous: processes wait for other processes to finish an iteration and make updated values available before starting a new iteration; iteration $j+1$ uses values at iteration j.

Asynchronous: processes do not wait for other processes; the updates use whatever values of x_i that are available (which might not be the "latest" ones due to communication).

Asynchronous iterative methods

Advantages

An asynchronous version of a synchronous method may be faster when there is load imbalance or non-uniform hardware.

Asynchronous methods are resilient against communication faults, e.g., dropped messages.

Communication traffic could be more uniform.

Asynchronous iterative methods

Advantages

An asynchronous version of a synchronous method may be faster when there is load imbalance or non-uniform hardware.

Asynchronous methods are resilient against communication faults, e.g., dropped messages.

Communication traffic could be more uniform.

Disadvantages

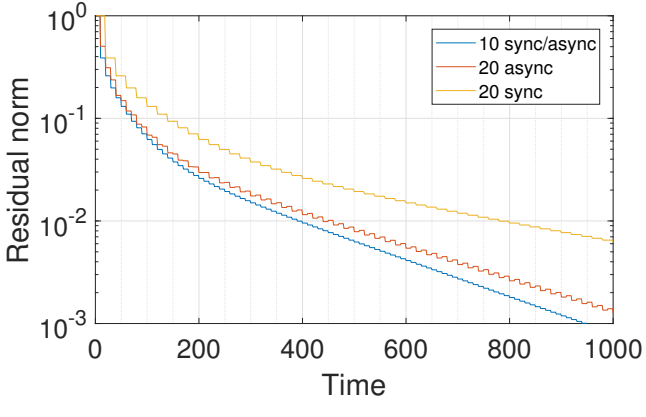
The asynchronous version of the method might not converge even if the synchronous method converges.

Results are not deterministic.

Checking whether an asynchronous iteration has satisfied a stopping criterion can be complicated (but may not be necessary).

Asynchronous vs. synchronous: 1 slow processor out of 10

In the legend, the number is the time interval between updates for the slow processor



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

Asynchronous iterations

Two factors make asynchronous iterations different from synchronous iterations:

- Not all updates are performed at the same time instant
- Updates may use stale information (due to communication delays in reading or writing)

Mathematical model of asynchronous iterations

Bertsekas and Tsitsiklis 1989. (See also Chazan-Miranker 1969, Baudet 1978.)
Let $x_i^{(j)}$ denote the ith component of x at time *instant* j.

$$x_i^{(j)} = \begin{cases} x_i^{(j-1)}, & \text{if } i \notin J_j \\ g_i(x_1^{(s_1^i(j))}, x_2^{(s_2^i(j))}, \ldots, x_n^{(s_n^i(j))}), & \text{if } i \in J_j \end{cases}$$

where J_j is the set of indices updated at instant j, and $s_k^i(j)$ is the instant that x_k is read when computing g_i at instant j (and account for read/write communication delays).

Assumptions:

1. $s_k^i(j) < j$, i.e., at instant j, a process cannot read other values computed at instant j or in the future of instant j
2. Cannot have a sequence $s_k^i(j)$ over time such that iterations only use old values, i.e., newer values must eventually be used
3. No component i is abandoned forever, i.e., no process stops updating

The model is very general and includes synchronous iterations as special cases.

Mathematical model of asynchronous iterations

Bertsekas and Tsitsiklis 1989. (See also Chazan-Miranker 1969, Baudet 1978.)
Let $x_i^{(j)}$ denote the ith component of x at time *instant* j,

$$x_i^{(j)} = \begin{cases} x_i^{(j-1)}, & \text{if } i \notin J_j \\ g_i(x_1^{(s_1^i(j))}, x_2^{(s_2^i(j))}, \ldots, x_n^{(s_n^i(j))}), & \text{if } i \in J_j \end{cases}$$

where J_j is the set of indices updated at instant j, and $s_k^i(j)$ is the instant that x_k is read when computing g_i at instant j (and account for read/write communication delays).

- It is reasonable (but not necessary) to assume $s_k^i(j_1) \leq s_k^i(j_2)$ if $j_1 < j_2$, i.e., once a value of x_k is read, an older version of x_k is not read.
- It could be reasonable to assume that, for a given k and j, all $s_k^i(j)$ are equal for different i, i.e., at instant j, all computations are performed with the same values of x.
- When x_i depends on x_i itself (not in the case of Jacobi), $x_i^{(j)}$ might not be computed using the latest value $x_i^{(j-1)}$. This handles the case that the computation of x_i is assigned to different processes at different times.

Convergence theorem for linear case

To solve the nonsingular system $Ax = b$, rewrite the equation in the form $x = G(x)$ as

$$\begin{aligned}(M - N)x &= b \\ x &= (M^{-1}N)x + M^{-1}b\end{aligned}$$

Define the iteration matrix $T = M^{-1}N$. The corresponding synchronous iterative method converges for any initial guess *if and only if* $\rho(T) < 1$.

Convergence theorem for linear case

To solve the nonsingular system $Ax = b$, rewrite the equation in the form $x = G(x)$ as

$$\begin{aligned}(M - N)x &= b \\ x &= (M^{-1}N)x + M^{-1}b\end{aligned}$$

Define the iteration matrix $T = M^{-1}N$. The corresponding synchronous iterative method converges for any initial guess *if and only if* $\rho(T) < 1$.

The corresponding asynchronous iterative method converges for any initial guess *if and only if*

$$\rho(|T|) < 1.$$

Note: $\rho(T) \leq \rho(|T|)$.

If $\rho(|T|) \geq 1$, then there exists an initial guess $x^{(0)}$ and a sequence of asynchronous iterations that does not converge to the fixed point.

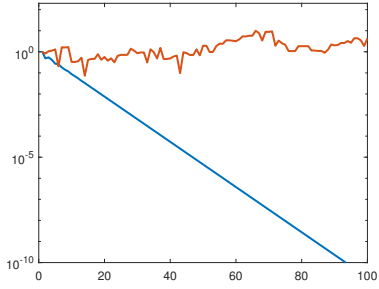
Simulating asynchronous iterations

- Update probability: at each time instant a variable is updated with a given probability
- Delay bound: when a variable is updated, the data is read from k past time instants, up to the delay bound ($k = 1$ is most recent data)

Simulation results for update probability 0.5 and delay bound 3.

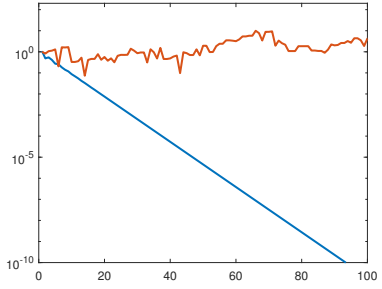
Asynchronous Jacobi convergence for 3×3 matrices with general model

$\rho(T) = 0.7812,\ \rho(|T|) = 1.6949$ $\rho(T) = 0.8677,\ \rho(|T|) = 1.2681$

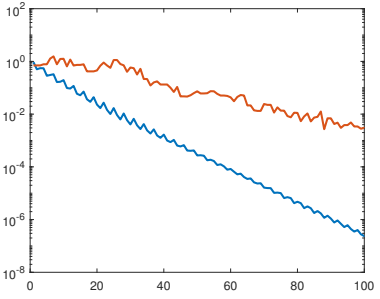


Asynchronous Jacobi convergence for 3×3 matrices with general model

$\rho(T) = 0.7812,\ \rho(|T|) = 1.6949$

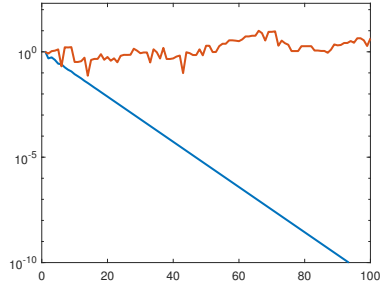


$\rho(T) = 0.8677,\ \rho(|T|) = 1.2681$

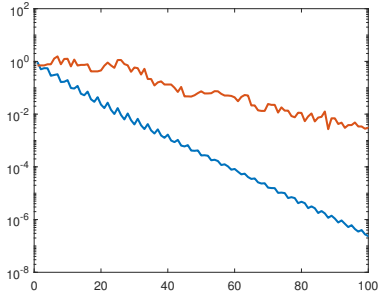


Asynchronous Jacobi convergence for 3×3 matrices with general model

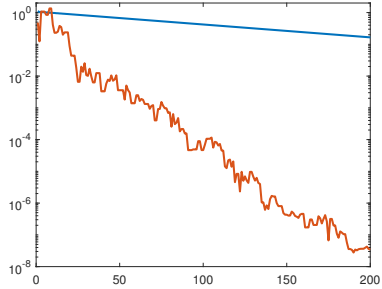
$\rho(T) = 0.7812$, $\rho(|T|) = 1.6949$



$\rho(T) = 0.8677$, $\rho(|T|) = 1.2681$



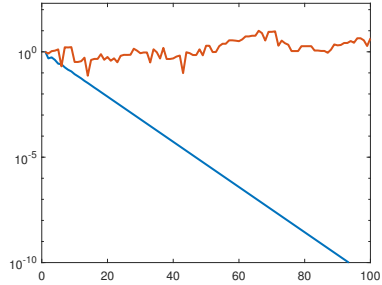
$\rho(T) = 0.9907$, $\rho(|T|) = 1.3003$



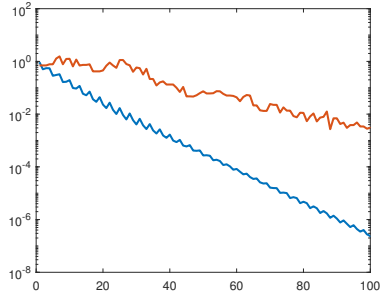
$\rho(T) = 1.0244$, $\rho(|T|) = 1.1545$

Asynchronous Jacobi convergence for 3×3 matrices with general model

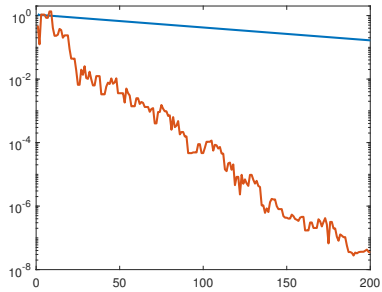
$\rho(T) = 0.7812,\ \rho(|T|) = 1.6949$



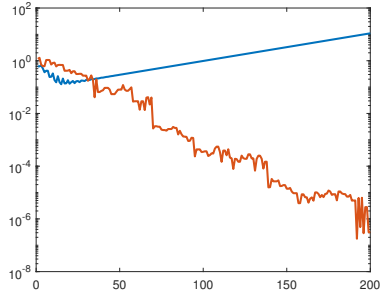
$\rho(T) = 0.8677,\ \rho(|T|) = 1.2681$



$\rho(T) = 0.9907,\ \rho(|T|) = 1.3003$



$\rho(T) = 1.0244,\ \rho(|T|) = 1.1545$



The convergence theory gives an overly negative view of asynchronous iterative methods

For synchronous methods, $\rho(T) < 1$ reliably predicts that the iterations will converge for any initial guess.

If $\rho(T) \geq 1$, the iterations "converge" if the initial error does not lie in the subspace spanned by the eigenvectors of T corresponding to eigenvalues ≥ 1.

The convergence theory gives an overly negative view of asynchronous iterative methods

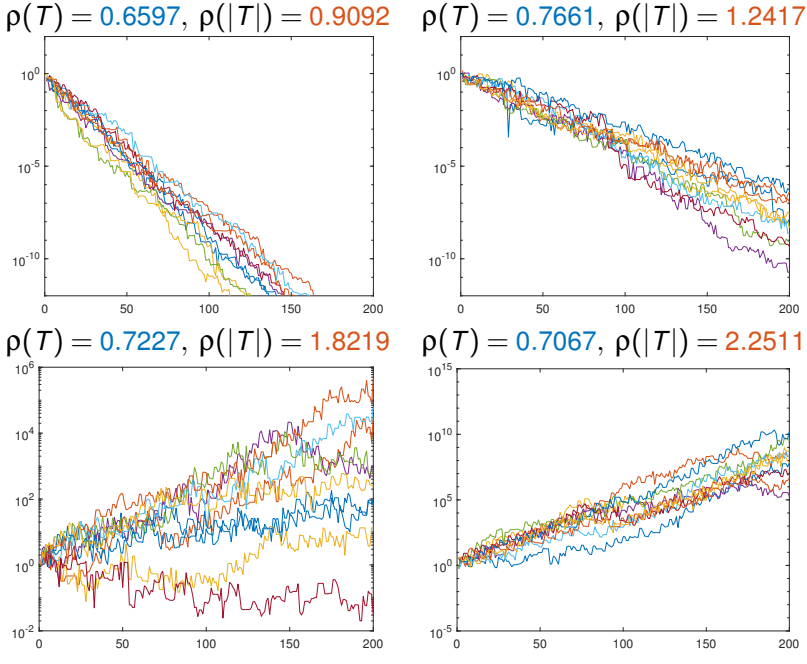
For synchronous methods, $\rho(T) < 1$ reliably predicts that the iterations will converge for any initial guess.

If $\rho(T) \geq 1$, the iterations "converge" if the initial error does not lie in the subspace spanned by the eigenvectors of T corresponding to eigenvalues ≥ 1.

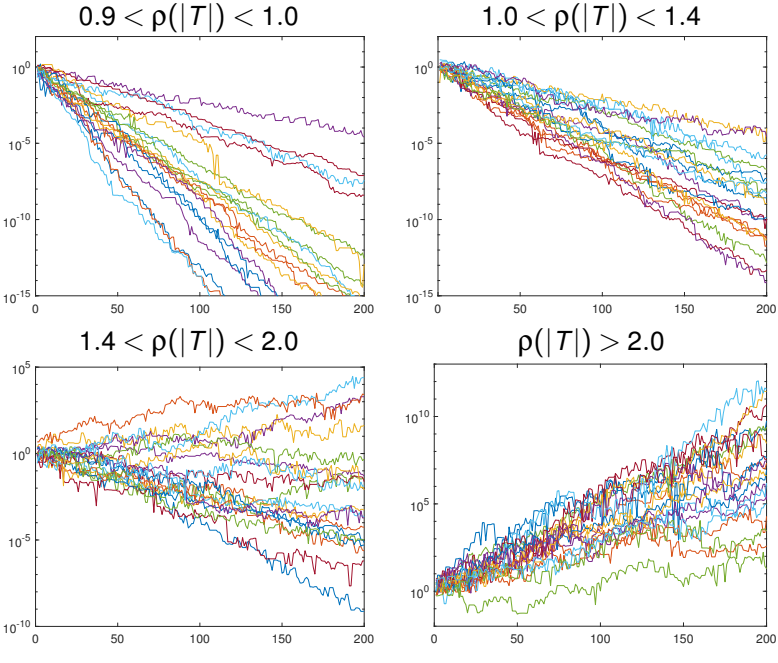
For asynchronous methods, $\rho(|T|) < 1$ guarantees the iterations will converge.

But, if $\rho(|T|) \geq 1$, the iterations may converge anyway (seems more likely if $\rho(|T|)$ is not too large).

10 asynchronous sequences per matrix; behavior related to $\rho(|T|)$

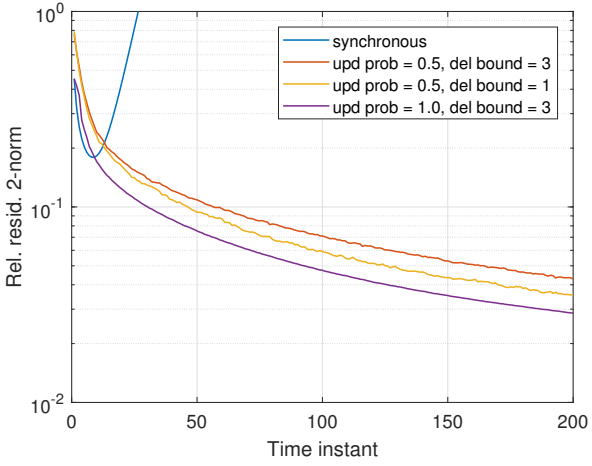


Convergence seems related to $\rho(|T|)$; 20 matrices with $\rho(T) < 0.9$



More realistic example: asynchronous Jacobi with general model

Isotropic diffusion with unstructured FEM discretization, $n = 3081$ and $\rho(T) > 1$



For symmetric matrices, $\rho(|T|) = \rho(T)$.
In this case, if $\rho(T) > 1$, the synchronous method diverges but the asynchronous method may converge.

Model of asynchronous iterations without communication delays

Wolfson-Pou and Chow 2018. (See also Robert, Charnay, and Musy 1975.)
Let $x_i^{(j)}$ denote the ith component of x at time *instant* j.

$$x_i^{(j)} = \begin{cases} x_i^{(j-1)}, & \text{if } i \notin J_j \\ g_i(x_1^{(j-1)}, x_2^{(j-1)}, \ldots, x_n^{(j-1)}), & \text{if } i \in J_j \end{cases}$$

where J_j is the set of indices updated at instant j.

Same assumptions as before.

Asynchronous Jacobi without communication delays can be written as

$$x^{(j)} = (I - \hat{D}^{(j-1)}A)x^{(j-1)} + \hat{D}^{(j-1)}b$$

where

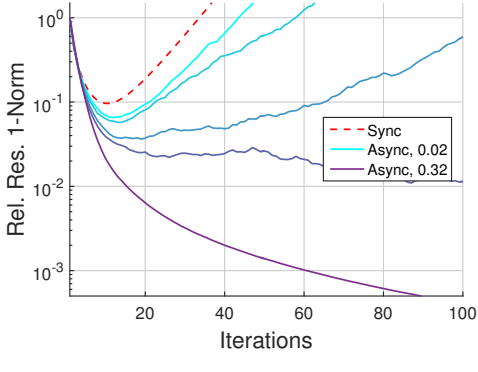
$$\hat{D}_{ii}^{(j-1)} = \begin{cases} 1/A_{ii}, & \text{if } i \in J_j, \\ 0, & \text{otherwise.} \end{cases}$$

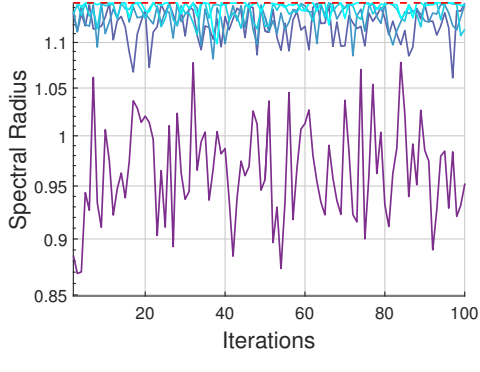
The matrix in brackets could be called a *propagation matrix*.

Model of convergence without communication delays

Isotopic diffusion FE matrix

Fraction of non-updated equations per time instant (iteration) is varied from 0.02 to 0.32.





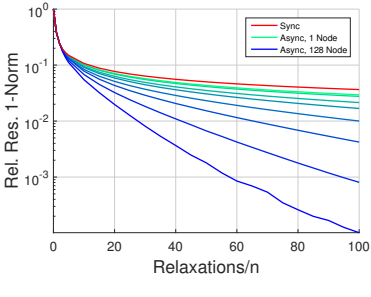
Right: Spectral radius of each propagation matrix is plotted.

Experiments on distributed memory computers

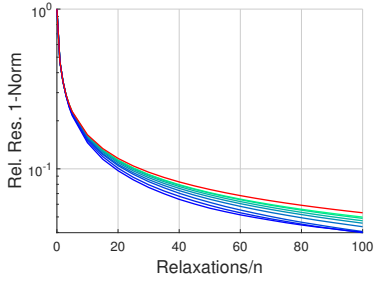
Residual norm history

32 to 4096 processes (1 to 128 nodes).
Problem size increases from left to right.

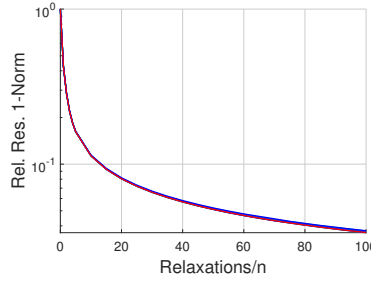
thermomech_dM
$n = 204,316$



parabolic_fem
$n = 525,825$



thermal2
$1,227,087$



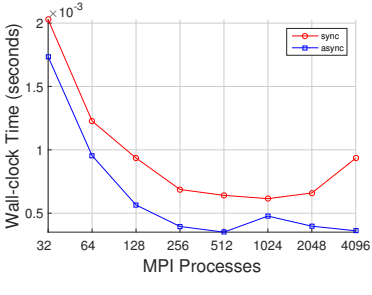
Asynchronous convergence is faster than synchronous convergence if more processes are used, especially for smaller problems (small number of equations per process).

Experiments on distributed memory computers

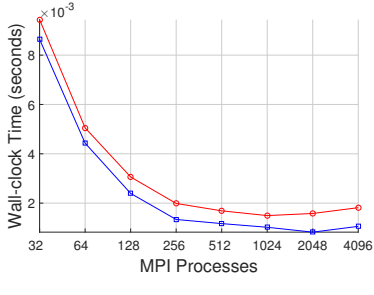
Timings

32 to 4096 processes (1 to 128 nodes).
Problem size increases from left to right.

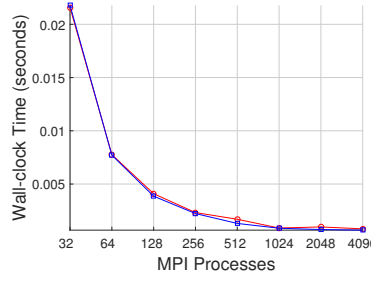
thermomech_dM
$n = 204,316$



parabolic_fem
$n = 525,825$



thermal2
$1,227,087$



Asynchronous convergence is faster than synchronous convergence if more processes are used, especially for smaller problems (small number of equations per process).

Implementation on distributed memory computers

On distributed memory systems, asynchronous iterative methods are naturally implemented using *one-sided* communication.

- Processes “put” data onto other processes without caring if/when it is written or read.

Implementation with one-sided communication

Origin of Put

```
MPI_Win_allocate(size, &data, &win);

MPI_Win_lock(MPI_LOCK_SHARED, other, win);

while (!converged) {
  MPI_Put(x, len, other, win);
  MPI_Win_flush_local(other, win);
}

MPI_Win_unlock(other, win);

MPI_Win_free(&win);
```

Target of Put

```
MPI_Win_allocate(size, &data, &win);

while (!converged) {
  MPI_Win_sync(win);
  x = data[i];
}

MPI_Win_free(&win);
```

Do we have any performance guarantees?

- Timely asynchronous progress?
- On different networks and different MPI implementations?
- Implementations or hardware may buffer one-sided communications if no synchronizing flush is used

Simulating remote memory access with two-sided MPI

Asynchronous iterative method communication can be simulated with two-sided MPI by allowing senders to have multiple messages in flight and never blocking to complete these sends (i.e., use a nonblocking Test for completion, which also completes a send if possible).

Number of messages in flight can be very large. This can happen if the receiver is delayed and does not complete receives during this delay. Other researchers just ignore messages if too many sends are in flight.

Use of two-sided MPI is unnatural for asynchronous iterative methods, but an interface layer could be created.

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

Bethune, Bull, Dingle, and Higham, “Performance analysis of asynchronous Jacobi’s method implemented in MPI, SHMEM and OpenMP,” 2014

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

Magoulès and Gbikpi-Benissan, “JACK2: An MPI-based communication library with non-blocking synchronization for asynchronous iterations,” 2018

Motivation for asynchronous optimized Schwarz

Optimized Schwarz methods (e.g., Gander, Nataf) are fixed-point iterations that converge rapidly.

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

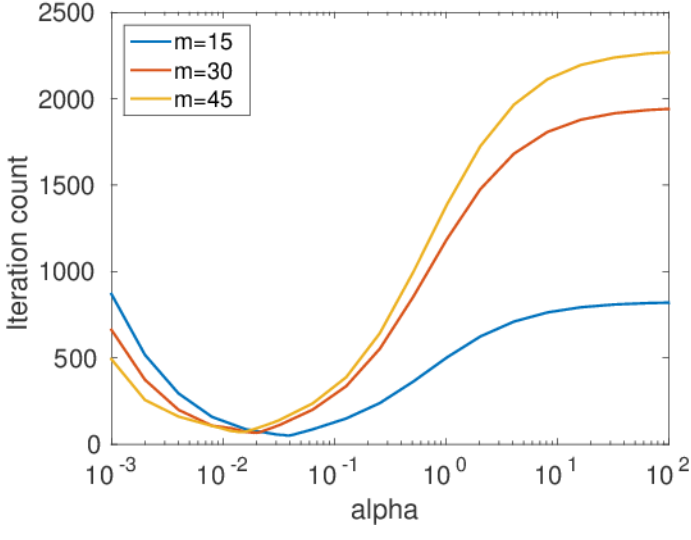
$$\frac{\partial u^{(1)}}{\partial n_1} + \alpha u^{(1)} = \frac{\partial u^{(2)}}{\partial n_1} + \alpha u^{(2)}$$

for subdomains (1) and (2).

Parameter α is chosen to optimize convergence.
The case $\alpha \to \infty$ corresponds to classical Schwarz.

Some theory provides for convergence of asynchronous versions in certain cases (Magoulès, Szyld, Venet, 2017).

Optimized Jacobi-Schwarz on 2D FD Laplacian



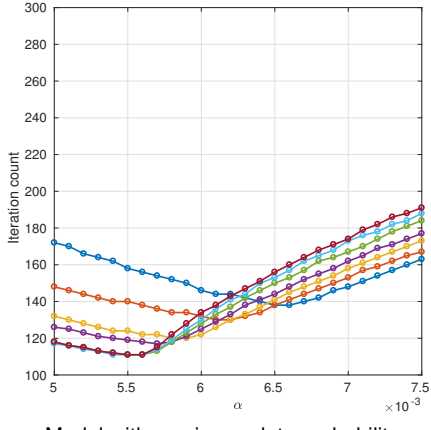
Processor grid: 10 by 10

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

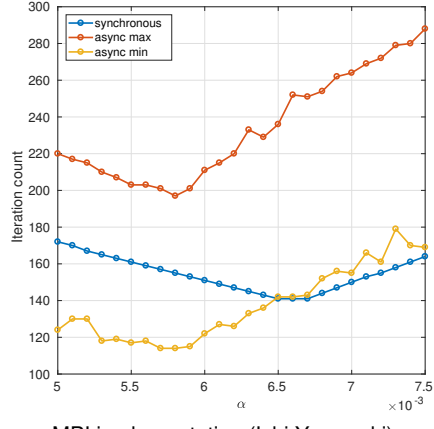
"Minimum overlap" is used.

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

Asynchronous optimized Jacobi-Schwarz; model and implementation



Model with varying update probability
1.00 to 0.88 (blue to maroon)



MPI implementation (Ichi Yamazaki)

Processor grid: 16 by 16; Local problem size: 64 by 64 grid.

Asynchronous multigrid

Multigrid is a scalable iterative method for solving discretized PDEs.
Iteration count (for given accuracy) is constant for all problem sizes.

Convergence of the basic iterative method slows down when the error becomes (algebraically) smooth. When this happens, project the problem to a coarser grid where the error looks oscillatory and convergence of the basic iterative method is rapid.

Try to remove synchronization points in multigrid $\rightarrow$ asynchronous multigrid.

Two-level method

One iteration of the standard two-level method for solving $Ax = b$ is

$$\begin{aligned} x^{(j-1/2)} &= x^{(j-1)} + M^{-1}(b - Ax^{(j-1)}) \\ x^{(j)} &= x^{(j-1/2)} + P_1^0 A_1^{-1} R_0^1 (b - Ax^{(j-1/2)}) \end{aligned}$$

where M^{-1} corresponds to one smoothing step.
The smoothing and coarse grid correction are multiplicative.

Standard (multiplicative) method: $x = K(S(x))$

Two-level method

One iteration of the standard two-level method for solving $Ax = b$ is

$$\begin{aligned} x^{(j-1/2)} &= x^{(j-1)} + M^{-1}(b - Ax^{(j-1)}) \\ x^{(j)} &= x^{(j-1/2)} + P_1^0 A_1^{-1} R_0^1 (b - Ax^{(j-1/2)}) \end{aligned}$$

where M^{-1} corresponds to one smoothing step.
The smoothing and coarse grid correction are multiplicative.

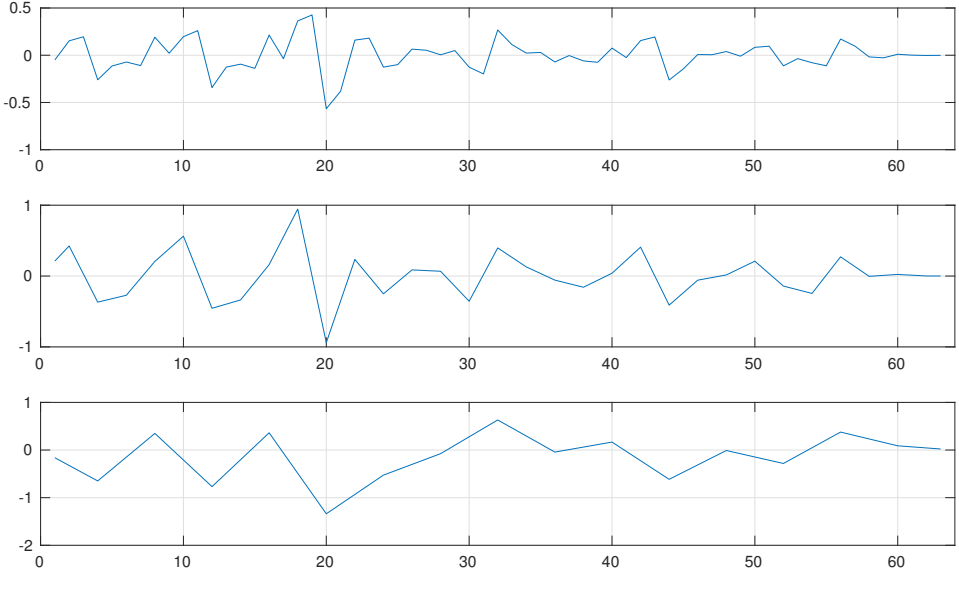
Standard (multiplicative) method: $x = K(S(x))$

In an asynchronous iterative method, the smoothing and coarse grid corrections should be independent.

Additive method: $x = x + S_e(x) + K_e(x)$

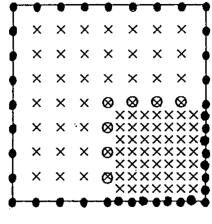
Additive method “overcorrects” x

Additive multigrid corrections from each grid, 3 levels, fine to coarse



Additive multigrid methods

- Bramble, Pasciak, and Xu (BPX) 1990
- Greenbaum 1986
- Vassilevski and Yang 2014
- McCormick, Quinlan, Lee, Philip (AFACx) 1989–



Writing the multiplicative method as an additive method (multadd)

Recall one iteration of the standard two-level method for solving $Ax = b$:

$$\begin{aligned} x^{(j-1/2)} &= x^{(j-1)} + M^{-1}(b - Ax^{(j-1)}) \\ x^{(j)} &= x^{(j-1/2)} + P_1^0 A_1^{-1} R_0^1 (b - Ax^{(j-1/2)}) \end{aligned}$$

Substitute the rhs of the smoothing step into the second term of the rhs of the coarse grid correction step:

$$x^{(j)} = x^{(j-1/2)} + P_1^0 A_1^{-1} R_0^1 (I - AM^{-1})(b - Ax^{(j-1)})$$

which is the sum of the corrections from the fine grid and from the coarse grid.

The factor $\bar{R} \equiv R_0^1 (I - AM^{-1})$ is the transpose of a *smoothed prolongation* operator, and can be approximated for computational efficiency.

The method could be extended to multiple levels and run asynchronously, i.e., run each level independently.

What could a convergence theory look like?

In the two-level, synchronous case, the error propagation is

$$e^{(j)} = K\, S\, e^{(j-1)}$$

where we want $\|KS\|_A$ to be bounded above by $\rho < 1$ independently of the grid parameters.

- If the smoother is not run in a given step, then

  $$e^{(j)} = K\, e^{(j-1)}$$

  which does not increase the error if K is an A-orthogonal projector.
- If the coarse grid correction is not run in a given step, then

  $$e^{(j)} = S\, e^{(j-1)}$$

  which does not increase the error if $\|S\|_A < 1$. However, $\|S\|_A$ is not independent of the coarse grid parameters.
- Conceptually, to maintain grid-independent convergence, we need to bound

  $$\|K^{\mu} S^{\nu}\|_A$$

  for *likely* combinations of μ, ν.

AFACx with full refinement

The additive method for solving $Ax = b$ is

$$x^{(j)} = x^{(j-1)} + \sum_{k=0}^{L} B_k(b - Ax^{(j-1)})$$

where

$$B_k = P_k^0 M_k^{-1} R_0^k - P_k^0 M_k^{-1} A_k P_{k+1}^k M_{k+1}^{-1} R_0^{k+1}, \quad k < L.$$

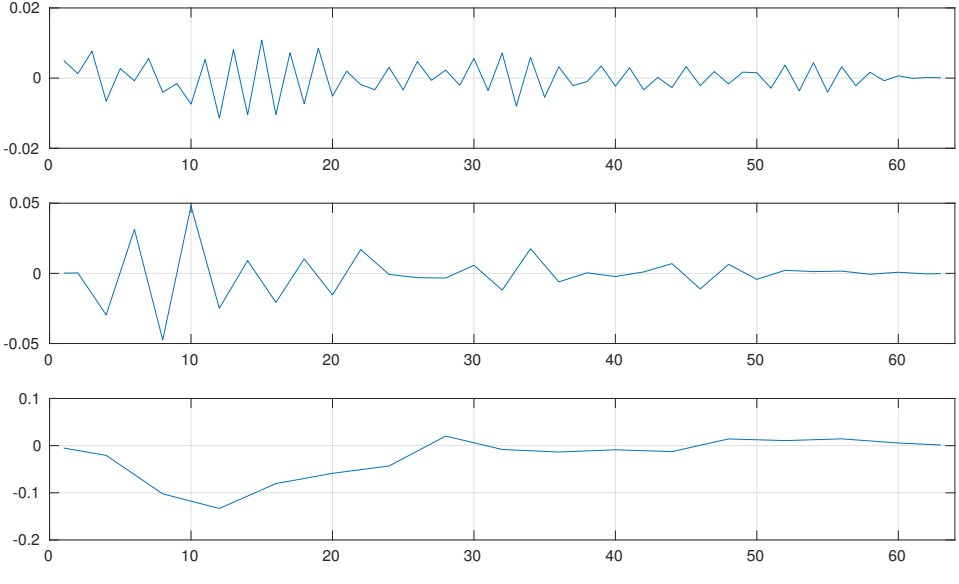
and where M_k^{-1} corresponds to smoothing on level k with a zero guess.

Interpretation: at each level k, except the coarsest,

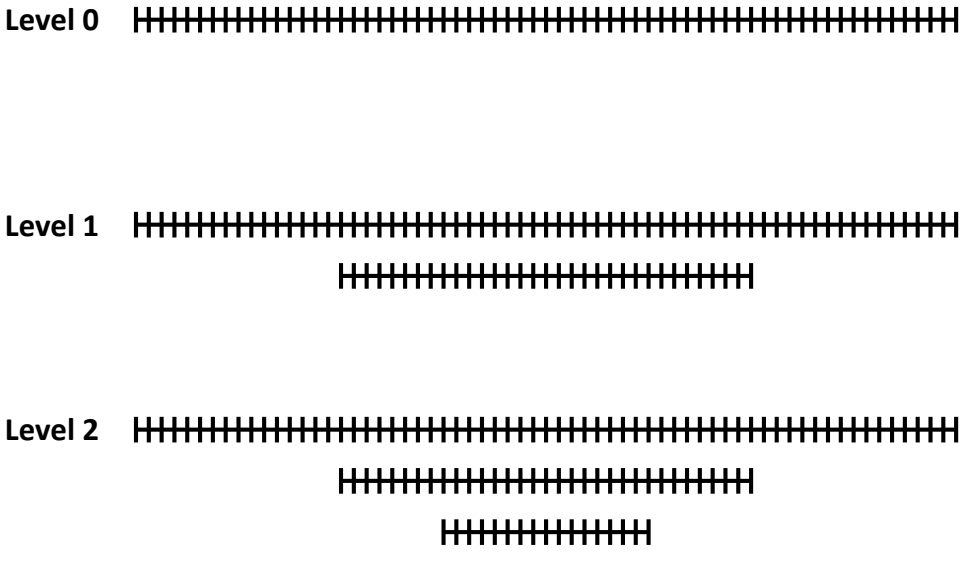
$$g_k = P_{k+1}^k M_{k+1}^{-1} R_0^{k+1} r$$
$$e_k = P_k^0 M_k^{-1} (R_0^k r - A_k g_k).$$

On the coarsest level, solve $A_k e_k = R_0^k r$ exactly.

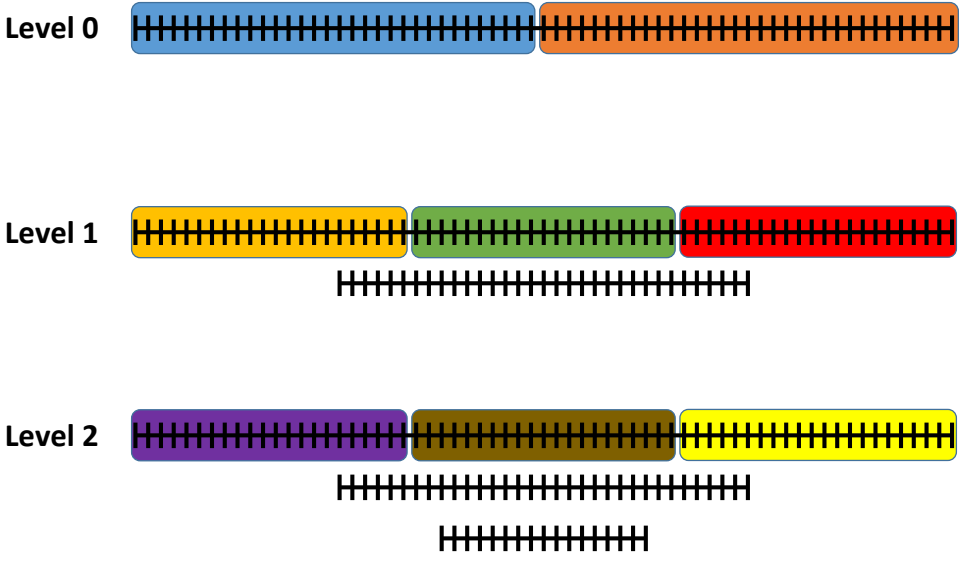
AFACx corrections from each grid, 3 levels, fine to coarse



Partitioning of work – example for 3 levels



Partitioning of work – 8 processes



Asynchronous multigrid model

Instead of partitioning the unknowns x among the processes, we partition the levels among the processes, e.g., one (or more) processes per level.

Instead of reading components of x from other processes,

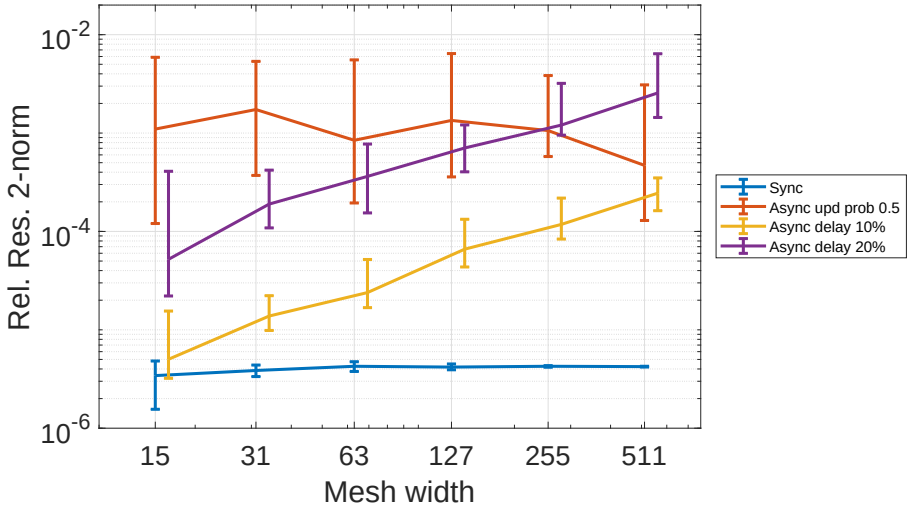
$$x^{(j)} = \sum_{i \in J_j} B_i(x_1^{(s_1^i(j))}, x_2^{(s_2^i(j))}, \ldots, x_n^{(s_n^i(j))})$$

we read components of the residual r. The asynchronous model for multigrid generates a sequence of residuals $r^{(j)}$ at time instants j,

$$r^{(j)} = \sum_{i \in J_j} G_i(r_1^{(s_1^i(j))}, r_2^{(s_2^i(j))}, \ldots, r_n^{(s_n^i(j))})$$

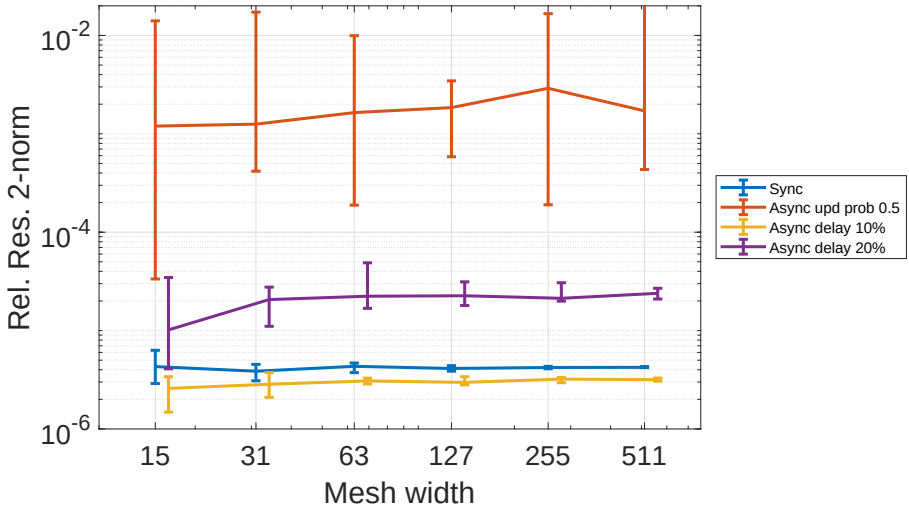
where J_j is the set of levels whose coarse grid corrections are applied at instant j, and $s_m^i(j)$ is the instant that r_m is read when computing G_i at instant j.

Model of asynchronous simplified multadd: 20 V(1,1) cycles for 2D FD Laplacian (async solution)



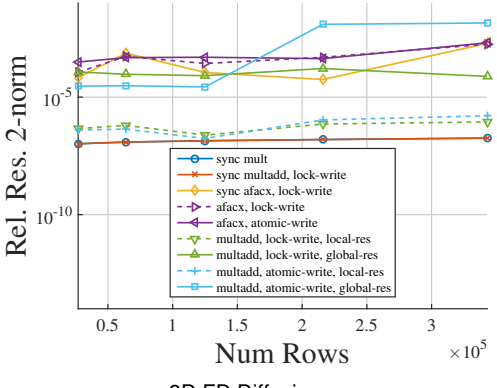
Error bars show max and min of 10 runs. Weighted Jacobi smoothing with $\omega = 4/5$ (synchronous). Exact solve on 3×3 coarsest grid.

Model of asynchronous simplified multadd: 20 V(1,1) cycles for 2D FD Laplacian (async residual)

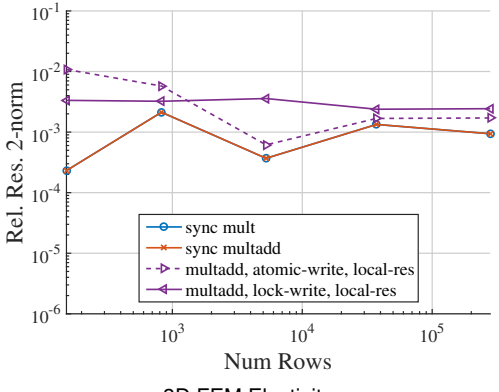


Error bars show max and min of 10 runs. Weighted Jacobi smoothing with $\omega = 4/5$ (synchronous). Exact solve on 3×3 coarsest grid.

OpenMP implementation with 32 threads, 30 V-cycles



3D FD Diffusion



3D FEM Elasticity

OpenMP timings: 32 threads

3D FD Diffusion, $30 \times 30 \times 30$ grid

Stopping criterion: rel. residual norm reduction by 10^{-6}

	time	avg. corrections	V-cycles
sync mult	0.0090	45	45
sync AFACx	0.0145	45	45
async AFACx	0.0196	102	75
sync multadd	0.0141	45	45
async multadd	0.0067	44	40

Conclusions

- Asynchronous iterative methods can converge even when the corresponding synchronous method does not converge, i.e., asynchronicity can improve convergence.
- Numerical tests show that asynchronous versions of the optimized Schwarz method do converge, complementing the limited existing theory.
- Asynchronous multigrid does not have a convergence theory but practical scalable performance of asynchronous multigrid appears possible.

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.