CS 598: Communication Cost Analysis of Algorithms

Lecture 18: Avoiding communication in iterative solvers

Edgar Solomonik

University of Illinois at Urbana-Champaign

October 24, 2016

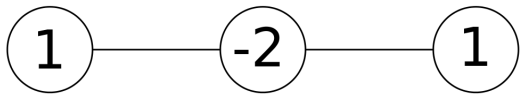
Stencils

Lets consider approximation of the second derivative of a function $u(x)$

- we can derive an approximation from a truncated Taylor expansion with step size h

$$\frac{d^2u}{dx^2}(x) \approx \frac{u(x-h) - 2u(x) + u(x+h)}{h^2}$$

- such approximations of derivatives can be represented by a **stencil**



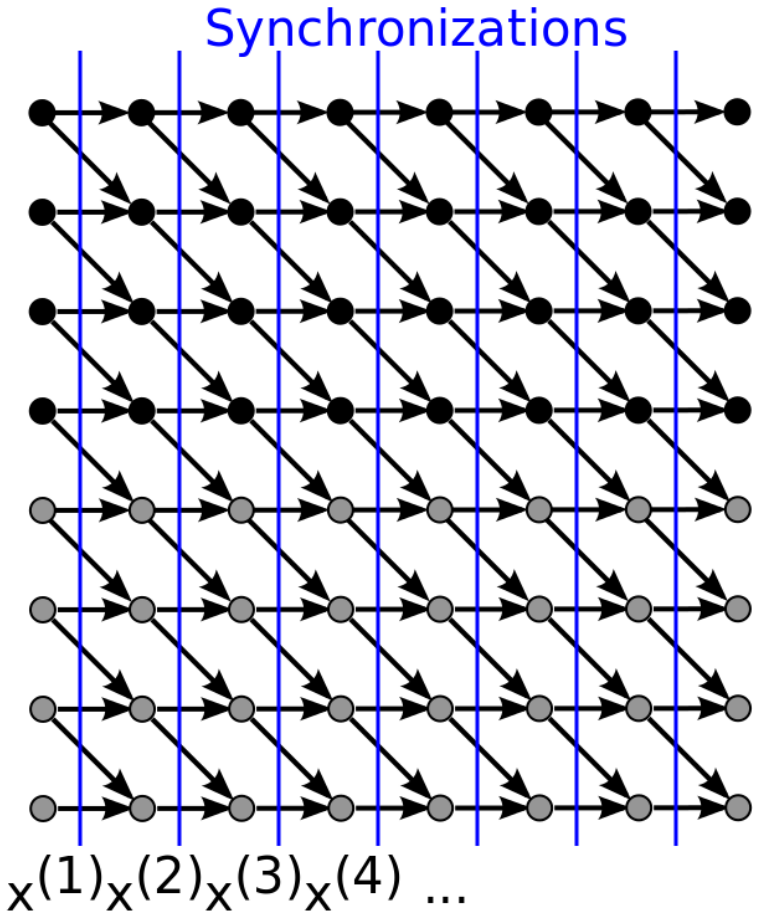
which is applied for every node in the mesh

- the application of this 1D 3-point stencil to n grid-nodes, can be done via SpMV with a tridiagonal matrix, like

$$\begin{pmatrix} \frac{d^2u}{dx^2}(h) \\ \vdots \\ \frac{d^2u}{dx^2}(nh) \end{pmatrix} = \frac{1}{h^2} \begin{pmatrix} -2 & 1 & \\ 1 & \ddots & \ddots \\ & \ddots & \end{pmatrix} \begin{pmatrix} u(h) \\ \vdots \\ u(nh) \end{pmatrix}$$

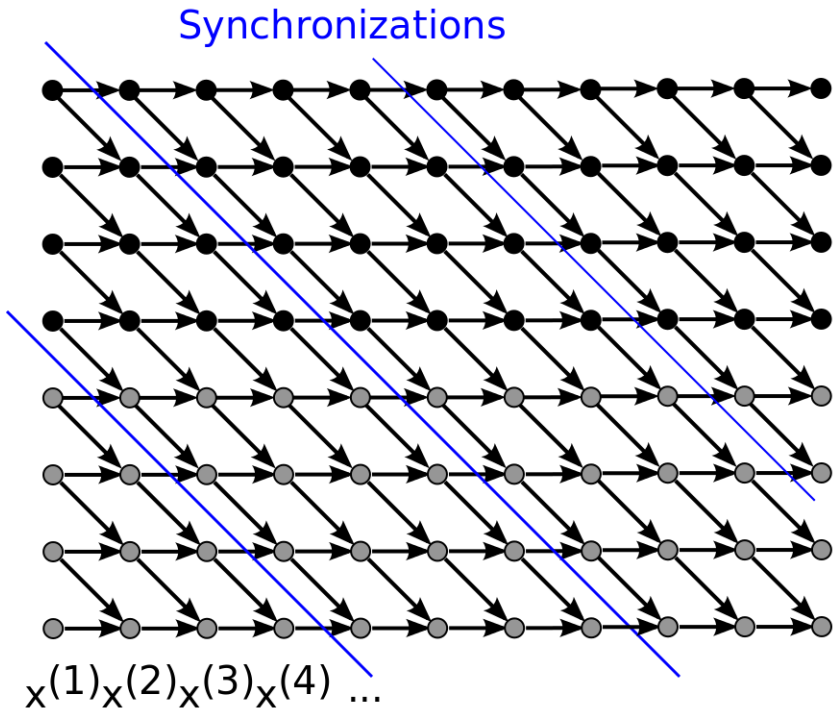
Lets start with a 1D 2-point stencil

Normally, synchronize between every stencil application



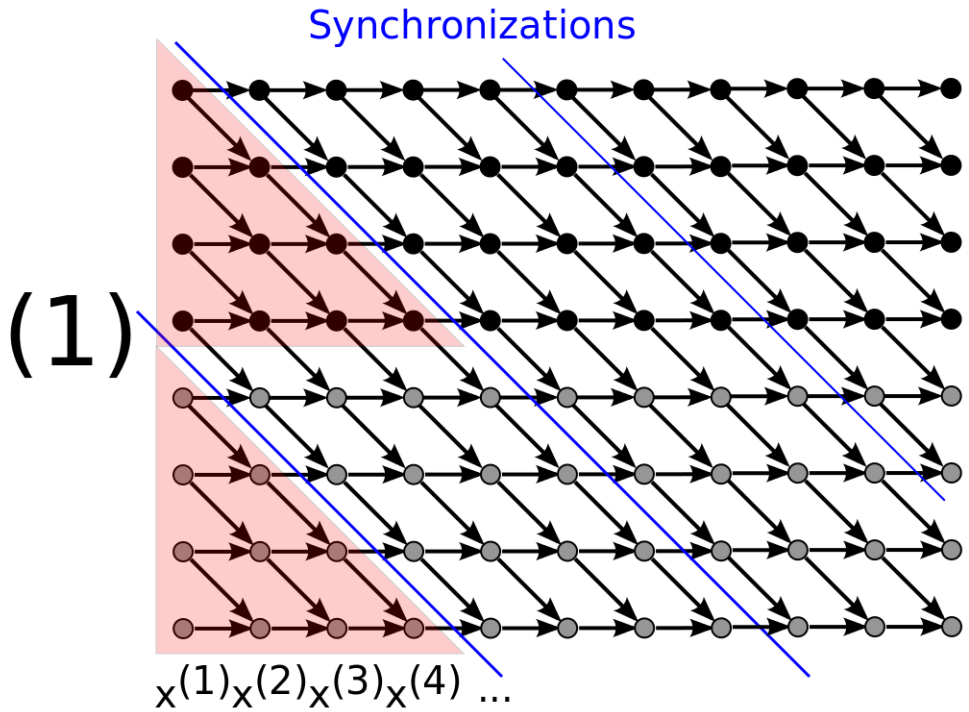
In-time blocking (matrix-powers kernel)

Avoid synchronization by applying stencil repeatedly before synchronizing



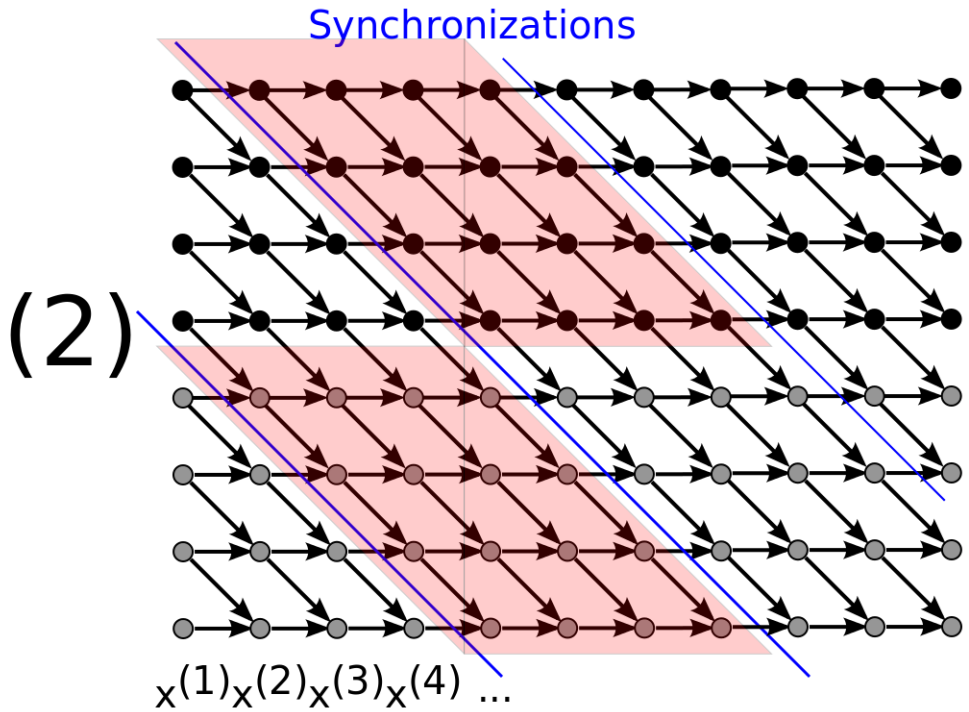
In-time blocking (matrix-powers kernel)

Avoid synchronization by applying stencil repeatedly before synchronizing



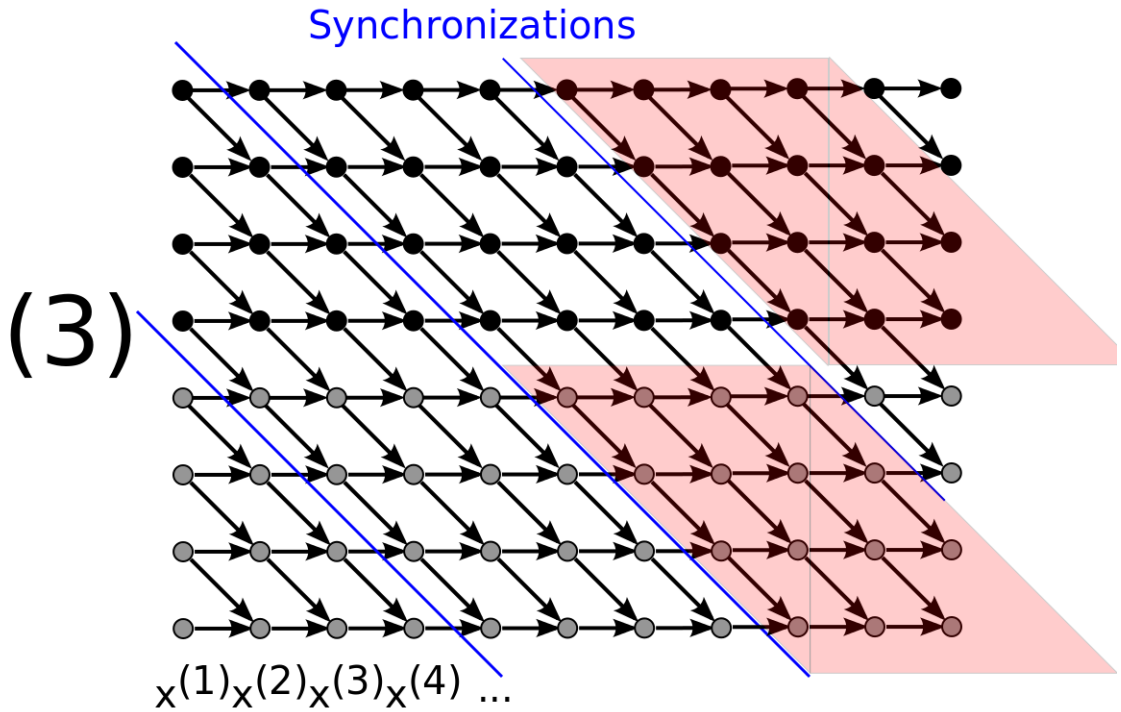
In-time blocking (matrix-powers kernel)

Avoid synchronization by applying stencil repeatedly before synchronizing



In-time blocking (matrix-powers kernel)

Avoid synchronization by applying stencil repeatedly before synchronizing



Analysis of in-time blocking for 1D mesh

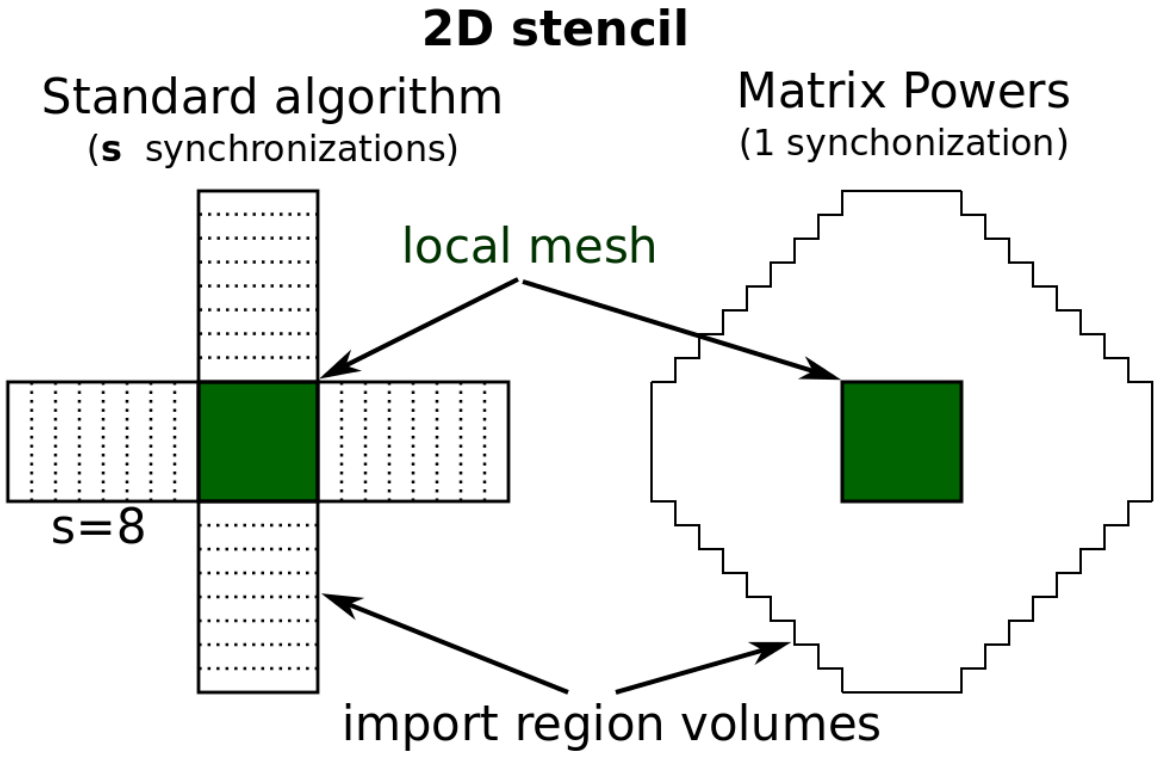
For 1D mesh its a tasty free lunch

- lets consider t steps, and execute s without synchronization
- we are constrained by $s \leq n/P$
- bring down latency cost by a factor of s, for $t = \Theta(n)$, we improve latency cost from $O(n \cdot \alpha)$ to $O(P \cdot \alpha)$
- Q: can we improve memory-bandwidth cost via in-time blocking?
- A: yes, but it depends on whether we need to output all vectors or just the last one we compute
- in the latter case, we can improve reuse by factor of $\Theta(\max(s, H))$

Analysis of in-time blocking for dD mesh

For dD mesh, there is more complexity

- again consider t steps, and execute s without synchronization
- we are constrained by $s \leq (n/P)^{1/d}$
  - otherwise we need to do asymptotically more computation and interprocessor communication



Analysis of in-time blocking for dD mesh

For dD mesh with t total steps, $s = \Theta((n/P)^{1/d})$ without syncs.

- for $t = \Theta(n^{1/d})$ we lower latency cost to $\Theta(t/s) = \Theta(P^{1/d})$
- Q: if we only need to output the final vector, by how much can we improve memory bandwidth cost with $s = \Theta((n/P)^{1/d})$?
- A: $\Theta(H^{1/d})$, by doing $O(H)$ loads and stores for computation volumes of size $H^{1/d} \times \cdots \times H^{1/d}$

Analysis of in-time blocking for dD mesh contd.

Lets now consider the full cost of t applications with s steps at a time

- recall from last lecture the cost of the one-at-a-time SpMV approach

$$t \cdot T_{\text{SpMV-d}}(n, d, P) = O\left(\frac{tn}{P} \cdot \nu + t\left(\frac{n}{P}\right)^{(d-1)/d} \cdot \beta + t \cdot \alpha\right)$$

- we now obtain a cost

$$T_{\text{CA-St}}(n, d, P, t, s) = O\left(\frac{tn}{\min(s, H^{1/d})P} \cdot \nu + \left[t\left(\frac{n}{P}\right)^{(d-1)/d} + ts^{d-1}\right] \cdot \beta + \frac{t}{s} \cdot \alpha\right)$$

- when we pick $s = (n/P)^{1/d}$, we obtain

$$T_{\text{CA-St}}(n, d, P, t, (n/P)^{1/d}) = O\left(\frac{tn}{H^{1/d}P} \cdot \nu + t\left(\frac{n}{P}\right)^{(d-1)/d} \cdot \beta + \frac{t}{(n/P)^{1/d}} \cdot \alpha\right)$$

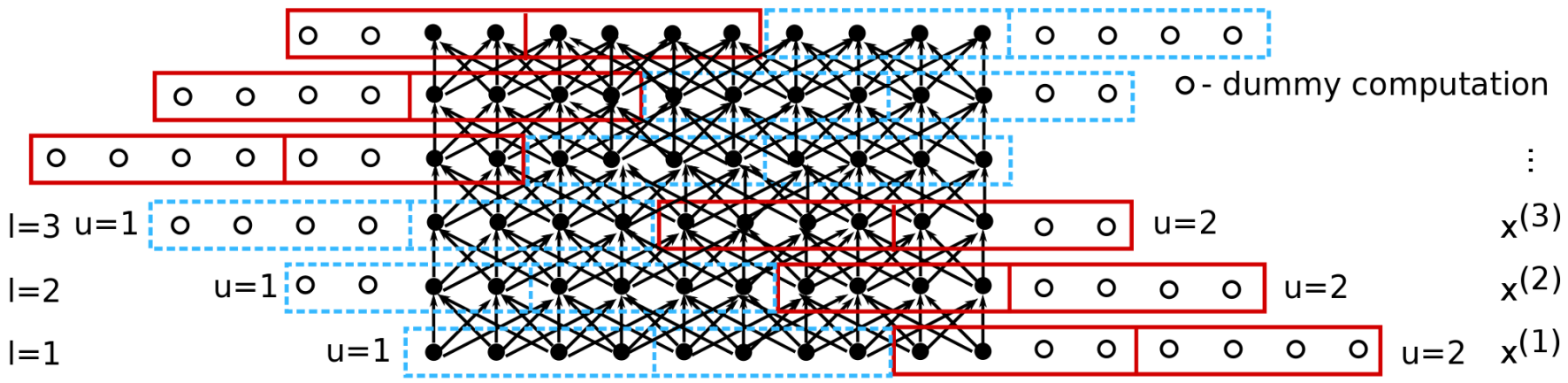
Short pause

In-cache computing

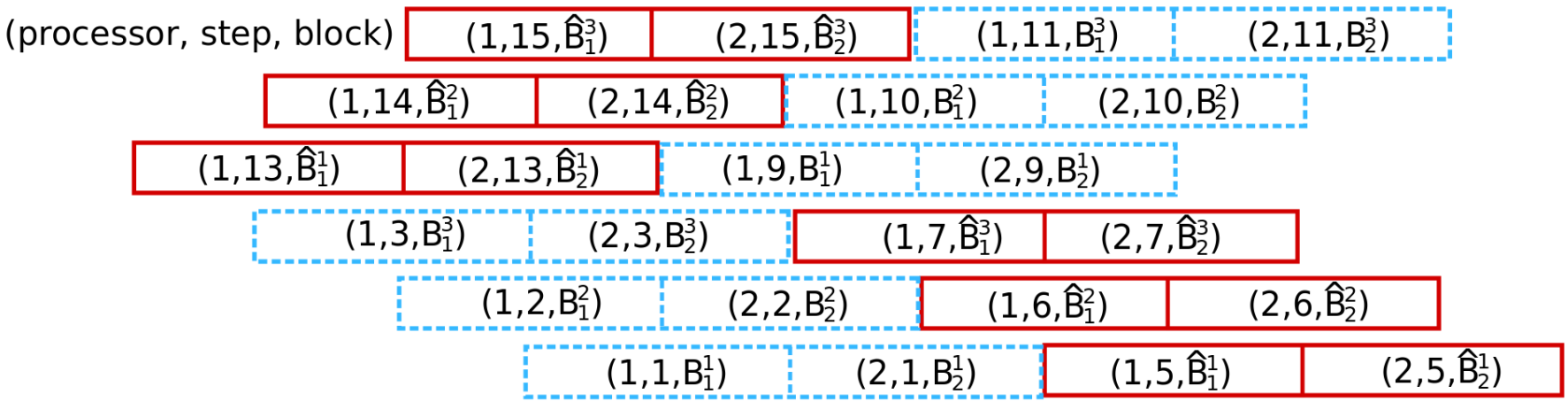
So far we have assumed that the problem does not fit in cache

- so we had $H < n/P$ or $H < m/P$ if the m nonzeros in the matrix are not implicit
- if $H > n/P$, we can keep data cache-resident between SpMVs
- we get better cache complexity without reducing synchronizations
- the memory-bandwidth cost would be proportional to interprocessor communication and insignificant if $\nu < \beta$
- we can also leverage cache-residency also when $H > m/P$
- idea: block-cyclic layout - subdivide mesh in n/b chunks so $H > b/P$

Global block-cyclic partitioning



Global block-cyclic partitioning schedule



Global block-cyclic partitioning analysis

Maintain cache residency by selecting chunks of size b with $H > b/P$

- can reduce memory bandwidth cost by up to $O(H^{1/d})$ or down to interprocessor communication cost
- however, we incur n/b more synchronizations
- incur more interprocessor communication depending on H, finer partitioning $\Rightarrow$ larger total boundary
- algorithm is a good idea when $kH = n/P$ for small k

Problems with partitioning real iterative methods

In-time blocking is hard for real applications

- we already noted the interprocessor communication overhead
- determining import region is expensive for irregular mesh/graph
- some iterative methods require computing a norm after every SpMV
- other iterative methods require orthogonalization after every SpMV
- there is ongoing research on modifying iterative methods such as biconjugate gradient (which is not orthogonalized) to allow in-time blocking while preserving convergence guarantees
- preconditioning and higher order methods, which make information propagate through the mesh *faster* make in-time blocking more expensive, as it works better for *slower* movement of information

Asynchronous iterative methods

Idea: avoid explicit synchronization by using latest-available values to apply stencil

- i.e. mix Gauss-Seidel / Bellman-Ford iterations arbitrarily
- avoid 'explicit' synchronization, but still send the same number of messages on a parallel system
- sensible in the presence of noise, soft-failures, or load-imbalance
- possible when on a shared memory-system or with one-sided communication on a distributed system
- benefit not reflected on most parallelism/communication models
- affects stability, past and ongoing research on designing asynchronous methods and bounding error

Multigrid

We can reduce communication in iterative methods the same way as computation

- improve convergence rate - lower number of SpMVs
- multigrid methods leverage a hierarchy of grids to accelerate convergence
  - perform smoothing on fine grid (e.g. stencil applications)
  - restrict residual to coarser grid
  - interpolate correction back to fine grid
  - perform smoothing again
- asymptotic view: restriction/interpolation as hard as smoothing
- grid hierarchy construction can be done 'offline'
- for detailed analysis, see recent paper: "Ballard, Siefert, and Hu, 2016. Reducing communication costs for sparse matrix multiplication within algebraic multigrid"