Parallel Numerical Algorithms

Chapter 15 – Partial Differential Equations

Prof. Michael T. Heath

Department of Computer Science
University of Illinois at Urbana-Champaign

CS 554 / CSE 512

Outline

Numerical Methods for Partial Differential Equations

- Partial differential equations are typically solved numerically by finite difference, finite element, finite volume, or spectral discretization
- Such discretization yields system of linear or nonlinear algebraic equations whose solution gives approximate solution to PDE
- Solving linear or nonlinear algebraic system is one major source of parallelism in solving PDEs numerically
- We will consider *domain decomposition* methods that exploit natural parallelism in PDE and its discretization

Alternating Schwarz Method

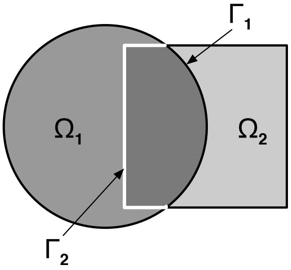
- Consider elliptic PDE

$$L\,u = f$$

on domain $\Omega = \Omega_1 \cup \Omega_2$, with boundary condition

$$u = g$$

on $\partial\,\Omega$



Alternating Schwarz Method

Given initial guess $u_2^{(0)}$ on Ω_2, for $k = 0, 1, \ldots$

- On Ω_1, solve
$$L\, u_1^{(k+1)} = f$$
with boundary conditions
$$\begin{aligned} u_1^{(k+1)} &= g \text{ on } \partial\Omega_1 \setminus \Gamma_1 \\ u_1^{(k+1)} &= u_2^{(k)} \text{ on } \Gamma_1 \end{aligned}$$
- On Ω_2, solve
$$L\, u_2^{(k+1)} = f$$
with boundary conditions
$$\begin{aligned} u_2^{(k+1)} &= g \text{ on } \partial\Omega_2 \setminus \Gamma_2 \\ u_2^{(k+1)} &= u_1^{(k+1)} \text{ on } \Gamma_2 \end{aligned}$$

Alternating Schwarz Method

- Alternating iterations continue until convergence to solution u on entire domain Ω
- Schwarz proposed this method in 1870 to deal with regions for which analytical solutions are not known
- Today it is of interest, in discretized form, for suggesting one of two major paradigms for solving PDEs numerically by *domain decomposition*
    - Overlapping subdomains (Schwarz)
    - Non-overlapping subdomains (Schur)

Discretized Schwarz Method

- Discretization yields $n \times n$ symmetric positive definite linear algebraic system

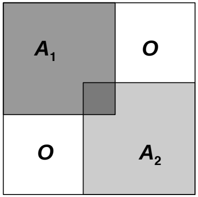
$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

- For $i = 1, 2$, let S_i be set of n_i indices of grid points in interior of Ω_i, where $n_i = |S_i|$
- Because subdomains overlap, $S_1 \cap S_2 \neq \varnothing$ and $n_1 + n_2 > n$
- For $i = 1, 2$, let $\boldsymbol{R}_i$ be $n_i \times n$ Boolean *restriction* matrix such that for any n-vector $\boldsymbol{v}$, $\boldsymbol{v}_i = \boldsymbol{R}_i\boldsymbol{v}$ contains precisely those components of $\boldsymbol{v}$ corresponding to indices in S_i (i.e., nodes in Ω_i)

Discretized Schwarz Method

- Conversely, $n \times n_i$ *extension* matrix $\boldsymbol{R}_i^T$ expands n_i-vector $\boldsymbol{v}_i$ to n-vector $\boldsymbol{v}$ whose components corresponding to indices in S_i are same as those of $\boldsymbol{v}_i$, and whose remaining components are all zero
- Principal submatrices of $\boldsymbol{A}$ of order n_1 and n_2 corresponding to two subdomains are given by

$$\begin{aligned} \boldsymbol{A}_1 &= \boldsymbol{R}_1 \boldsymbol{A} \boldsymbol{R}_1^T \\ \boldsymbol{A}_2 &= \boldsymbol{R}_2 \boldsymbol{A} \boldsymbol{R}_2^T \end{aligned}$$



Discretized Schwarz Method

- For discretized problem, *alternating Schwarz iteration* takes form

$$\begin{aligned} \boldsymbol{x}_{k+\frac{1}{2}} &= \boldsymbol{x}_k + \boldsymbol{R}_1^T \boldsymbol{A}_1^{-1} \boldsymbol{R}_1 (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_{k+\frac{1}{2}} + \boldsymbol{R}_2^T \boldsymbol{A}_2^{-1} \boldsymbol{R}_2 (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{k+\frac{1}{2}}) \end{aligned}$$

- This method is analogous to block Gauss-Seidel, but with overlapping blocks

- Overall iteration matrix has form

$$(\boldsymbol{I} - \boldsymbol{R}_1^T \boldsymbol{A}_1^{-1} \boldsymbol{R}_1 \boldsymbol{A})(\boldsymbol{I} - \boldsymbol{R}_2^T \boldsymbol{A}_2^{-1} \boldsymbol{R}_2 \boldsymbol{A})$$

so this is known as *multiplicative Schwarz method*

Discretized Schwarz Method

- We have as yet achieved no parallelism, since two subproblems must be solved sequentially for each iteration, but instead of Gauss-Seidel, we can use block Jacobi approach

$$\begin{aligned}\boldsymbol{x}_{k+\frac{1}{2}} &= \boldsymbol{x}_k + \boldsymbol{R}_1^T \boldsymbol{A}_1^{-1} \boldsymbol{R}_1 (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_{k+\frac{1}{2}} + \boldsymbol{R}_2^T \boldsymbol{A}_2^{-1} \boldsymbol{R}_2 (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k)\end{aligned}$$

  whose subproblems can be solved simultaneously

- With either Gauss-Seidel or Jacobi version, it can be shown that iteration converges at rate independent of mesh size, provided overlap between subdomains is sufficiently large (and independent of mesh size)

Discretized Schwarz Method

- Eliminating $\boldsymbol{x}_{k+\frac{1}{2}}$ in Jacobi version, we obtain

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + (\boldsymbol{R}_1^T \boldsymbol{A}_1^{-1} \boldsymbol{R}_1 + \boldsymbol{R}_2^T \boldsymbol{A}_2^{-1} \boldsymbol{R}_2)(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k)$$

  which is just Richardson iteration with *additive Schwarz preconditioner*

$$\boldsymbol{R}_1^T \boldsymbol{A}_1^{-1} \boldsymbol{R}_1 + \boldsymbol{R}_2^T \boldsymbol{A}_2^{-1} \boldsymbol{R}_2$$

- Symmetry of preconditioned system means it can be used in conjunction with conjugate gradient method to accelerate convergence

Discretized Schwarz Method

- Multiplicative Schwarz iteration matrix is not symmetric, but can be made symmetric by additional step with $\boldsymbol{A}_1^{-1}$ each iteration

$$
\begin{aligned}
\boldsymbol{x}_{k+\frac{1}{3}} &= \boldsymbol{x}_k + \boldsymbol{R}_1^T \boldsymbol{A}_1^{-1} \boldsymbol{R}_1 (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k) \\
\boldsymbol{x}_{k+\frac{2}{3}} &= \boldsymbol{x}_{k+\frac{1}{3}} + \boldsymbol{R}_2^T \boldsymbol{A}_2^{-1} \boldsymbol{R}_2 (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{k+\frac{1}{3}}) \\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_{k+\frac{2}{3}} + \boldsymbol{R}_1^T \boldsymbol{A}_1^{-1} \boldsymbol{R}_1 (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{k+\frac{2}{3}})
\end{aligned}
$$

which yields symmetric preconditioner that can be used in conjunction with conjugate gradient method to accelerate convergence

Many Overlapping Subdomains

- To achieve higher degree of parallelism with Schwarz method, we can apply two-domain algorithm recursively or use many subdomains
- If there are p overlapping subdomains, then define matrices $\boldsymbol{R}_i$ and $\boldsymbol{A}_i$ as before, $i = 1, \dots, p$
- Additive Schwarz preconditioner then takes form

$$\sum_{i=1}^{p} \boldsymbol{R}_i^T \boldsymbol{A}_i^{-1} \boldsymbol{R}_i$$

Many Overlapping Subdomains

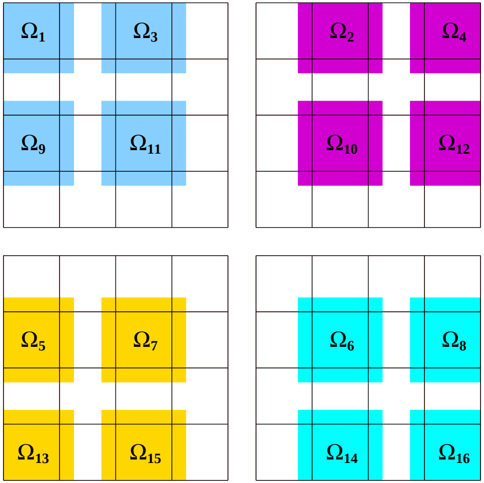
- Resulting generalization of block-Jacobi iteration is highly parallel, but not algorithmically scalable because convergence rate degrades as p grows
- Convergence rate can be restored by using coarse grid correction to provide global coupling
- If $\boldsymbol{R}_0$ and $\boldsymbol{R}_0^T$ are restriction and interpolation matrices between coarse and fine grids, and $\boldsymbol{A}_0 = \boldsymbol{R}_0 \boldsymbol{A} \boldsymbol{R}_0^T$, then additive Schwarz preconditioner becomes

$$\sum_{i=0}^{p} \boldsymbol{R}_i^T \boldsymbol{A}_i^{-1} \boldsymbol{R}_i$$

Many Overlapping Subdomains

- Multiplicative Schwarz iteration for p domains is defined analogously
- As with classical Gauss-Seidel vs. Jacobi, multiplicative Schwarz has faster convergence rate than corresponding additive Schwarz (though it still requires coarse grid correction to remain scalable)
- But unfortunately, multiplicative Schwarz appears to provide no parallelism, as p subproblems per iteration must be solved sequentially
- As with classical Gauss-Seidel, parallelism can be introduced by coloring subdomains to identify independent subproblems that can be solved simultaneously

Many Overlapping Subdomains

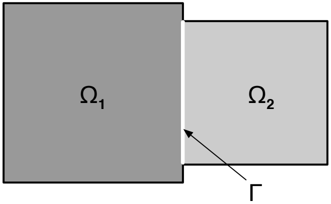


Coarse Grid Correction

- Coarse grid correction is necessary to retain algorithmic scalability as number of subdomains grows
- Reasonable choice for resolution of coarse grid is $\sqrt{h}$, where h is mesh size of underlying fine grid
- Options for solving coarse grid problem include
  - Partition coarse grid problem across processors in same manner as fine grid problem
  - Solve coarse grid problem serially on one processor and broadcast results
  - Solve entire coarse grid problem redundantly in parallel on each processor
- Choice among these depends on relative size of coarse grid problem and relative speeds of communication and computation

Non-Overlapping Subdomains

- We now consider adjacent subdomains whose only points in common are along their mutual boundary



- We partition indices of unknowns in corresponding discrete linear system into three sets, S_1 and S_2 corresponding to interior nodes in Ω_1 and Ω_2, respectively, and S_3 corresponding to interface nodes in Γ

Non-Overlapping Subdomains

- Partitioning matrix and right-hand-side vector accordingly, we obtain symmetric block linear system

$$\begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{O} & \boldsymbol{A}_{13} \\ \boldsymbol{O} & \boldsymbol{A}_{22} & \boldsymbol{A}_{23} \\ \boldsymbol{A}_{13}^T & \boldsymbol{A}_{23}^T & \boldsymbol{A}_{33} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \boldsymbol{b}_3 \end{bmatrix}$$

- Zero blocks result from assumption that nodes in Ω_1 are not directly connected to nodes in Ω_2, but only through interface nodes in Γ

Schur Complement

- Block LU factorization of matrix $\boldsymbol{A}$ yields

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{O} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{A}_{13}^T \boldsymbol{A}_{11}^{-1} & \boldsymbol{A}_{23}^T \boldsymbol{A}_{22}^{-1} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{O} & \boldsymbol{A}_{13} \\ \boldsymbol{O} & \boldsymbol{A}_{22} & \boldsymbol{A}_{23} \\ \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{S} \end{bmatrix}$$

where *Schur complement* matrix $\boldsymbol{S}$ is given by

$$\boldsymbol{S} = \boldsymbol{A}_{33} - \boldsymbol{A}_{13}^T \boldsymbol{A}_{11}^{-1} \boldsymbol{A}_{13} - \boldsymbol{A}_{23}^T \boldsymbol{A}_{22}^{-1} \boldsymbol{A}_{23}$$

Schur Complement

- We can now determine interface unknowns $\boldsymbol{x}_3$ by solving system

  $$\boldsymbol{S}\boldsymbol{x}_3 = \hat{\boldsymbol{b}}_3$$

  where

  $$\hat{\boldsymbol{b}}_3 = \boldsymbol{b}_3 - \boldsymbol{A}_{13}^T \boldsymbol{A}_{11}^{-1} \boldsymbol{b}_1 - \boldsymbol{A}_{23}^T \boldsymbol{A}_{22}^{-1} \boldsymbol{b}_2$$

- Remaining unknowns are then given by

  $$\begin{aligned} \boldsymbol{x}_1 &= \boldsymbol{A}_{11}^{-1}(\boldsymbol{b}_1 - \boldsymbol{A}_{13}\boldsymbol{x}_3) \\ \boldsymbol{x}_2 &= \boldsymbol{A}_{22}^{-1}(\boldsymbol{b}_2 - \boldsymbol{A}_{23}\boldsymbol{x}_3) \end{aligned}$$

  which can be computed simultaneously

Schur Complement

- Schur complement matrix $\boldsymbol{S}$ is expensive to compute and is generally dense even if $\boldsymbol{A}$ is sparse
- But if Schur complement system $\boldsymbol{S}\boldsymbol{x}_3 = \hat{\boldsymbol{b}}_3$ is solved iteratively, then $\boldsymbol{S}$ need not be formed explicity
- Matrix-vector multiplication by $\boldsymbol{S}$ requires solution in each subdomain, implicitly involving $\boldsymbol{A}_{11}^{-1}$ and $\boldsymbol{A}_{22}^{-1}$, which can be done independently in parallel
- Conditioning of $\boldsymbol{S}$ is generally better than that of $\boldsymbol{A}$, typically $\mathcal{O}(h^{-1})$ instead of $\mathcal{O}(h^{-2})$ for mesh size h, but *interface preconditioner* is still needed to accelerate convergence

Many Non-Overlapping Subdomains

- To achieve higher degree of parallelism with Schur method, we can apply two-domain algorithm recursively or use many subdomains
- If there are p non-overlapping subdomains, let I be set of indices of interior nodes of subdomains and B be set of indices of interface nodes separating subdomains
- Then discrete linear system has block form

$$\begin{bmatrix} \boldsymbol{A}_{II} & \boldsymbol{A}_{IB} \\ \boldsymbol{A}_{IB}^T & \boldsymbol{A}_{BB} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_I \\ \boldsymbol{x}_B \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_I \\ \boldsymbol{b}_B \end{bmatrix}$$

where $\boldsymbol{A}_{II} = \operatorname{diag}(\boldsymbol{A}_{11}, \ldots, \boldsymbol{A}_{pp})$ is block diagonal

Many Non-Overlapping Subdomains

- Block LU factorization of matrix $\boldsymbol{A}$ yields system

$$\boldsymbol{S}\boldsymbol{x}_B = \hat{\boldsymbol{b}}_B$$

  where Schur complement matrix $\boldsymbol{S}$ is given by

$$\boldsymbol{S} = \boldsymbol{A}_{BB} - \boldsymbol{A}_{IB}^T \boldsymbol{A}_{II}^{-1} \boldsymbol{A}_{IB}$$

  and $\hat{\boldsymbol{b}}_B = \boldsymbol{b}_B - \boldsymbol{A}_{IB}^T \boldsymbol{A}_{II}^{-1} \boldsymbol{b}_I$

- As before, this system can be solved iteratively without forming $\boldsymbol{S}$ explicitly, and interface preconditioner is used to accelerate convergence

Many Non-Overlapping Subdomains

- Interior unknowns are then given by

$$\boldsymbol{x}_I = \boldsymbol{A}_{II}^{-1}(\boldsymbol{b}_I - \boldsymbol{A}_{IB}\boldsymbol{x}_B)$$

- All solves involving $\boldsymbol{A}_{II}^{-1}$, both in iterative phase for computing interface unknowns and subsequent computation of interior unknowns, can be performed on all subdomains in parallel because $\boldsymbol{A}_{II}$ is block diagonal

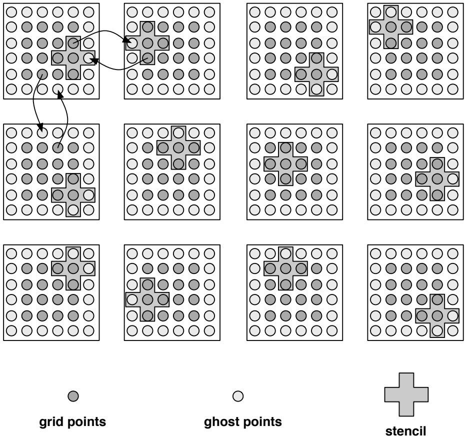
Parallel Computation with Grids

- Two basic approaches to parallel numerical solution of PDEs
  - Domain decomposition based on original PDE, which yields multiple problems to be solved in parallel, each on different subdomain
  - Parallel implementation of serial algorithm for solving discretized version of original problem
- Either approach ultimately leads to distribution of discrete mesh or grid across processors
- Communication between processors required to provide interface between subdomains or for parallel solution of discrete problem (e.g., matrix-vector multiplication for iterative methods)

Parallel Computation with Grids

- We will now consider some practical issues in implementing such grid-based computations in parallel
- For simplicity, we focus on regular finite difference grids in 2-D; unstructured meshes (e.g., finite element) and 3-D are slightly more complicated but issues are similar
- Communication is required whenever stencil for given point includes points on another processor
- For greater efficiency in message passing, all such communications for given step (iteration or time step) are bundled together by maintaining "*ghost*" points, overlapping between two subgrids

Ghost Points



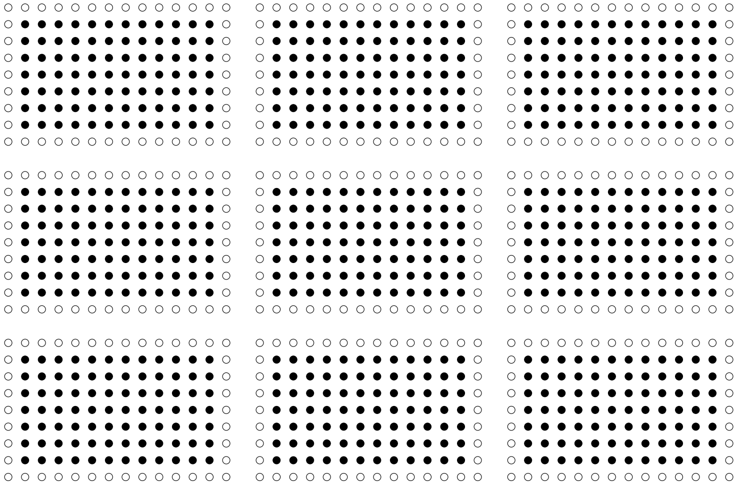
Ghost Points

- For simple five-point stencil, single layer of ghost points suffices
- For higher-order approximation with larger stencil, more layers of ghost points may be needed for wider overlap
- Wider overlap may also benefit some algorithms, such as Gauss-Seidel with red-black ordering
- For standard nine-point stencil, corner ghost points are required, but communication between “diagonal” processors can be avoided by properly synchronizing successive horizontal and vertical communications

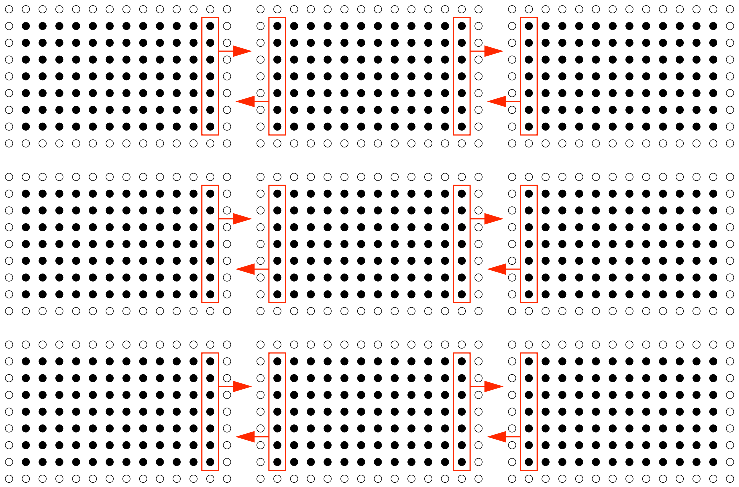
Diagonal Trick

- In nine-point stencil, there are two types of ghost point communication: edges must be exchanged with North/South/East/West neighbors, and corners must be exchanged with NE/SE/NW/SW neighbors
- Why not simply perform eight communication steps?
- Time to exchange ghost points can be estimated as $T_{comm} = 4(t_s + nt_w) + 4(t_s + t_w)$
- Recall that $t_s \gg t_w$, so time becomes $T_{comm} \approx 8t_s + 4nt_w$
- Can we avoid t_s part of cost of moving corner points?

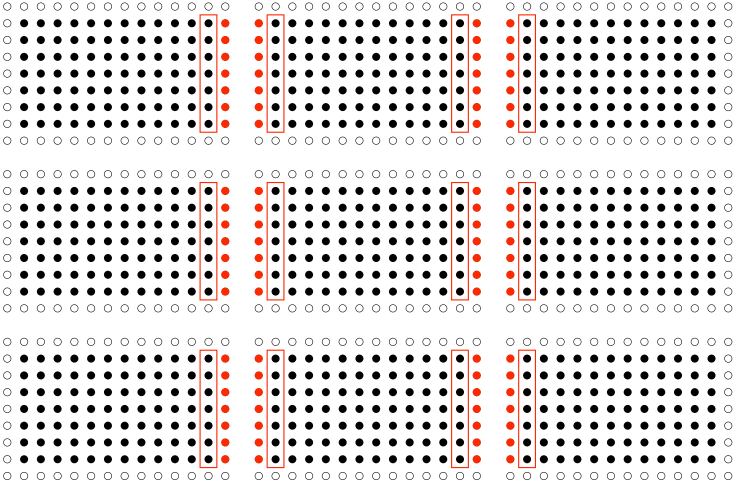
Diagonal Trick



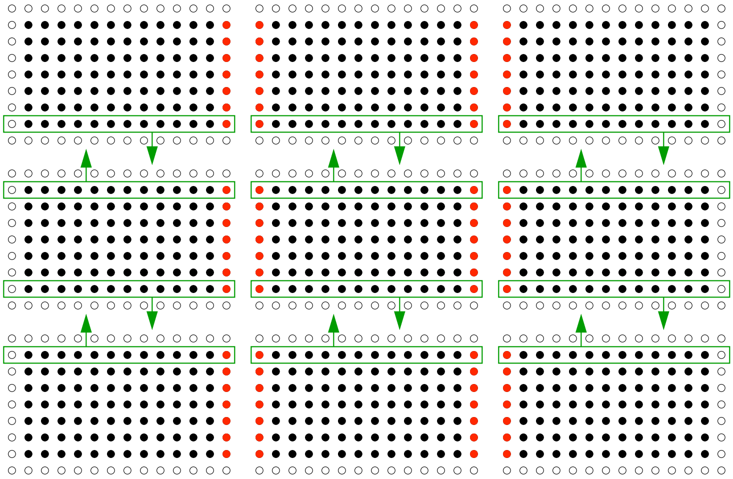
Diagonal Trick



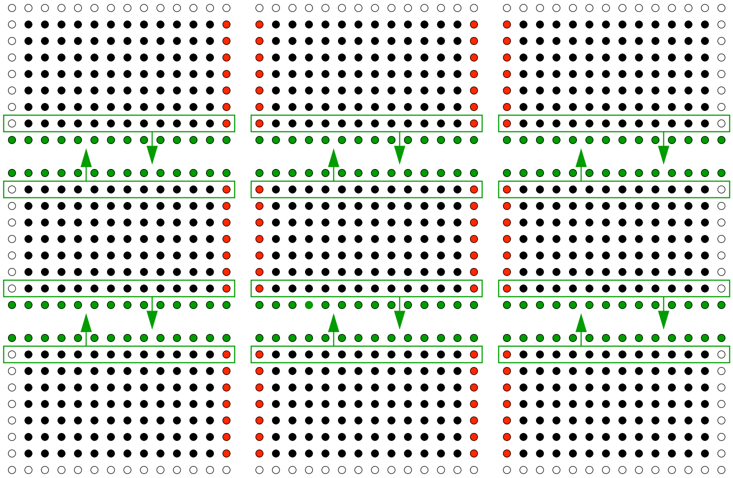
Diagonal Trick



Diagonal Trick



Diagonal Trick



Taking Multiple Time Steps

- In explicit methods for time-dependent PDEs, communication of ghost points can be significant cost
- We saw that *diagonal trick* reduces number of separate messages, improving parallel performance
- In time-dependent problems, there is one exchange per time step; for 2-D problem, each time step takes communication time

$$T_{comm} = 4(t_s + nt_w)$$

  using diagonal trick if necessary
- Can do better if willing to perform redundant computation

Parallel Iterative Methods

- For some iterative solvers, such as Jacobi and red-black Gauss-Seidel, serial and parallel versions are equivalent and produce same results
- In parallel setting, with grid partitioned across processors, additional options arise that are not relevant in serial context, such as using standard Gauss-Seidel within each processor and Jacobi between processors
- Although iterative methods such as Jacobi and red-black Gauss-Seidel often yield good parallel efficiency, their relatively slow asymptotic convergence rate limits their usefulness in practice

Smoothers

- Stationary iterative methods, such as Jacobi and Gauss-Seidel, usually make fairly rapid initial progress in reducing error before settling into slow asymptotic phase
- In particular, they reduce high-frequency (i.e., oscillatory) components of error rapidly, but reduce low-frequency (i.e., smooth) components of error much more slowly
- For this reason, such methods are sometimes called *smoothers*

Smoothers

- Smooth or oscillatory components of error are relative to grid on which solution is defined
- Component that appears smooth on fine grid may appear oscillatory when sampled on coarser grid
- If we apply smoother on coarser grid, then we may make rapid progress in reducing this (now oscillatory) component of error
- After few iterations of smoother, results can then be interpolated back to fine grid to produce solution that has both higher-frequency and lower-frequency components of error reduced

Multigrid

- This idea can be extended to multiple levels of grids, so that error components of various frequencies can be reduced rapidly, each at appropriate level
- Transition from finer grid to coarser grid involves *restriction* or *injection*
- Transition from coarser grid to finer grid involves *interpolation* or *prolongation*

Residual Equation

- If $\hat{\boldsymbol{x}}$ is approximate solution to $\boldsymbol{Ax} = \boldsymbol{b}$, with residual $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\hat{\boldsymbol{x}}$, then error $\boldsymbol{e} = \boldsymbol{x} - \hat{\boldsymbol{x}}$ satisfies *residual equation*

$$\boldsymbol{Ae} = \boldsymbol{r}$$

- Thus, in improving approximate solution we can work with just this *residual equation* involving error and residual, rather than solution and original right-hand side

- One advantage of residual equation is that zero is reasonable starting guess for its solution

Two-Grid Algorithm

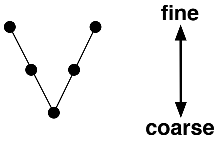
1. On fine grid, use few iterations of *smoother* to compute approximate solution $\hat{\boldsymbol{x}}$ for system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$
2. Compute residual $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\hat{\boldsymbol{x}}$
3. *Restrict* residual to coarse grid
4. On coarse grid, use few iterations of *smoother* on residual equation to obtain coarse-grid approximation to error
5. *Interpolate* coarse-grid correction to fine grid to obtain improved approximate solution on fine grid
6. Apply few iterations of *smoother* to corrected solution on fine grid

Multigrid Algorithm

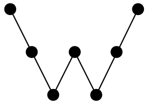
- Multigrid method results from recursion in Step 4: coarse grid correction is itself improved by using still coarser grid, and so on down to some bottom level
- Computations become progressively cheaper on coarser and coarser grids because systems become successively smaller
- In particular, direct method may be feasible on coarsest grid if system is small enough
- There are many possible strategies for cycling through various grid levels

Multigrid Cycles

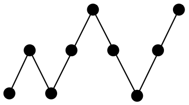
- *V-cycle* goes from finest grid goes down through successive levels to coarsest grid and then back up to finest grid



- *W-cycle* zig-zags among lower level (and cheaper) grids before going back up to finest grid



- *Full multigrid* bootstraps coarse solution up through grid levels, ultimately reaching finest grid



Multigrid

- By exploiting strengths of underlying iterative smoothers and avoiding their weaknesses, multigrid methods are capable of extraordinarily good performance
- At each level, smoother reduces oscillatory component of error rapidly, at rate independent of mesh size h, since few iterations of smoother, often only one, are performed at each level
- Since all components of error appear oscillatory at some level, convergence rate of entire multigrid scheme should be rapid and independent of mesh size, in contrast to most other iterative methods

Multigrid

- Moreover, cost of entire cycle of multigrid is only modest multiple of cost of single sweep on finest grid
- As result, multigrid methods are among most powerful methods available for solving sparse linear systems arising from PDEs
- In many cases, they are capable of converging to within truncation error of discretization at cost proportional to number of grid points, which is much faster than most other methods

Parallel Multigrid

- Many aspects of multigrid algorithms are readily implemented in parallel
  - Point smoothers, such as Jacobi and multi-color Gauss-Seidel
  - Residual computation
  - Restriction (fine-to-coarse)
  - Interpolation (coarse-to-fine)
- Other aspects are more problematic
  - Sequential cycling through grids
  - Parallel efficiency for coarse grids

Parallel Multigrid

- Compared with fine grids, coarse grids
  - Inherently allow less parallelism
  - Incur higher communication cost relative to computation
  - Risk poor load balance (some processors may even be idle for coarsest grids)
  - Do not necessarily grow as overall problem grows
- For these reasons, parallel implementations of multigrid try to minimize time spent on coarse grids

Parallel Multigrid

For model problem with n grid points on finest grid, complexity of parallel multigrid is

- $\Theta(\log n)$ for V-cycle
- $\Theta(\log^2 n)$ for FMG
- $\Theta(\sqrt{n}\,)$ for W-cycle

Parallel Multigrid

- For coarse grids, communication/computation ratio could be improved by using *fewer* processors, and load balance could be improved by redistributing work across processors
- However, such measures affect other aspects of algorithm negatively; for example, restriction and interpolation would no longer be local operations within individual processors

Parallel Multigrid

- Because of parallel inefficiencies associated with coarse grids, alternatives have been proposed to enhance parallelism in multigrid
- *Additive multigrid* performs smoothing on all grid levels simultaneously, but convergence is not guaranteed, so it is used as preconditioner
- *Parallel superconvergent multigrid* performs smoothing on multiple grids at each level simultaneously, thereby (hopefully) accelerating convergence

Parallel Multigrid

- Such variants of multigrid motivated by parallelism are not equivalent to serial multigrid and sacrifice some of its serial efficiency to gain greater parallelism
- Whether such strategies actually reduce overall time to solution depends on specific problem and parallel system
- Even with “classical” multigrid, serial superiority of FMG over V-cycle may outweigh parallel superiority of V-cycle over FMG

Multigrid and Domain Decomposition

- Domain decomposition with coarse problem can be viewed as two-level multigrid
- As with domain decomposition, restriction operator should often be transpose of interpolation operator, whose choice is critical for success of both methods
- Parallel solution of coarse problem is same
- Domain decomposition involves less communication per iteration than parallel multigrid but may require (a constant factor) more iterations

Scalability in Solving PDEs

- How scalable are numerical methods for solving PDEs?
- Consider 3-D Poisson equation, discretized with finite differences or low-order finite elements on $n \times n \times n$ grid
- Decompose in three dimensions, yielding cubes of size $n/p^{1/3} \times n/p^{1/3} \times n/p^{1/3}$
- Method involves one ghost cell exchange, one dot product (e.g., for CG), and local evaluation of matrix-vector product

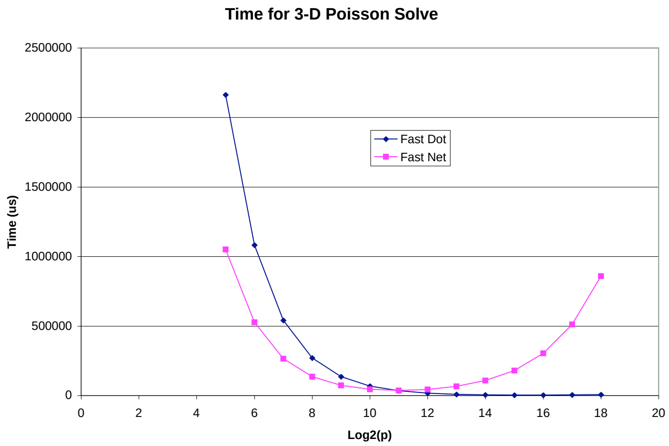
Cost Model

- Per iteration cost is

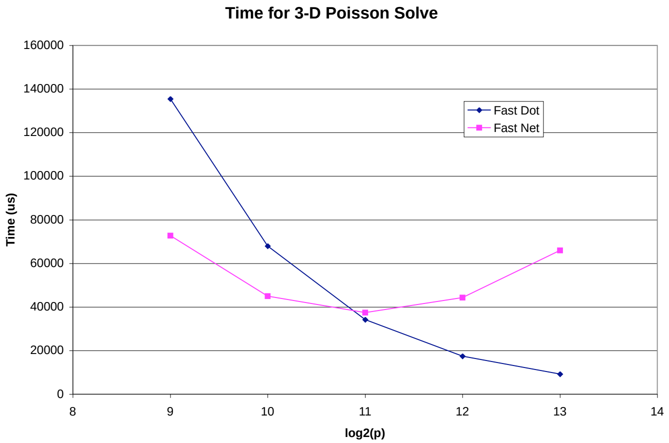
$$T = \frac{n^3}{p} t_c + 6 \left(t_s + t_w \left(\frac{n}{p^{1/3}} \right) \right) + 2(t_s + t_w) \log p$$

- If used as preconditioner or as part of parallel multigrid or domain decomposition, cost is similar and number of iterations can be independent of p
- Consider two systems: one with fast network (low t_w) and one with fast network for reduction operation (replaces $\log p$ term)

Scaling for 3-D Poisson



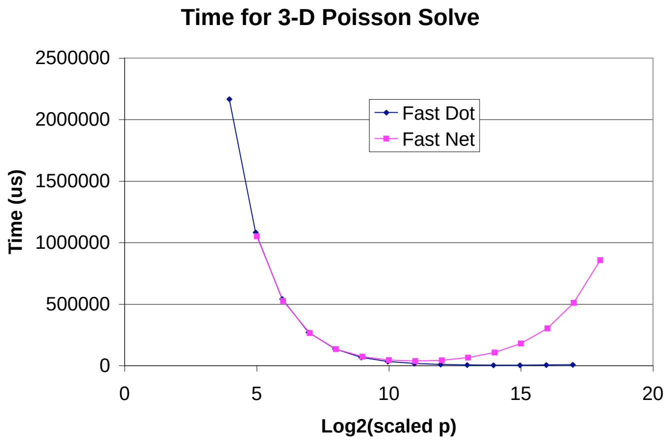
Scaling Details



Comparing Systems

- What is best way to compare two systems?
- Previous graphs have compared with respect to *processors*
- What if processors are multicore, or have special thread or vector processors (e.g., Cray MTA or Cray X1), or have different costs?
- What if processor clock rates are very different?
    - e.g., Blue Gene uses relatively slow clock, but can pack many more processors into same cabinet than system with faster (but larger and hotter) processors
- Possible ways to normalize include
    - Cost/processor
    - Electric power/processor
    - Machine room area/processor
    - Achieved performance/processor

3-D Poisson Scaled by Performance



Fault Detection and Fault Tolerance

- We have assumed that software and hardware that execute algorithm are infallible
- At scales of 100,000 processors, errors may happen once every few days (more often with commodity hardware)
- What is role for numerical algorithm in managing faults?

Fault Detection

- Numerical algorithms often include additional properties that are not explicitly used in computation
- Conservation is one example: many hyperbolic equations express *conservation laws*, in which some quantity is constant
- Another example is *maximum principle*, which holds for class of elliptic PDEs
- Checking that such properties are preserved by computation provides check on presence of fault

Fault Tolerance and Recovery

- Fault tolerance and recovery—ability to continue an accurate computation after fault—is more difficult
- For many PDE problems, easiest approach is checkpoint-restart
- It is sometimes possible, with small loss of accuracy, to regenerate lost data using other available data (by using Green's theorem for elliptic problems, for example)
- For time-dependent problems, it may be possible to regenerate just missing data from prior checkpoint
- At large scale, numerical algorithms must address issues of fault detection and tolerance

Summary and Suggestions

- Use redundant computation to reduce communication, increasing parallel efficiency
  - Limits *maximum* efficiency
- Consider alternate approximation or solution methods to improve concurency
  - E.g., domain decomposition; some block decompositions
- Parallelize best methods
  - E.g., multigrid, Krylov methods with high-quality preconditioners, not block Jacobi

References – Domain Decomposition

- T. F. Chan and T. P. Mathew, Domain decomposition algorithms, *Acta Numerica* 3:61-143, 1994
- A. Quarteroni and A. Valli, *Domain Decomposition Methods for Partial Differential Equations*, Oxford Univ. Press, 1999
- B. F. Smith, Domain decomposition methods for partial differential equations, D. E. Keyes, A. Sameh, and V. Venkatakrishnan, eds., *Parallel Numerical Algorithms*, pp. 225-243, Kluwer, 1997
- B. F. Smith, P. E. Bjørstad, and W. D. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge Univ. Press, 1996
- A. Toselli and O. Widlund, *Domain Decomposition Methods: Algorithms and Theory*, Springer, 2005

References – Multigrid

- P. Bastian, W. Hackbusch, and G. Wittum, Additive and multiplicative multigrid — a comparison, *Computing* 60:345-364, 1998
- J. H. Bramble, *Multigrid Methods*, Pitman, 1993
- W. L. Briggs, V. E. Henson and S. F. McCormick, *A Multigrid Tutorial*, 2nd ed., SIAM, 2000
- W. Hackbusch, *Multigrid Methods and Applications*, Springer-Verlag, 1985
- U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, 2001
- P. Wesseling, *An Introduction to Multigrid Methods*, John Wiley & Sons, 1992

References – Parallel Multigrid

- A. Brandt, Multigrid solvers on parallel computers, M. H. Schultz, ed., *Elliptic Problem Solvers*, pp. 39-83, Academic Press, 1981
- T. F. Chan and Y. Saad, Multigrid algorithms on the hypercube multiprocessor, *IEEE Trans. Comput.*, 35:969-977, 1986
- T. F. Chan and R. Schreiber, Parallel networks for multigrid algorithms: architecture and complexity, *SIAM J. Sci. Stat. Comput.* 6:698-711, 1985
- A. Greenbaum, A multigrid method for multiprocessors, *Appl. Math. Comput.* 19:75-88, 1986
- H. Hoppe and H. Muhlenbein, Parallel adaptive full-multigrid methods on message-based multiprocessors, *Parallel Computing* 3:269-287, 1986
- J. E. Jones and S. F. McCormick, Parallel multigrid methods, D. E. Keyes, A. Sameh, and V. Venkatakrishnan, eds., *Parallel Numerical Algorithms*, pp. 203-224, Kluwer, 1997