CS 598: Communication Cost Analysis of Algorithms

Lecture 20: Communication-efficient preconditioning, domain decomposition, graph partitioning

Edgar Solomonik

University of Illinois at Urbana-Champaign

October 31, 2016

Iterative computation of ILU

Recent idea: Chow and Patel, "Fine-Grained Parallel Incomplete LU Factorization", 2015

- interpret ILU factorization as system of bilinear equations
- the unknowns are L_{ij} for $i < j$, U_{ij} for $i \geq j$ with $(i,j) \in S$ and we have constraints/objectives

$$\sum_{k=1}^{\min(i,j)} L_{ik} U_{kj} = A_{ij}, \quad \forall (i,j) \in S$$

- can be reformulated as $(L, U) = \mathcal{F}(L, U)$, where function $\mathcal{F}$ performs

$$L_{ij} = \frac{1}{U_{jj}} \Big(A_{ij} - \sum_{k=1}^{j-1} L_{ik} U_{kj} \Big), \quad U_{ij} = A_{ij} - \sum_{k=1}^{i-1} L_{ik} U_{kj}$$

- use equations as fixed-point iteration $(L^{(h+1)}, U^{(h+1)}) = \mathcal{F}(L^{(h)}, U^{(h)})$

Iterative computation of ILU, contd.

We can consider applying the equations in $\mathcal{F}$ in various orderings

- use latest available values of L and U entries
- the "Gaussian elimination ordering" just computes ILU as before
- other orderings have more parallelism but slower convergence
- need a good starting guess to speed-up convergence
- approximate solution ok, exact ILU is an inexact factorization anyway
- convergence in k fixed-point iterations implies factor of k more work
- Q: is changing the ordering of equations the same as changing the order of rows/cols in standard ILU?
- A: no they are generally different, the latter changes the dependency graph, the former just changes the order of 'relaxations' of its edges

Iterative computation of ILU for cube DAG

Recall $n^{1/3} \times n^{1/3} \times n^{1/3}$ cube DAG

- standard cube DAG execution had cost

$$T_{\text{ILU[0]-cube}}(n, P) = O\left(n/P \cdot \gamma + \frac{n^{2/3}}{\sqrt{P}} \cdot \beta + \sqrt{P} \cdot \alpha\right)$$

- iterative computation with k iterations has cost

$$T_{\text{ILU[0]-iter-cube}}(n, P, k) = O\left(kn/P \cdot \gamma + k\frac{n^{2/3}}{P^{2/3}} \cdot \beta + k \cdot \alpha\right)$$

- with Gaussian-elimination ordering within blocks $k = O(\sqrt{P})$
- when $k = \sqrt{P}$, iterative approach does factor of $\sqrt{P}$ more work and $P^{1/3}$ more communication
- however, if we get a satisfactory ILU in $k = O(1)$ iterations, obtain $\Theta(P^{1/6})$ less communication and $\Theta(\sqrt{P})$ fewer synchronizations

Approximating the inverse

Rather than looking for $A \approx M = LU$ to apply M^{-1}, we can try to directly approximate $W = M^{-1} \approx A^{-1}$

- formally, we will try to minimize $||I - AW||_F^2$
- we would like W to be sparse, while A^{-1} may be dense
- we can write this again as an optimization problem which minimizes

$$\min_{w_j} ||e_j - Aw_j||_2^2$$

  where w_j, e_j are the jth columns of W, I
- we can compute each w_j by a sparse iterative method with a sparse initial guess and try to preserve sparsity
  - each SpMSpV will spread nonzeros, so we can try doing $O(1)$ iterations
  - or we can compute (some approximation to) the full vector, then drop small entries
- to lower number of parallel steps, can compute many w_j at a time
- key benefit: **applying the preconditioner is SpMV** and not TRSV

Polynomial preconditioning

Motivation and definition

- recall that we can write the inverse of A in terms of

$$\mathcal{S}(X) = I + X + X^2 + X^3 \ldots$$

as $\mathcal{S}(X) - \mathcal{S}(X)X = I$ and therefore

$$\mathcal{S}(I - A) - \mathcal{S}(I - A)(I - A) = \mathcal{S}(I - A)A = I$$

so $A^{-1} = \mathcal{S}(I - A)$

- further, recall that in Krylov subspace methods, we want to span the space $\{x, Ax, A^2x \ldots .\}$
- the idea of polynomial preconditioning is to use the preconditioner $M^{-1} = \rho(A)$ and solve

$$\rho(A)Ax = \rho(A)b$$

where $\rho(A)$ is a polynomial in A

- Chebyshev polynomials improve conditioning for many systems

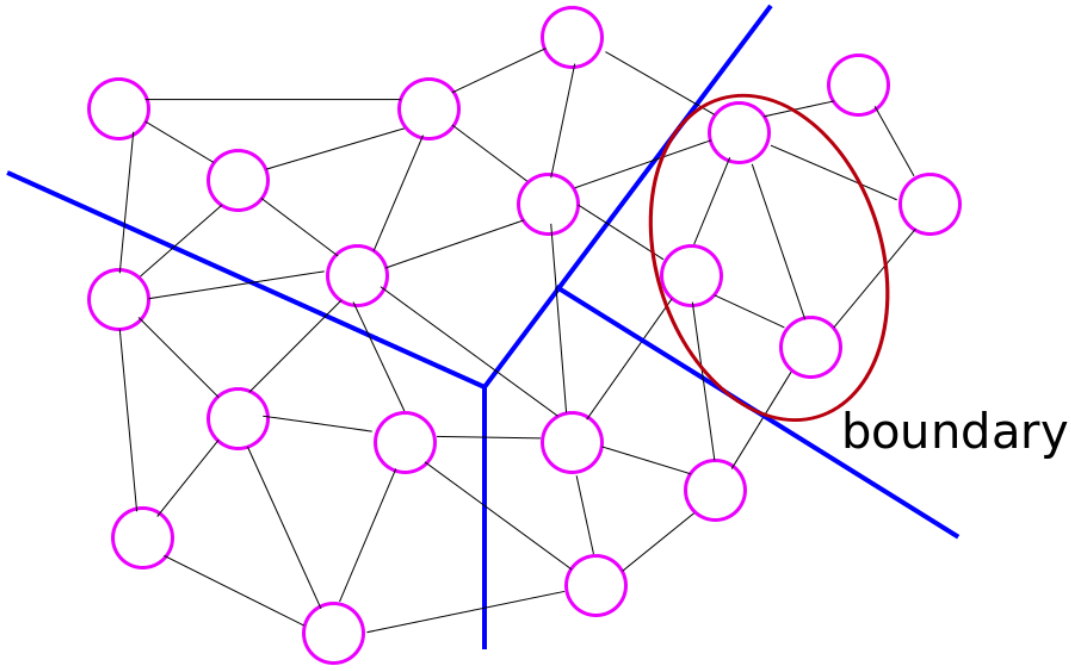
Applying a polynomial preconditioner

Polynomial preconditioners are very easy to apply

- never compute $\rho(A)$ explicitly, but compute $\rho(A)v$ by steps of the form $z + Aw$
- in contrast to ILU there is no triangular solve to perform
- in-time blocking to avoid communication is now possible
  - not just possible, but necessary in practice to achieve numerical stability
- unfortunately this simple preconditioner is not as effective as methods that spread information globally, such as multigrid
  - expect to improve convergence by order of polynomial, but need to do as many more SpMVs

Short pause

Partitioning



General idea in domain decomposition

- assign a subdomain to each processor
- solve independent problems on inner subdomains in parallel
- work with reduced problem to resolve domain boundaries

Partitioning in matrix form

Vertex partitioning into 3 parts $\Rightarrow$ row/column ordered adjacency matrix:

$$A = \begin{bmatrix} B & E \\ F & C \end{bmatrix} = \begin{bmatrix} B_1 & 0 & 0 & E_1 & 0 & 0 \\ 0 & B_2 & 0 & 0 & E_2 & 0 \\ 0 & 0 & B_3 & 0 & 0 & E_3 \\ F_1 & 0 & 0 & C_{11} & C_{12} & C_{13} \\ 0 & F_2 & 0 & C_{21} & C_{22} & C_{23} \\ 0 & 0 & F_3 & C_{31} & C_{32} & C_{33} \end{bmatrix}$$

where $C_{ij} = 0$ if boundaries of partition i and j are disconnected

- B encodes edges between inner nodes in each subdomain
- E and F are the connections between inner nodes and boundary of each subdomain
- C are the connections between boundaries of different subdomains
- a good partitioning should have C of much smaller dimension than B
- when the number of partitions is large, C can be very sparse

Schur complement methods

Consider the partitioned matrix $A = \begin{bmatrix} B & E \\ F & C \end{bmatrix}$

- a block LU factorization would compute the Schur complement

$$C - FB^{-1}E$$

- the Schur complement allows us to solve linear equations

$$\begin{bmatrix} B & E \\ F & C \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}$$

- first write x in terms of y

$$Bx + Ey = f \quad \Rightarrow \quad x = B^{-1}f - B^{-1}Ey$$

- the substitute x into the second equation $Fx + Cy = g$

$$FB^{-1}f - FB^{-1}Ey + Cy = g \quad \Rightarrow \quad (C - FB^{-1}E)y = g - FB^{-1}f$$

- computing $B^{-1}E$ and $B^{-1}f$ would allow us to solve a new set of linear systems to get y and cheaply compute x

Schur complement preconditioning

Our main problem is to solve $(C - FB^{-1}E)y = g - FB^{-1}f$

- recall that B is block diagonal and E, F are also structured
- computing $E' = B^{-1}E$ can be done via $E'_i = B_i^{-1}E_i$ for each i
  - yields coefficients for equations between boundary vertices within each subdomain
  - E'_i is usually dense, unlike B_i
- computing $f' = B^{-1}f$ can also be done via $f'_i = B_i^{-1}f_i$ for each i
  - solves within each subdomain, transforming linear system
- we now have $(C - FE')y = g - Ff'$
  - can obtain explicit form of linear system by multiplications alone
  - FE' is block diagonal with blocks $F_iE'_i$

Cost of Schur complement preconditioning

We will usually have P partitions (one per processor)

- the reduction to the new linear system is embarrassingly parallel
- interprocessor communication cost is effectively zero
- if number of vertices in each domain is n/P and each boundary has $(n/P)^{(d-1)/d}$ vertices
  - Q: what is the dimension of C?
  - A: $P \cdot (n/P)^{(d-1)/d} = n(P/n)^{1/d}$
  - the fill we create by the Schur complement updates connects all nodes within each subdomain boundary
  - so the number of new nonzeros in C is roughly

$$P\left((n/P)^{(d-1)/d}\right)^2 = \frac{n^{2(d-1)/d}}{P^{(d-2)/d}}$$

  - for $d = 2$ this is n, so about as many nonzeros as in A
  - for $d = 3$ this is $n^{4/3}/P^{1/3}$, which is $O(n^{1/3}/P^{1/3})$ more than in A

Implicit Schur complement preconditioning

Rather than computing $C - FB^{-1}E$, we can solve the linear system

$$(C - FB^{-1}E)y = g - FB^{-1}f$$

by computing $z = (F(B^{-1}(Ew)))$ whenever necessary

- requires an 'inner' method for solving $B^{-1}(Ew)$
- for each stencil application to the reduced system, we propagate information fully within each subdomain
- more useful computation performed within the local subdomains, for the same communication cost
- a downside is that its not possible to do in-time blocking

Coordinate-based partitioning

Domain decomposition methods require graph partitioning

- first consider partitioning graphs embedded in d-dimensional space
- we expect to have *coordinates* for finite element meshes
- to get good partitions, still need to tie *connectivity* to *locality*
- Miller, Teng, Thurston, and Vavasis (1997) provide a good notion of locality and an efficient graph partitioning algorithm

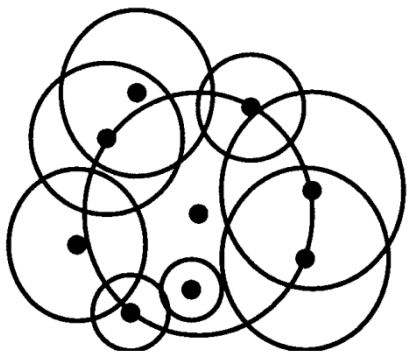
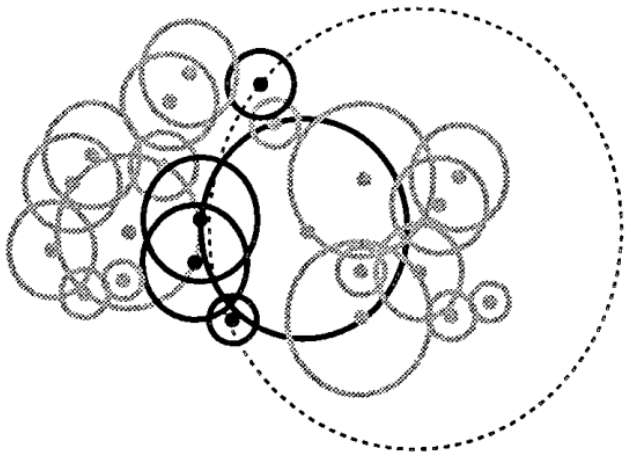


FIG. 1. A 3-ply system.

  k-ply neighborhood is a set of n balls with $\leq k$ intersecting anywhere
- local graphs of interest can be embedded into k-ply neighborhoods

Partitioning of k-ply neighborhoods

Miller, Teng, Thurston, and Vavasis (1997) give an algorithm to find a sphere that partitions a neighborhood and intersects $O(k^{1/d} n^{(d-1)/d})$ balls



- translates into vertex separators of size $O(n^{(d-1)/d})$ for meshes with constant *aspect ratio* – max relative distance of edges in space
- algorithm based on finding *centerpoints*, every hyperplane that includes one is a good partition
- centerpoints can be computed by a linear program and well-approximated by computing centerpoints of small random subsets

Coordinate-free partitioning

Graph partitioning is much harder to do in general

- some techniques leverage BFS or graph hierarchies constructed using maximal independent sets
- spectral partitioning is an elegant algebraic approach
- the *Laplacian* matrix L of a graph $G = (V, E)$ is

$$L_{ij} = \begin{cases} i = j & : \text{degree}(V(i)) \\ (i,j) \in E & : -1 \\ (i,j) \notin E & : 0 \end{cases}$$

- the eigenvector of L with the second smallest eigenvalue (*the Fiedler vector*) provides a good partition of G!

The Fiedler vector

Why is the second smallest eigenvector w useful?

- the smallest eigenvector has eigenvalue zero and is a constant vector

$$\sum_j L_{ij} = \text{degree}(V(i)) + \sum_{(i,j)\in E} -1 = 0$$

- we can define two partitions by sorting w and taking the smallest $n/2$ values to be one partition
- consider two equal partitions V_1 and V_2 with a cut $n_c = |V_1 \times V_2 \cap E|$
- define vector v to be 1 for all vertices in V_1 and -1 for vertices in V_2
- Q: if $n_c = 0$ what would like Lv look like?
- A: $Lv = 0$, if we order vertices in V_1 before V_2 to define L and v,

$$Lv = \begin{bmatrix} L_1 & 0 \\ 0 & L_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} L_1 v_1 \\ L_2 v_2 \end{bmatrix} = 0$$

where L_1 and L_2 are Laplacians of disjoint subgraphs and v_1, v_2 are constant vectors

Partitioning using the Fiedler vector

More generally, we have

$$Lv = \begin{bmatrix} L_{11} & L_{21} \\ L_{12} & L_{22} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

where for undirected graphs $||\operatorname{vec}(L_{12})||_1 = ||\operatorname{vec}(L_{21})||_1 = n_c$

- now note that $L_{21}v_2 > 0$ is added to (positive vector) v_1 and $L_{12}v_1 < 0$ is added to (negative vector) v_2
- so the vectors representing the two partitions grow depending on how many edges there are in $V_1 \times V_2 \cap E$
- the smallest eigenvector has one cluster of vertices and eigenvalue 0
- the second smallest eigenvector provides an imbalance (partitioning) with minimal resistance (push-back between the partitions)