CS 598: Communication Cost Analysis of Algorithms

Lecture 19: Preconditioning via incomplete LU

Edgar Solomonik

University of Illinois at Urbana-Champaign

October 26, 2016

Preconditioning for iterative solvers

To accelerate convergence, we can try to find *preconditioner* M and solve

$$M^{-1}Ax = M^{-1}b$$

- want M^{-1} to be close to A^{-1} or to improve spectral radius of A
- at the same time need M^{-1} to be easy to apply, iterative methods, e.g. Richardson iteration will 'multiply' by M^{-1} at every iteration

$$x_{i+1} = x_i - \gamma_i M^{-1}(Ax_i - b)$$

- generally, we want M to be structured, e.g. diagonal, block-diagonal, or factorized into sparse triangular matrices with not much more nonzeros than A
- there are lots of preconditioning techniques, often specialized for specific applications and matrices

Incomplete LU (ILU) factorization

ILU is a popular choice of preconditioner with interesting algorithmic characteristics

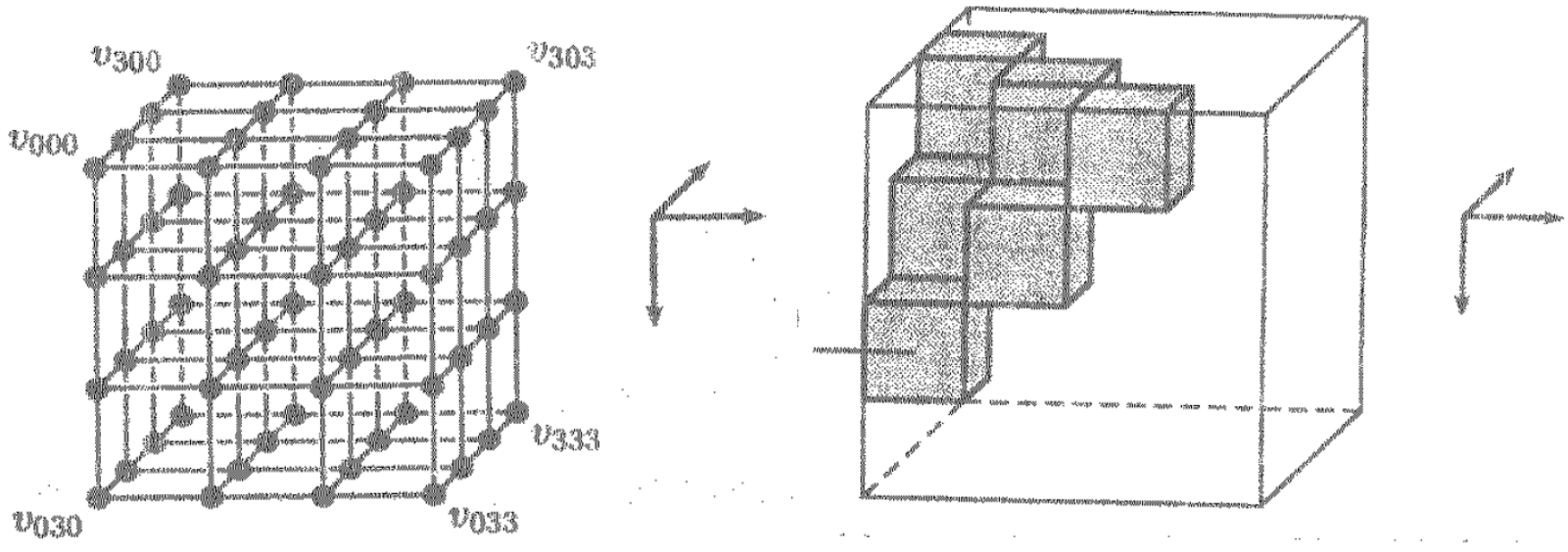
- define $S \in \mathbb{N}^2$ to be a sparsity mask and compute L, U on S
- for instance **ILU[0]**: $(i,j) \in S$ iff $A_{ij} \neq 0$
- Gaussian elimination on A, compute L_{ij} and U_{ij} only if $(i,j) \in S$
- given $[L^{(0)}, U^{(0)}] \leftarrow \text{ILU}[0](A)$, our preconditioner will be $M = L^{(0)} U^{(0)} \approx A$
- to multiply a vector by M^{-1}, we need two sparse triangular solves
- popular variant ILU[1], $(i,j) \in S$ if $\exists k, L^{(0)}_{ik} U^{(0)}_{kj} \neq 0$ (sparser than plain LU, since we do not get additional fill from multiplying fill)
- ILU can *break down* (divide by zero) even when normal LU would not
- lets first study the cost of ILU[0], then consider ILU[l]

Row/column ordering in ILU[0]

Like in sparse Cholesky/LU, ordering of rows/columns affects ILU[0]

- fill will not change for ILU[0], but quality of solution does
- affects dependency structure and parallelism
- for instance, consider 3D level sets $V_i = \{(x, y, z) : x + y + z = i\}$
- we can compute in the order $V_0, V_1, V_2, \ldots$, propagating information at each step
- or we can compute odds $V_0, V_2, V_4, \ldots$ at the same time and evens $V_1, V_3, V_5, \ldots$ at the same time
- latter approach has more parallelism, former may produce a much better preconditioner
- ordering of rows chosen also restricts reorderings for subsequent iterative method
- Q: could we eliminate all odd and then all even level sets independently in sparse Cholesky?
- A: no, eliminating odd level sets would connect the even level sets

Natural level set ordering: cube DAG



Sources: Tiskin 2002, "Bulk Synchronous Parallel Gaussian Elimination", Google image search for "cube DAG", boingboing.net

Cost analysis of cube DAG computation

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

- each vertex depends on neighbors with lower coordinates
- subdivide into blocks of size $n^{1/3}/\sqrt{P} \times n^{1/3}/\sqrt{P} \times n^{1/3}/\sqrt{P}$
- wavefront of depth $O(\sqrt{P})$ with $O(P)$ blocks in each level set
- cost of each wavefront is

$$O(n/P^{3/2} \cdot \gamma + n^{2/3}/P \cdot \beta + \alpha)$$

- therefore the total cost is

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

- Q: is this more or less than than LU of $n^{1/3} \times n^{1/3}$ dense matrix?
- A: the computation cost is the same, but more communication and synchronization required (above equal to 2D LU cost and is optimal for cube DAG, 3D LU cost is less)

Advanced ILU schemes

Lets now consider **ILU[l]**

- let $\mathcal{S}(A)$ be the sparsity mask for matrix A
- $[L^{(l)}, U^{(l)}] =$ILU[l](A) uses sparsity mark S_l where

$$S_l = \mathcal{S}\left(L^{(l-j)}U^{(j-1)} + L^{(j-1)}U^{(l-j)}\right)$$

  which is the same for any $j \in [1, l]$
- different interpretation of ILU[l]
  - label each entry of L_{ij}, U_{ij} with ζ_{ij}
  - if $(i,j) \in \mathcal{S}(A)$, $\zeta_{ij} = 0$
  - when a Schur complement update would yield a new entry of fill $L_{ik} \cdot U_{kj}$ let $\zeta_{ij} = \zeta_{ik} + \zeta_{kj} + 1$
  - create entry only if $\zeta_{ij} \leq l$

Advanced ILU[l] in terms of paths

Let A be the adjacency matrix of an unweighted direct graph G

- we refer to **length** of a path as the unweighted distance (#edges)
- let $D[l](G)$ be the matrix of shortest distances in G that are of length less than or equal to m
- the ILU[l] mask for A is $S_l = \mathcal{S}(D[l](G))$
- **fill path**: a new edge will be added between two vertices in G if there exists a path
  - of any length for complete sparse Cholesky/LU
  - of length l for ILU[l]
- we start with an undirected graph corresponding to our mesh
  - forming the adjacency matrix puts an ordering on the vertices
  - this ordering turns the mesh into a DAG
  - ILU[l] adds new edges for each path of length l in the DAG

Cost analysis of computing ILU[l]

Generally, ILU[l] has a space complexity overhead proportional to the number of paths of length l in G

- on a d-dimensional mesh ordered lexicographically, this is $\Theta(l^d)$, for a total amount of space of $O(nl^d)$
- Q: how many times would we update each entry of the Schur complement in ILU[l] for this mesh?
- A: $O(l^d)$, since this is the number of vertices (rows/cols) on path of length less than l between a pair of vertices
- the total computation cost is therefore $F = O(nl^{2d})$
- note that we never have a path of length greater than $l = O(n^{1/3})$ in the cube DAG
- so long as $l \leq n^{1/3}/\sqrt{P}$, $d = 3$ we can use a wavefront like ILU[0]

$$T_{\text{ILU[l]-cube}}(n, P) = O(l^6 n/P \cdot \gamma + l^3 n^{2/3}/\sqrt{P} \cdot \beta + \sqrt{P} \cdot \alpha)$$

- for $l > n^{1/3}/\sqrt{P}$, we might want to have layers of processors doing updates ahead of the wavefront (3D parallelization)

Cost analysis of applying ILU[l]

Once we start running our preconditioned method, we need to apply the ILU[l] decomposition

- each time it is a triangular solve
- the amount of work in a triangular solve is as much as SpMV, so $O(nl^d)$
- however, the operations are interdependent and not as parallelizable
- we can use a wavefront approach for $d = 3$, as when computing ILU

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

- so long as $l \leq n^{1/3}/\sqrt{P}$, since the ghost-zone we need for each block of each wavefront has dimensions roughly $l \times n^{1/3}/\sqrt{P} \times n^{1/3}/\sqrt{P}$

Short pause

Threshold-based sparsification by magnitude

The sparsity mask in ILU[l] may not be the best one

- we might do better by keeping L and U entries that are relatively large in magnitude
- for instance, drop (set to zero) any entry $|L_{ij}|, |U_{ij}| < \tau$
- however, in a right-looking LU algorithm with Schur complement updates, we would need to use extra memory to form potential nonzeros before we know to drop them
- Q: how can we do LU sequentially with the above drop criterion using no extra memory?
- A: use a left-looking algorithm, performing all updates to a given entry all at once and decide whether to drop immediately
- each such update is an SpMSpV (sparse matrix times sparse vector)
- more parallelism and less communication at the cost of a bit more memory may be obtained by updating a few columns via SpMSpM (sparse matrix times sparse matrix)

ILU preconditioning techniques

Applying ILU preconditioner $M = LU$,

$$M^{-1}v = U^{-1}L^{-1}v$$

requires triangular solves

- need to apply at every iteration of iterative scheme
- each can be almost as expensive as computing ILU
- basic idea behind *domain decomposition*: precondition within subdomains

$$M_{\text{DD}}^{-1} = \begin{bmatrix} U_1^{-1}L_1^{-1} & 0 & \cdots \\ 0 & \ddots & 0 \\ \vdots & 0 & U_P^{-1}L_P^{-1} \end{bmatrix}$$

- given subdivision into P domains, ILU and triangular solves can be completed in parallel with no synchronization or communication cost
- Jacobi-preconditioning: overlap diagonal blocks to achieve better preconditioner

Basic domain decomposition

A block-diagonal preconditioner is not generally robust

- its important to consider boundaries between partitions
- we can classify each partition as having interior and boundary nodes
- with a careful ordering we e can then solve for the interior nodes separately from the boundary nodes
- we still need a global ordering among processors
- solving for the boundary nodes needs to be done in order for ILU computation and application
- by defining different global orderings, for instance via graph coloring, its possible to allow subdomains to operate simultaneously, but generally with some loss of quality in the preconditioning