CS 598: Communication Cost Analysis of Algorithms

Lecture 17: Sparse linear systems: communication-avoiding algorithms

Edgar Solomonik

University of Illinois at Urbana-Champaign

October 19, 2016

PDE discretization

The primary source of sparse matrix problems in computational science are partial differential equations

- we restrict ourself only to considerations necessary for communication complexity analysis of iterative schemes
- in particular, we care about the structure of the sparse matrices associated with different discretizations
- numerical methods for PDEs seek a mesh-based representation solution to the PDE over a given domain
  - **structured grids** define a mesh with a regular (uniform) connectivity pattern, which be inferred *implicitly*
  - **unstructured grids** define a mesh with an irregular connectivity pattern that needs to be stored *explicitly*

Basic approximation (finite differences)

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

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

Sparsity of multidimensional discretization

Recall the sparse matrix given by 1D centered differences

$$D = \begin{pmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & & \end{pmatrix}$$

- Q: what sparse matrix would a centered difference approximation yield for a $n \times n$ uniform grid?
- A: assuming a natural ordering of elements

$$A = \begin{pmatrix} F & I_n & & \\ I_n & \ddots & \ddots & \\ & \ddots & & \end{pmatrix}$$

where I_n is the identity matrix with dimension n and $F = D - 2I_n$

- Q: what does the first subdiagonal of A look like?
- A: written in row-vector form: $[\mathbf{1}_{n-1}\ 0\ \mathbf{1}_{n-1}\ 0\ \cdots\ \mathbf{1}_{n-1}]$ where $\mathbf{1}_{n-1} = [1\ \cdots\ 1]$ is a row vector of dimension $n-1$

PDE discretization methods

Lets consider characteristics of the two most basic types of discretizations

- finite difference methods
  - for derivative approximations on uniform grids, yield structured (nearly Toeplitz) matrices
  - simple and attractive for regular grids due to potential for fast methods
  - when applied on irregular grids or for generalized differential operators, may need to work with sparse matrix representation
- finite element methods (FEM)
  - define n localized basis functions over n mesh-points
  - entries of matrix given by pairwise integrals of basis functions over the whole space
  - matrix is sparse because most pairs of functions have disjoint **support** (one or the other is zero at every point)
  - can yield structured or unstructured matrices
  - the matrix **assembly** can happen statically or dynamically (on the fly)
  - well-understood and general, extensible to high-order methods

Sparse linear systems of equations

After a PDE discretization and also in other types of applications, we are left with the ubiquitous matrix equation

$$Ax = b$$

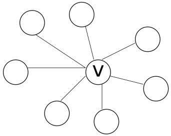
where A is square and sparse

- A may be structured and/or may have an implicit representation
- solutions can be found by **direct** or **iterative** methods
- direct methods compute $x = A^{-1}b$ by an approximation to A^{-1}
  - A^{-1} may be dense or may not exist
  - can try to preserve sparsity in factorization of A (e.g. by LU)
  - can also obtain an approximate solution by an inexact factorization, e.g. incomplete LU: $A \approx LU$ where L, U have the same sparsity as A
  - an inexact factorization may be useful as a **preconditioner** $Ax = b \quad \rightarrow \quad U^{-1}L^{-1}Ax = U^{-1}L^{-1}b$
- iterative methods solve $Ax = b$ by improving an approximation for x
  - by evolving a guess for x rather than trying to determine A^{-1}, we can use less memory and possibly do less computation

Sparse Cholesky

Lets consider Cholesky $A = LL^T$ on a sparse symmetric matrix A

- as an illustrative example, lets consider the following graph



  with adjacency matrix A
- we would like to perform sparse Cholesky of $I + A$
- Q: if v is the first row/column in A, how many nonzeros will there be after we do the first Cholesky update?
- A: we will get a dense matrix, since $I + A = \begin{pmatrix} x_1 & x_2 & \cdots \\ x_2 & 1 & 0 \\ \vdots & 0 & \ddots \end{pmatrix}$ so, the first row/column of L will be dense, and the update will be a dense vector outer product $L(:,1) \cdot L(1,:)$

Sparse Cholesky

Lets consider Cholesky $A = LL^T$ on a sparse symmetric matrix A

- Q: now, how many nonzeros will the update introduce if v is the last row/column of A? That means we have $I + A = \begin{pmatrix} 1 & 0 & y_1 \\ 0 & \ddots & y_2 \\ y_1 & y_2 & \ddots \end{pmatrix}$
- A: none, in fact the final L will have the same sparsity as the lower-triangular part of A, the update is an inner product
- the whole algorithm will require $O(n)$ computation rather than $O(n^3)$
- takeaway idea: the ordering of the rows and columns in the sparse matrix is quintessential for minimizing **fill-in** during sparse matrix factorization

Nested dissection intuition

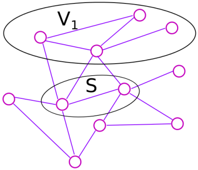
In the toy example, we note that v was a vertex separator

- the general idea of **nested dissection** is to find vertex separators recursively and put them at the end of the *elimination ordering*
- a step of Cholesky/LU, graphically corresponds to eliminating a vertex and connecting all of its neighbors together
- if we first eliminate vertices local to different graph partitions, we can perform them independently
- on the other hand, if we eliminate the separator between the graph partitions, we will potentially create many new edges (fill-in) between the two partitions

Nested dissection

The nested dissection algorithm works on a sparse matrix A as follows

- let G be the graph with adjacency matrix A
- find a small balanced vertex separator S in G



- reorder the rows/columns of A as $[V_1, V_2, S]$, obtaining

$$A = \begin{pmatrix} A_1 & 0 & A_{1S} \\ 0 & A_2 & A_{2S} \\ A_{S1} & A_{S2} & A_S \end{pmatrix}$$

- factorize $A_1 = L_1 L_1^T$ and $A_2 = L_2 L_2^T$ recursively (in parallel)
- compute $L_{S1} = L_{1S}^T = A_{S1} L_1^{-T}$, $L_{S2} = L_{2S}^T = A_{S2} L_2^{-T}$
- factorize $A_S - L_{S1} \cdot L_{1S}^T = L_S L_S^T$ by dense Cholesky

Nested dissection analysis

Lets now consider the cost of nested dissection

- finding a good balanced vertex separator for a general graph is hard and may not even be possible
  - if the sparse matrix comes from a PDE discretization, we can subdivide the physical domain
  - for a uniform (regular) grid, we should slice the longest dimension
  - Q: what is the size of a minimal balanced separator for a 2D grid? 3D?
  - A: 2D $|S| = O(\sqrt{n})$, 3D $|S| = O(n^{2/3})$, dD $|S| = O(n^{(d-1)/d})$
- with additional assumptions on the partitioning, it can be shown that the triangular solves like $A_{S1}L_1^{-T}$ and the update $A_S - L_{S1} \cdot L_{1S}^T$ have costs that do not asymptotically exceed the final Cholesky on an $|S| \times |S|$ matrix
- given this, we have the recurrence

$$T_{\text{sp-Chol}}(n, d, P) = T_{\text{sp-Chol}}(n/2, d, P/2) + O(T_{\text{Chol}}(n^{(d-1)/d}, P))$$

Nested dissection cost analysis

Lets expand the cost recurrence of Cholesky with nested dissection

$$T_{\text{sp-Chol}}(n, d, P) = T_{\text{sp-Chol}}(n/2, d, P/2) + O(T_{\text{Chol}}(n^{(d-1)/d}, P))$$

- $T_{\text{Chol}} \leq T_{\text{LU}}$, so we can recall the cost for dense Cholesky

$$T_{\text{Chol}}(n^{(d-1)/d}, P) = O\left(\frac{n^{3(d-1)/d}}{P} \cdot \gamma + \frac{n^{2(d-1)/d}}{\sqrt{cP}} \cdot \beta + \sqrt{cP} \cdot \alpha\right)$$

- Q: does the flop cost decrease geometrically in nested dissection?
- A: need $2^{3(d-1)/d} > 2$, so $3(d-1)/d > 1$ and $(d-1)/d > 1/3$, which is always true
- Q: does the communication cost also always decrease geometrically?
- A: yes, $2^{2(d-1)/d} > \sqrt{2}$, as $2(d-1)/d > 1/2$ since $(d-1)/d > 1/4$
- Q: lastly, how about the synchronization cost (α term)?
- A: also decreases geometrically, since P does, so we obtain

$$T_{\text{sp-Chol}}(n, d, P) = O(T_{\text{Chol}}(n^{(d-1)/d}, P))$$

Short pause

Jacobi iteration

Jacobi iteration is a basic sparse iterative method for solving $Ax = b$

- start with an initial guess x_0
- compute $x_{i+1} = D^{-1}(b - (A - D)x_i)$ where D is the diagonal of A, so

$$x_{i+1}(j) = \frac{1}{A(j,j)}\left(b(j) - \sum_{k \neq j} A(j,k)x_i(k)\right)$$

- the expensive part is the SpMV $(A - D)x_i$
- slight variations can improve convergence by rescaling some terms
- if A is a stencil, Jacobi iteration is just a simultaneous stencil application to all nodes in the mesh

Gauss-Seidel iteration

Gauss-Seidel tries to improve convergence of Jacobi iteration by using applications of the stencil in one part of the mesh as inputs to the next

- this is the same as computing a sparse subset of x_{i+1} at every iteration, and taking the rest to be elements of x_i
- naive Gauss-Seidel computes one element at a time and has almost no parallelism
- Gauss-Seidel can be seen as Bellman-Ford where edge relaxations are done in some order and use the latest values
- in the worst case it has as little parallelism as Dijkstra's algorithm
- Gauss-Seidel with red-black ordering tries to find an 'ordering' that has parallelism, in particular a 2-coloring of the graph (partition vertices in two sets such that each set has no internal edges), and updates 1 color at a time

Krylov subspace methods

An m-dimensional **Krylov subspace** for matrix A with starting vector v is

$$\mathcal{K}_m(A, v) = \text{span}\{v, Av, A^2 v \ldots A^{m-1} v\}$$

- from $\mathcal{K}_m(A, b - Ax_0)$ where x_0 is an initial guess, we can extract a better approximation for $Ax - b$
- Krylov methods for linear systems include Arnoldi, CG, GMRES, Lanczos and many variants of these
- other choices of v allow computation of eigenvectors of A ($A^k x$ converges to the eigenvector of A with the largest eigenvalue)
- from a bird's eye view, these methods are dominated in cost by SpMV
- except **block Krylov methods**, which compute AV for tall-skinny V
  - the main motivation for these is increasing (communication) efficiency
- we can also distinguish between orthogonalized and non-orthogonalized methods
  - orhtogonalizing each iterate with the previous can improve convergence
  - orthogonalization can be done by tall-skinny QR, but implies a strict dependence between SpMV iterations

Cost analysis of iterative methods

We already studied the cost of SpMV earlier in the course

- given a matrix with m nonzeros, randomization and 2D blocking gives

$$T_{\text{SpMV}}(n, m, P) = O\left(\frac{m}{P} \cdot \nu + \frac{n}{\sqrt{P}} \cdot \beta + \log(P) \cdot \alpha\right)$$

  where we assume $\nu > \gamma$ and point-to-point messages for latency cost
- if we have a low-order stencil ($m = O(n)$) on a uniform d-dimensional grid, it makes sense to partition vertices (matrix rows)
- we can pick out subvolumes of n/P vertices, which are connected to $O((n/P)^{(d-1)/d})$ external vertices
- this partitioning can lower interprocessor communication cost

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

- Q: how much lower is the interprocessor bandwidth cost for $d = 2$? 3?
- A: a factor of $\Theta(\sqrt{n})$ in 2D and $\Theta(n^{1/3}P^{1/6})$ in 3D

Cost comparison of sparse linear solvers

Lets compare the cost of iterative methods with that of sparse Cholesky

$$T_{\text{sp-Chol}}(n,d,P) = O\left(\frac{n^{3(d-1)/d}}{P}\cdot\gamma + \frac{n^{2(d-1)/d}}{\sqrt{cP}}\cdot\beta + \sqrt{cP}\cdot\alpha\right)$$

the memory-bandwidth cost would be $O\left(\frac{n^{3(d-1)/d}}{P\sqrt{H}}\cdot\nu\right)$

- let s be the number of iterations our method takes to converge
- the total cost of a sparse iterative method with s iterations is

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

- one can argue for the expectation, $s = \Theta(n^{1/d})$

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

- direct methods better for $d=2$ and worse for $d=3$? (apples and oranges are both spherical)

Improving the cost of sparse iterative solvers

We have observed that sparse iterative methods entail a few communication bottlenecks

- the flop-to-byte ratio is $O(1)$ (excepting block Krylov methods and the case of each processor fitting the whole sub-problem in cache)
- the synchronization (latency) cost scales with the number of iterations
- the interprocessor communication cost is non-trivial, but smaller than the memory-bandwidth cost by a factor of $\Theta((n/P)^{1/d})$
- to do better, we need to find ways to execute many SpMVs faster than performing each one at a time