

AA220 / CS 238
Parallel Methods in Numerical Analysis

Matrix Computation: Iterative Methods II

Outline:

- CG & its parallelization.
- Sparse Matrix-vector Multiplication.

Basic iterative methods:

$$Ax = b$$

$$r = b - Ax \text{ (residual)}$$

- Split the matrix $A = M - N$
- Solve $Mx^{k+1} = Nx^k + b$
 $x^{k+1} = x^k + M^{-1}(b - Ax) = x^k + M^{-1}r$
- Jacobi: $M = D, N = L + U$
- GS: $M = D + L, N = U$

Summary: Jacobi vs GS

- Jacobi is parallel.
- GS is sequential because of the data dependency. Can be fixed by coloring technique.
- Comparison:

	Jacobi	GS
Parallelism	Good	Poor
Convergence	Slow	Fast

Which one is faster on parallel machines?

Jacobi vs RB Gauss-Seidel

- RB GS converges twice as fast as Jacobi, but requires twice as many parallel steps; about the same run time in practice.
- Parallel efficiency **alone** is not sufficient to determine overall performance.
- We also need fast converging algorithms.

Conjugate Gradient Method

- For $A=SPD$, i.e.
 - ▷ symmetric: $A = A^T$.
 - ▷ positive definite: $x^T Ax > 0, \quad \forall x \neq 0$.

- Consider the quadratic function:

$$\Phi(x) = \frac{1}{2}x^T Ax - x^T b$$

- ▷ Its unique minimum x satisfies:

$$Ax = b$$

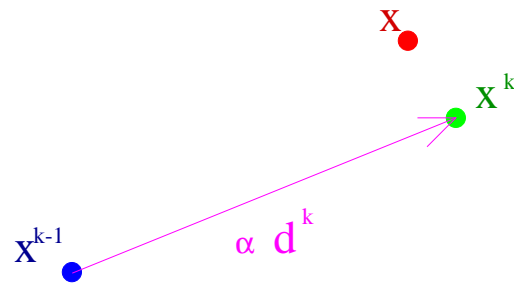
- ▷ Hence, solving $Ax = b \Leftrightarrow \min \Phi(x)$.

- Given x^{k-1} and search direction d^k , let

$$x^k = x^{k-1} + \alpha d^k,$$

where $\alpha =$ step length.

CG (cont.)



- Determine α by min. $\Phi(x^k)$ along d^k :

$$\min_{\alpha} \Phi(x^{k-1} + \alpha d^k).$$

- By differentiation, $\alpha_k = \alpha_{\text{opt}}$ is given by

$$\alpha_k = \frac{(r^{k-1})^T d^k}{(d^k)^T A d^k}$$

where $r^{k-1} = b - Ax^{k-1} =$ residual vector.

- The search direction d^k is chosen such that it is A -orthogonal to all previous directions, i.e.

$$(d^k)^T A d^j = 0, \quad j = 0, \dots, k-1.$$

Using this fact, it can be shown that:

$$\alpha_k = \frac{(r^{k-1})^T r^{k-1}}{(d^k)^T A d^k}$$

CG Algorithm

```
 $k = 0; r^0 = b - Ax^0; \rho_0 = \|r^0\|_2^2;$   
while ( $\sqrt{\rho_k} > \epsilon \|r^0\|_2$ ) do  
   $k = k + 1;$   
   $\rho_{k-1} = (r^{k-1})^T r^{k-1};$   
  if ( $k = 1$ )  
     $d^1 = r^0;$   
  else  
     $\beta_{k-1} = \rho_{k-1} / \rho_{k-2};$   
     $d^k = r^{k-1} + \beta_{k-1} d^{k-1};$   
  endif  
   $e^k = Ad^k;$   
   $\alpha_k = \rho_{k-1} / (d^k)^T e^k;$   
   $x^k = x^{k-1} + \alpha_k d^k;$   
   $r^k = r^{k-1} - \alpha_k e^k;$   
   $\rho_k = (r^k)^T r^k;$   
end
```

Properties of CG

- No need to form A . Only Av is needed.
- Minimal storage:
 - ▷ Short recurrence for the update of x^k .
 - ▷ Need only store: x^k, r^k, d^k, Ad^k .
- Minimal error:
 - ▷ Define $\mathcal{K}_k(A; r^0)$, *Krylov subspace* of dimension k :

$$\mathcal{K}_k(A; r^0) \equiv \{r^0, Ar^0, \dots, A^{k-1}r^0\}.$$

It can be proved that:

$$\begin{aligned}\mathcal{K}_k(A; r^0) &= \text{span}\{r^0, r^1, \dots, r^{k-1}\} \\ &= \text{span}\{d^1, d^2, \dots, d^k\}.\end{aligned}$$

- ▷ $x^k \in \mathcal{K}_k(A; r^0)$ min. the error $e^k \equiv x^k - x$ in the A -norm:

$$\|x^k - x\|_A \leq \|\tilde{x} - x\|_A \quad \forall \tilde{x} \in \mathcal{K}_k(A; r^0),$$

where $\|w\|_A = \sqrt{w^T A w}$.

Properties of CG (cont.)

- In exact arithmetic, CG achieves exact solution in at most n steps (finite termination).
- In practice, an accurate approx. is obtained long before n steps. So, CG is usually used as an iterative method.
- Convergence of CG:
 - ▷ Upper bound:

$$\|x - x^k\|_A \leq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \|x - x^0\|_A,$$

where κ is the condition number of A :

$$\kappa \equiv \|A\| \|A^{-1}\| = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}.$$

- ▷ In practice, the rate of convergence is usually **much faster than the upper bound**, especially in the case when the eigenvalues of A are located in **clusters**.
- 2D mesh with N unknowns, Poisson eqn: CG converges in $O(\sqrt{N})$ steps, same as optimal SOR.

Parallel CG

$p = \#$ of processors.

- **DAXPY**: scalar \times vector $+$ vector (BLAS I)
 - ▷ 3 saxpy operations: d^k, x^k, r^k .
 - ▷ $O(n/p)$ flops.
 - ▷ No communication.
- **Inner products** (BLAS I)
 - ▷ 2 inner products: $(r^{k-1})^T r^{k-1}, (e^k)^T d^k$.
 - ▷ $O(n/p)$ flops.
 - ▷ Communication for the reduction process (collecting partial sums from all processors).
- **Matrix-vector product** (Essentially BLAS I)
 - ▷ Major cost of CG.
 - ▷ Efficiency depends on sparse structure of A .
- Minor note: $\{x^k\}$ are not needed in the CG process. We may delay the update of x^k, \dots, x^{k+j} by storing d^k, \dots, d^{k+j} .

Overlapping Communication

- Want to overlap communication time with useful computations.
- In standard CG, there are 2 synchronization points (inner products).
- Possible to reduce to 1 synch. point by rearranging terms.
- A version is given in next slide.
- 1 additional saxpy operation.
- 2 inner products can be computed in parallel → 1 synch. point.
- Main disadvantage is the possible *numerical instability*.

Variant CG Algorithm

$$r^0 = b - Ax^0;$$

$$q^{-1} = p^{-1} = 0; \beta_{-1} = 0;$$

$$s^0 = Ar^0;$$

$$\rho_0 = (r^0)^T r^0; \mu_0 = (s^0)^T r^0; \alpha_0 = \rho_0 / \mu_0;$$

for $k = 0, 1, \dots$

$$p^k = r^k + \beta_{k-1} p^{k-1};$$

$$q^k = s^k + \beta_{k-1} q^{k-1};$$

$$x^{k+1} = x^k + \alpha_k p^k;$$

$$r^{k+1} = r^k - \alpha_k q^k;$$

check convergence; continue if necessary

$$s^{k+1} = Ar^{k+1};$$

$$\rho_{k+1} = (r^{k+1})^T r^{k+1};$$

$$\mu_{k+1} = (s^{k+1})^T r^{k+1};$$

$$\beta_k = \rho_{k+1} / \rho_k;$$

$$\alpha_{k+1} = \rho_{k+1} / (\mu_{k+1} - \rho_{k+1} \beta_k / \alpha_k);$$

end

CG for non symmetric matrices

- $Ax = b$, where A is not SPD
- $A^T Ax = A^T b$ (minimal residual)
- $x = A^T y$
solve $AA^T y = b$ (minimal error)
get x from y
- When A is square, the condition number of $A^T A$ is bigger than the condition number of A : slower convergence

Krylov Subspace Methods

- A can be any nonsymmetric matrices.
- Let $V_k = \{v_1, \dots, v_k\}$ be a basis for $\mathcal{K}_k(A; r^0)$.
Look for approx. solution $x^k \in \mathcal{K}_k(A; r^0)$, i.e.,

$$x^k = V_k y,$$

for some $y \in \mathbb{R}^k$.

- Four projection-type approaches to pick x^k :
 - ▷ **Ritz-Galerkin**: (CG, Lanczos, FOM, GENCG)

$$b - Ax^k \perp \mathcal{K}_k(A; r^0).$$

- ▷ **Minimum Residual**: (GMRES, MINRES)

$$\min_{x^k \in \mathcal{K}_k} \|b - Ax^k\|_2.$$

- ▷ **Petrov-Galerkin**: (BiCG, BiCGSTAB, QMR)

$$b - Ax^k \perp \mathcal{S}_k.$$

e.g. $\mathcal{S}_k = \mathcal{K}_k(A^T; s^0)$.

- ▷ **Minimum Error**: (SYMMLQ, GMERR)

$$\min_{x^k \in \mathcal{K}_k} \|x - x^k\|_2.$$

Sparse Matrix Computations

- Computations only carried out at nonzero entries. Thus, only nonzero entries are stored.

Storage formats:

- **Coordinate format:**

$A = \{I, J, VAL\}$, I, J, VAL , are $nnz \times 1$ array;
 $nnz = \#$ of nonzeros.

- **Compress sparse row format:**

$A = \{I, J, VAL\}$

$VAL = nnz \times 1$ array of nonzero entries

$J = nnz \times 1$ array of column indices.

$I = n \times 1$ array; i th entries of which points to the 1st entry of the i th row in VAL & J .

- **Ellpack-Itpack format:**

$m = \max.$ $\#$ of nonzeros in any row.

$A = \{VAL, J\}$

$VAL = n \times m$ array of nonzero entries.

$J = n \times m$ array of column indices.

Example: storage formats

$$A = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 3 & 4 & 0 & 0 \\ 0 & 5 & 6 & 0 \\ 7 & 0 & 8 & 9 \end{bmatrix}$$

- Coordinate format:

$$\begin{aligned} I &= [1 \ 1 \ 2 \ 2 \ 3 \ 3 \ 4 \ 4 \ 4] \\ J &= [1 \ 4 \ 1 \ 2 \ 2 \ 3 \ 1 \ 3 \ 4] \\ VAL &= [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9] \end{aligned}$$

- CSR format:

$$\begin{aligned} VAL &= [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9] \\ J &= [1 \ 4 \ 1 \ 2 \ 2 \ 3 \ 1 \ 3 \ 4] \\ I &= [1 \ 3 \ 5 \ 7] \end{aligned}$$

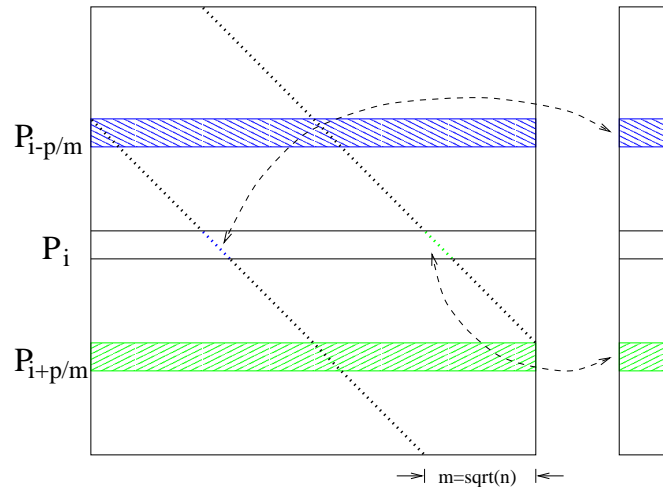
- Ellpack-Itpack format:

$$VAL = \begin{bmatrix} 1 & 2 & - \\ 3 & 4 & - \\ 5 & 6 & - \\ 7 & 8 & 9 \end{bmatrix}, \quad J = \begin{bmatrix} 1 & 4 & -1 \\ 1 & 2 & -1 \\ 2 & 3 & -1 \\ 1 & 3 & 4 \end{bmatrix}$$

Sparse Matrix-Vector Multiplication

- For iterative methods such as Jacobi, Gauss-Seidel, CG, etc, the major computational cost per iteration is Av .
- Sparse matrices usually have a special structure which should be exploited for max efficiency.
- Consider 3 cases:
 - ▷ Matrices whose nonzero entries lie on a few diagonals.
 - ▷ Unstructured matrices; the locations of the nonzero entries can be anywhere.
 - ▷ Banded matrices; the nonzero elements are confined within a band.
- **Key Idea**: data mapping.

Pentadiagonal Matrices (cont.)



- III: $p \leq \sqrt{n}$:
Comm between neighboring processors exchanging \sqrt{n} elements.

Timing: $T_{\text{comm}} = 2(t_s + t_w \sqrt{n})$.

(including T_{comm} for II)

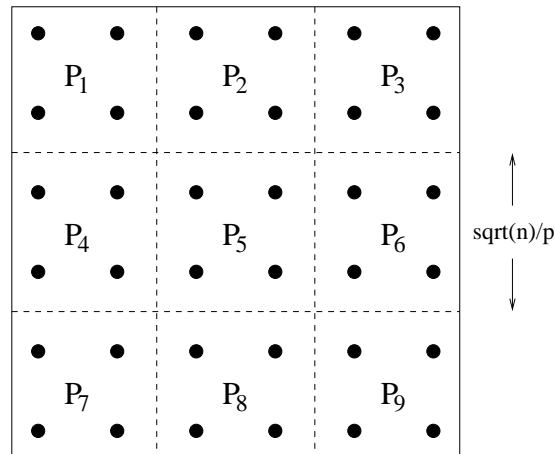
- III: $p > \sqrt{n}$:
Need comm with $P_{i \pm p/\sqrt{n}}$ for n/p elements.

Timing:

$$T_{\text{comm}} = 2(t_s + t_w n/p + t_h \log p),$$

where t_h = time per hop between processors.

Checkerboard (domain) Partitioning



- Each processor stores $\sqrt{n/p}$ clusters of $\sqrt{n/p}$ matrix rows, i.e. n/p elements.
- Need comm only along boundary data with length $= 4\sqrt{n/p}$.

Timing:

$$T_{\text{comm}} = 4(t_s + t_w\sqrt{n/p}).$$

Unstructured Sparse Matrices

- Partition by rows.
- Require the entire vector in general. Hence all-to-all broadcast.
- Perform local matrix-vector multiplication.
- Parallel run time (hypercube):

$$\begin{aligned} T_p &= t_c m \frac{n}{p} + t_s \log p + t_w n \\ &= O(n). \end{aligned}$$

t_c =computational time per data,
 m =max number of nonzero per row.
(Sequential run time = $O(n)$)

- Hence, no gain in parallel.

Checkerboard Matrix Partitioning

- Hypercube:

$$T_{\text{comm}} = t_s \log p + \frac{3(t_w n \log p)}{2\sqrt{p}}.$$

- Average computation = $\frac{mn}{p}$,
 m = average number of nonzeros per row.
- Parallel run time, speedup, efficiency:

$$T_p = t_c \frac{mn}{p} + t_s \log p + \frac{3}{2} t_w \frac{n \log p}{\sqrt{p}}$$

$$S = \frac{mt_c pn}{mt_c n + t_s p \log p + (3t_w n \sqrt{p}/2) \log p}$$

$$E = \frac{T_1}{p T_p}$$

$$= \frac{mt_c}{mt_c + (t_s p \log p)/n + (3\sqrt{p} \log p)/2}$$

- $E \not\rightarrow 1$ as $n \rightarrow \infty \Rightarrow$ unscalable.
- $T_p = O(\log p) + O(\frac{n \log p}{\sqrt{p}})$
 better than sequential run time for large p .

Planar Graph

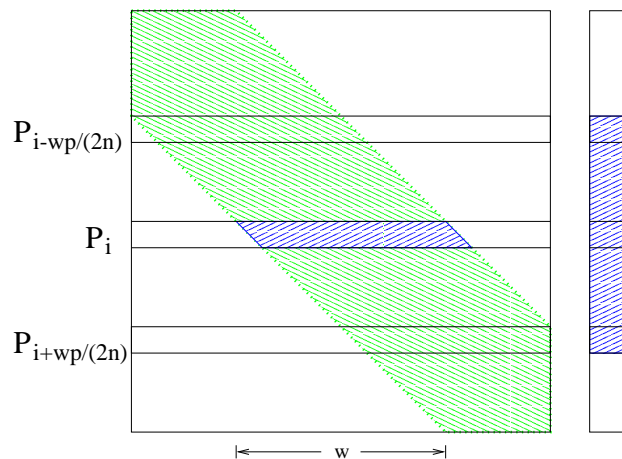
- Suppose A sparse matrix with a symmetric structure, i.e.

$$a_{i,j} \neq 0 \text{ iff } a_{j,i} \neq 0.$$

- Let $G(A)$ be the graph of A . A scalable parallel implementation of matrix-vector multiply exists if $G(A)$ is planar.
- If $G(A)$ is planar, communication occurs only at the boundaries of each processor's partition.
- Hence, by reducing the number of partitions, or equivalently, increasing the size of the partitions, it is possible to increase the computation to comm. ratio of the processors.
- Partition a graph to minimize interprocessor comm. is NP-hard.

Banded Unstructured Sparse Matrices

- Suppose bandwidth = w , Ellpack-Itpack format, partition the matrix by rows, average # of nonzeros = m .
- Assume $n/p \ll w$:



- Need vector element indices from $i-(w-1)/2$ to $i+(w-1)/2 \Rightarrow$ comm. with $(w-1)p/n$ processors, i.e. P_i comm. with

$$P_{i-wp/(2n)} \cdots P_{i+wp/2n}$$