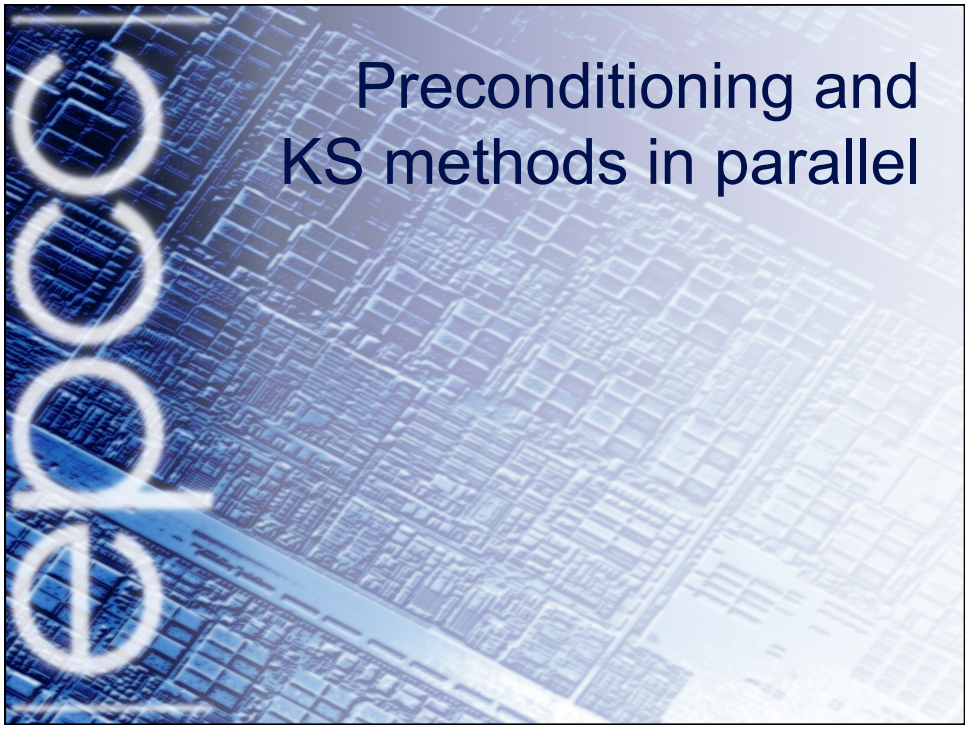


Outline

- Preconditioning
- KS method in parallel

Overview

- Motivation
- What is preconditioning
- What is its purpose
- Common preconditioners

3

What is preconditioning?

"a preconditioner is any form of implicit or explicit modification of an original linear system which makes it "easier" to solve by a given iterative method"

Y Saad, Iterative methods for Sparse Linear Systems

- Examples or preconditioners
  - Scaling all rows so that diagonal entries are equal to 1
  - Pre-mulitplying the matrix by a given matrix, e.g $A \rightarrow M^{-1} A$
    - Unlikely M or $M^{-1} A$ ever computed directly
    - M^{-1} may be complicated – e.g. result of some FFT transformations or integral calculations

4

Motivation

- We would like Krylov subspace method to converge (smoothly)
- We would like KS method to converge in as few iterations as possible
  - reduce the effects of rounding error
  - make method more tractable for multiple righthand sides
- Number of iterations is affected by condition number, *C*, of matrix
  - a lower condition number implies fewer KS iterations

$$C = \frac{\lambda_{\max}}{\lambda_{\min}}$$ Eigenvalue equation $$Ax = \lambda x$$

- Choose pre-conditioner which reduces the condition number

Preconditioning in linear systems

- Solving system

$$A\underline{u} = \underline{b}$$

is equivalent to solving

$$M^{-1}A\underline{u} = M^{-1}\underline{b}$$

where *M* is a SPD (for CG, at least).

- Idea is to choose *M* such that similar to *A* but easier to invert.
- Jacobi/Gauss-Seidel can be thought of as pre-conditioner, e.g.

$$A = L + D + U \rightarrow D^{-1}A = D^{-1}(L + U) + I$$

What is preconditioning? cont ...

- Two extremal cases:
  - choice ***M=I*** is equivalent to no preconditioning
  - choice ***M=A*** is equivalent to factorising the problem directly
- We seek an intermediate ***M*** which ensures that KS method will converge and reduces (minimises) the cost of solver.
- Idea to preserve structure of ***M*** (particularly symmetry!)

Incomplete LU factorisation

- A more powerful/expensive preconditioner is ILU factorisation
- Elements are computed as in LU factorisation, but those that fall outwith the sparsity pattern are discarded

  $A = LU - R$
- This preserves sparsity pattern
- No guarantee of existence of non-singular ILU factors.
- Many modifications exist

Preconditioned CG algorithm

Choose $\underline{v}_0$, compute $\underline{r}_0=\underline{b}-A\underline{v}_0$, **k=0**,
Solve $M\underline{s}_0=\underline{r}_0$ (using a direct method), $\underline{p}_0=\underline{s}_0$
While (**k<maxiter**)

$\alpha = \underline{r}_k.\underline{s}_k \,/\, \underline{p}_k.A\underline{p}_k$
$\underline{v}_{k+1} = \underline{v}_k + \alpha\underline{p}_k$
$\underline{r}_{k+1} = \underline{r}_k - \alpha A\underline{p}_k$
if ($||\underline{r}_{k+1}||_2 \,/\, ||\underline{b}||_2 <$ **tol**) **break**
Solve $M\underline{s}_{k+1}=\underline{r}_{k+1}$
$\beta = \underline{r}_{k+1}.\underline{s}_{k+1} \,/\, \underline{r}_k.\underline{s}_k$
$\underline{p}_{k+1} = \underline{s}_{k+1} + \beta\underline{p}_k$
k=k+1

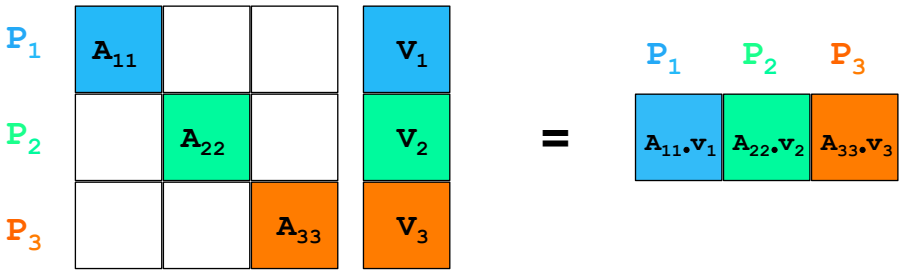
end while

Conclusions

- Preconditioning of linear system can improve reliability of KS method
- Preconditioning can also reduce iteration count and computational costs
- Simple preconditioners based on stationary splitting methods
- More complex methods such as ILU, or preconditioning using Fourier transforms are more effective but may not work at all.

Parallel KS methods

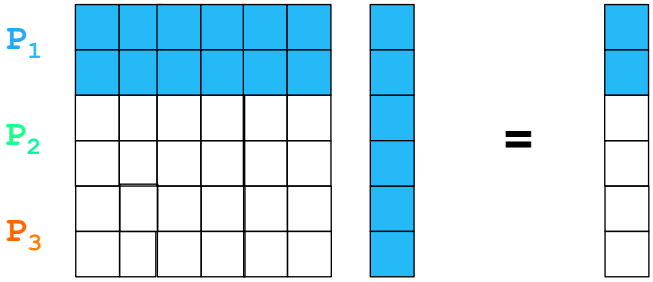
- How to decompose matrix across parallel machine?
  - Depends on how the matrix is stored
  - Depends on what the matrix is
- Block diagonal matrix MV routine is completely parallelisable



11

General parallel matrix-vector

- More general approach is to decompose matrix by row
  - each processor has m rows of matrix
- Consider first processor



- Could simply use replicated data for vector
  - i.e. all processors know about all data
  - Need to broadcast

12

General parallel matrix-vector

- Replicated vector
  - extra memory requirement
  - Each processor has to broadcast result to populate result vector
- Distributed vector
  - efficient use of memory
  - accumulate locally matrix times vector block
  - processors exchange vector blocks
  - Answer ends up distributed
  - Sparse: may not need all processors to exchange blocks
- Smart implementations for specific matrix types
- Global sum required for scalar product

Simple 2 processor 2x2 example

Distributed vector:

$$\begin{matrix} P_1 \\ P_2 \end{matrix} \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} a_{11}b_1 + a_{12}\textcircled{b_2} \\ a_{21}\textcircled{b_1} + a_{22}b_2 \end{pmatrix}$$

- Circled elements initially not available
- Calculate $a_{11}\ b_1$ on P_1 and $a_{22}b_2$ on P_2
- Then swap elements b_1 and b_2 between processors
- Then calculate $a_{12}b_2$ on P1 and $a_{21}b_1$ on P_2

- Generalise to more processors (e.g. pass around a ring rather than simply "swap" elements
- Can also generalise to bigger matrices
- Then deal with blocks instead