EPFL 1988

Iterative methods for multiprocessor vector computers

Gérard Meurant

C E A

Centre d'Etudes de Limeil-Valenton

Outline :

1) Problems to be solved

2) Issues for multiprocessor vector computers

3) Iterative methods

- linear splittings
- conjugate gradient

4) Preconditioners

- incomplete decompositions
- polynomials
- block preconditioners

5) Domain Decomposition

Problems to be solved

We are interested into solving linear systems arising from the discretization of linear partial differential equations with (possibly) strongly discontinuous coefficients like,

$$-\frac{\partial}{\partial x}(a(x,y)\frac{\partial u}{\partial x}) - \frac{\partial}{\partial y}(b(x,y)\frac{\partial u}{\partial y})$$

$$+2\alpha(x,y)\frac{\partial u}{\partial x} + 2\beta(x,y)\frac{\partial u}{\partial y} + \sigma u = f$$

$$\text{in } \Omega \subset R^d, \quad d = 2 \quad \text{or} \quad d = 3$$

$$u\,|_{\partial\Omega} = 0 \quad \text{or} \quad \frac{\partial u}{\partial n}\,|_{\partial\Omega} = 0$$

The problem can be either

- non symmetric
- symmetric ($\alpha = \beta = 0$)

We can use different methods for discretizing

- finite differences (5 point, 9 point,...)
  with various schemes (centered, upwind)

- finite elements (P1, Q1, P2,...)

These methods give (usually) large sparse (depending on the scheme) systems of linear equations.

The structure of the matrix is different depending on

- finite difference : strongly structured mesh, matrix usually stored by diagonals $\longrightarrow$ regular access patterns to data

- finite elements : unstructured mesh, matrix stored with pointers $\longrightarrow$ scattered acesses to data (SCATTER-GATHER).

Other methods could be used like boundary integral methods or spectral methods. These give rise to dense linear systems.

All these issues have a strong influence on the performance of the algorithm; as well as the characteristics of the computer.

2) **Issues for multiprocessor vector computers**

- shared memory machines (ex : CRAY)
- local memory machines (ex : Hypercubes)
- a combination of both (ex : ETA)

multi	**mono**
CRAY Y–MP (8)	CRAY 1–S
CRAY X–MP (4)	CYBER 205
CRAY 2 (4)	FUJITSU VP
ETA 10	NEC SX
Minisupers	HITACHI
⋮	⋮

An important issue for parallel computation is the **granularity** of the computations, that is roughly the number of operations that we are doing between 2 synchronizations of the processors that usually require a call to the operating system.

There should be a balance between the speed of the processor and the time needed for synchronization.

For 2D problems the granularity of efficient iterative methods is too small to get the best possible results (all processor running all the time at full speed and very few synchronizations).

They require very efficient hardware and software synchronisation, especially when the processors are very fast.

Another important issue is how to express the parallelism in the code. No really satisfactory answer has been given yet (multitasking, microtasking, "super" languages, Schedule,...).
There is also a problem of portability.

The main method to introduce parallelism in linear algebra computations is to use special orderings of the unknowns (or grid points) to "decouple" some of the equations.

Unfortunately introducing parallelism usually slows down convergence. So the rule of the game is to find a trading off between parallelism and the speed of convergence.

Iterative methods

Many "old" iterative methods are based on a splitting of the matrix

$$A = M - N$$

with M non singular.
Then we construct a sequence x^k starting from x^0,

$$Mx^{k+1} = Nx^k + b$$

This is equivalent to

$$x^{k+1} = M^{-1}Nx^k + M^{-1}b.$$

If $\varepsilon^k = x^k - x$, then

$$\varepsilon^{k+1} = M^{-1}N\varepsilon^k = \left(M^{-1}N\right)^{k+1}\varepsilon^0$$

It is well known that

$$B^k \longrightarrow 0 \Longleftrightarrow \rho(B) < 1$$

where ρ is the spectral radius $= \max_i |\lambda_i|$.

So the linear method converges to the solution of $Ax = b$ if and only if $\rho(M^{-1}N) < 1$.

If $A = D + L + L^T$, the most well known methods are

- Jacobi, $M = D$

- Gauss–Seidel, $M = D + L$

- Relaxation (S.O.R.), $M = D + \omega L$

Convergence conditions

Definitions and theorems

A strictly diagonally dominant

$$|a_{i,i}| > \sum_{j \neq i} |a_{i,j}|, \ \forall i$$

A generalized strictly diagonally dominant (GSDD)

$$|a_{i,i}| \ d_i > \sum_{j \neq i} |a_{i,j}| \ d_j, \ \forall i, \ d_i > 0$$

A M–matrix $\Longleftrightarrow$ A non singular,

$$a_{i,j} \leq 0, i \neq j, A^{-1} \geq 0.$$

A M–matrix $\Longleftrightarrow$ A GSDD, $a_{i,j} \leq 0, \ i \neq j$

$$M(A)_{i,i} = |a_{i,i}|, \ M(A)_{i,j} = -|a_{i,j}| \ i \neq j$$

A H–matrix $\Longleftrightarrow$ $M(A)$ M–matrix

A H–matrix $\Longrightarrow$ Jacobi converges

A H–matrix $\Longrightarrow$ Gauss–Seidel converges

A H–matrix $\Longrightarrow$ SOR converges, $\forall B \in \Omega(A) = \{B \mid M(B) = M(A)\}$ if

$$0 < \omega < \frac{2}{1 + \rho(|J(B)|)}$$

For SOR, $M^{-1}N$ is not symmetric

SSOR

$$1, 2, \ldots, n$$

$$n, n-1, \ldots, 2, 1$$

$$(D + \omega L)x^{k+1/2} = \omega b + (1 - \omega)Dx^k - \omega U x^k$$

$$(D + \omega U)x^{k+1} = \omega b + (1 - \omega)Dx^{k+1/2} - \omega L x^{k+1/2}$$

The iteration matrix is symmetric

Jacobi is parallel but inefficient, with Gauss–Seidel and relaxation there is a recursion.

We can handle this problem by renumbering the unknowns.

example : Red–Black (or block Red–Black) ordering

Another solution for the 5 point scheme is to compute by diagonals

More generally one can use Multicolor orderings.

Richardson

$$x^{k+1} = x^k + \alpha(b - Ax^k)$$

$$\text{Richardson converges} \iff \alpha < \frac{2}{\lambda_1}$$

$$\alpha_{opt} = \frac{2}{\lambda_1 + \lambda_n}$$

Generalized Richarson

$$Mx^{k+1} = Mx^k + \alpha_k(b - Ax^k)$$

Acceleration of Richardson

$$x^{k+1} = \omega_{k+1}(\alpha_k z^k + x^k - x^{k-1}) + x^{k-1}$$

$$Mz^k = r^k = b - Ax^k$$

Suppose A and M are symmetric positive definite

We choose α_k and ω_k such that

$$(z^i, Mz^j) = 0, \ i \neq j$$

This gives $r^n = 0$

We have to take

$$\alpha_k = \frac{(z^k, Mz^k)}{(z^k, Az^k)}$$

$$\omega_{k+1} = \frac{1}{1 - \frac{\alpha_k (z^k, Mz^k)}{\omega_k \alpha_{k-1} (z^{k-1}, Mz^{k-1})}}$$

Between methods which can be written as

$$x^{k+1} = x^0 + Q_k(M^{-1}A)z^0$$

Q_k polynomial of degree k, it minimizes $E(x^k)$

$$E(x^k) = (A(x - x^k), x - x^k)$$

Consequence

$$E(x^k) \leq 4\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^{2k} E(x^0)$$

where $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$

We can use this method as an iterative algorithm

This the ...

Conjugate Gradient method

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

for $k = 0, 1, \ldots$ until convergence,

$$Mz^k = r^k,$$

$$\beta_k = \frac{(r^k, z^k)}{(r^{k-1}, z^{k-1})},\ \beta_0 = 0,$$

$$p^k = z^k + \beta_k p^{k-1},$$

$$\alpha_k = \frac{(r^k, z^k)}{(Ap^k, p^k)},$$

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

$$r^{k+1} = r^k - \alpha_k A p^k.$$

The problems come from

- the dot products for parallel computation (synchronization)
- Ap if the data accesses are not regular
- $Mz = r$ for both vector and parallel computation

There are (partial) solution to these problems.

Other possibilities are to use Tchebycheff, Richardson or other polynomial methods but usually they require estimations of the extreme eigenvalues.

Basic CG ($M = I$) is well suited for vector computation, as there exist fast implementations of the 3 first operations on all today vector supercomputers. As an example consider the asymptotic Mflops rates for these operations on one processor of a CRAY X–MP/416 and of the ETA 10–E.

operation	CRAY X–MP dedicated	CRAY X–MP loaded
saxpy	185	145
dot product	209	180
matrix vector	169	145

operation	CRAY Y–MP dedicated	ETA 10–E dedicated
saxpy	236	378
dot product	?	187
matrix vector	250	187

For basic CG without preconditioning, the percentages of time spent in the different parts of the computation (not including the initialization phase) on the CRAY X–MP are given for a system of order 36100 in the following table

operation	percentage
saxpy	32
dot product	19.9
matrix vector	47.9
stopping test	0.

From these figures, we can estimate the computational speed we can achieve as we know the percentage of time we are spending in each of the basic operations.

Let n_{op}, t_{op}, fl_{op}, p_{op} respectively the number of operations, the time, the number of floating point operations per second and the percentage of time for one basic operation and t_{tot} the total time; then, as $p_{op} = \frac{t_{op}}{t_{tot}}$, the number of flops for the whole algorithm is

$$fl = \sum_{op} \frac{n_{op}}{t_{tot}} = \sum_{op} fl_{op}\ p_{op}$$

For basic CG on a loaded X–MP system, we can predict a speed of 151 Mflops; the measurements give 145 Mflops, which shows a good agreement. A dedicated system would deliver 181 Mflops (recall that the maximum speed for one processor of the X–MP is 235 Mflops). For a 2500 system, the measured speed is 125 Mflops.

Some of the data dependancies in CG can be removed, noticing that (in exact arithmetic) we have

$$(Mz^i, z^j) = 0, i \neq j$$

Then, it follows

$$(r^{k+1}, z^{k+1}) = \alpha_k^2 (M^{-1} A p^k, A p^k) - (r^k, z^k).$$

The modified algorithm is :

$$x^0 \text{ given}, r^0 = b - A x^0, M z^0 = r^0, p^0 = z^0.$$

For each k until convergence,

$$M v^k = A p^k,$$

$$(v^k, A p^k), \qquad (p^k, A p^k), \qquad (r^k, z^k)$$

$$\alpha_k = \frac{(r^k, z^k)}{(p^k, A p^k)}$$

$$s_{k+1} = \alpha_k^2 (v^k, A p^k) - (r^k, z^k)$$

$$\beta_{k+1} = \frac{s_{k+1}}{(r^k, z^k)}$$

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

$$r^{k+1} = r^k - \alpha_k A p^k$$

$$z^{k+1} = z^k - \alpha_k v^k$$

$$p^{k+1} = (z^k - \alpha_k v^k) + \beta_{k+1} p^k$$

In this algorithm we use a predictor–corrector for the dot product (r^{k+1}, z^{k+1}). First, we predict a value to be able to compute β_{k+1}; then, we correct the value after having computed the new vectors, to improve the stability of the algorithm.

Preconditioners

M symmetric positive definite

M sparse

M easy to construct

$Mz = r$ easy to solve

"good' distribution of the eigenvalues of $M^{-1}A$

Simplest idea

$$M = diag(A)$$

SSOR or block SSOR

$$A = D + L + L^T$$

$$M = \frac{1}{\omega(2-\omega)}(D + \omega L)D^{-1}(D + \omega L^T)$$

Model problem

$$\kappa(A) = O(\frac{1}{h^2})$$

$$\exists\ \omega_{opt} \text{ st } \kappa(M^{-1}A) = O(\frac{1}{h})$$

Pb : ω ?

Incomplete Cholesky decomposition

We have

If we do the Cholesky decomposition of A, we get **fill–in**

To get an **Incomplete Cholesky** decomposition, at each stage of the elimination we drop the fill–ins

This is particularly easy for a 5 diagonal matrix

$$M = L\ D^{-1}\ L^T$$

with

If

$$A = (c_i, b_i, a_i, b_{i+1}, c_{i+m})$$

and

$$L = (\bar{c}_i, \bar{b}_i, d_i, 0, 0),\ \ D = (0, 0, d_i, 0, 0)$$

By inspection

$$\bar{c}_i = c_i$$

$$\bar{b}_i = b_i$$

$$d_i = \frac{1}{a_i - b_{i-1}^2 d_{i-1} - c_{i-m}^2 d_{i-m}}$$

This called **IC(1,1)**

For the model problem we have

$$\kappa(M^{-1}A) = O(\frac{1}{h^2})$$

but a good distribution of the eigenvalues

We can also keep more terms in L

For any pattern, we can do **IC** for H–matrices

Solving $Lw = c$, we have recurrences, so **IC** is not directly vectorizable

We have to modify it

The **Van der Vorst** method

Symmetrize M

$$(D^{-1/2}LD^{-1/2})(D^{-1/2}L^TD^{-1/2})D^{1/2}z = D^{-1/2}r$$

$$D^{-1/2}LD^{-1/2}y = D^{-1/2}r$$

$$D^{-1/2}LD^{-1/2} = \begin{pmatrix} E_1 & & & \\ B_2 & E_2 & & \\ & \ddots & \ddots & \\ & & B_n & E_n \end{pmatrix}$$

$$E_1y_1 = D_1^{-1/2}r_1$$

$$E_iy_i = -B_iy_{i-1} + D_i^{-1/2}r_i$$

E_i is lower bidiagonal

$$E_i = I + F_i$$

$$F_i = \begin{pmatrix} 0 & & & & & \\ x & 0 & & & & \\ & x & 0 & & & \\ & & x & 0 & & \\ & & & x & 0 & \\ & & & & x & 0 \end{pmatrix}$$

We approximate $E_i^{-1} = (I + F_i)^{-1}$ by a Neumann series

$$E_i^{-1} \approx I - F_i + F_i^2 - F_i^3 = (I - F_i)(I + F_i^2)$$

F_i^2 is easily computed

$$F_i^2 = \begin{pmatrix} 0 & & & & & \\ 0 & 0 & & & & \\ x & 0 & 0 & & & \\ & x & 0 & 0 & & \\ & & x & 0 & 0 & \\ & & & x & 0 & 0 \end{pmatrix}$$

Bidiagonal solves are done via matrix products $\longrightarrow$ vectorizable

Pb : number of iterations ?

We call this method **ICVDV**

Another way to vectorize or parallelize is to use **different orderings** of the unknowns

Modified preconditioners

$$M = LD^{-1}L^T = A + R$$

Modify D such that $rowsum(R) = 0$ or ch^2

For a 5 diagonal matrix :

$$\frac{1}{d_i} = (1 + ch^2)a_i - b_{i-1}(b_{i-1} + c_{i-1})d_{i-1}$$

$$-c_{i-m}(c_{i-m} + b_{i-m})d_{i-m}$$

More efficient for some problems

For the model problem

$$\kappa(M^{-1}A) = O(\frac{1}{h})$$

Same problems for vectorization

Block preconditioners

$$A = \begin{pmatrix} D_1 & A_2^T & & & \\ A_2 & D_2 & A_3^T & & \\ & \ddots & \ddots & \ddots & \\ & & A_{n-1} & D_{n-1} & A_n^T \\ & & & A_n & D_n \end{pmatrix}.$$

D_i is point tridiagonal strictly diagonally dominant, A_i is diagonal.

$$L = \begin{pmatrix} 0 & & & \\ A_2 & 0 & & \\ & \ddots & \ddots & \\ & & A_n & 0 \end{pmatrix}$$

The **block Cholesky** decomposition is

$$A = (\Sigma + L)\ \Sigma^{-1}\ (\Sigma + L^T)$$

$$\Sigma = \begin{pmatrix} \Sigma_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \Sigma_n \end{pmatrix}$$

$$\begin{cases} \Sigma_1 = D_1, \\ \Sigma_i = D_i - A_i\ \Sigma_{i-1}^{-1}\ {A_i}^T. \end{cases}$$

Matrices Σ_i are dense, but ...

The elements decay away from the diagonal

We approximate the dense matrices by tridiagonal matrices

we approximate T^{-1} by $trid(T^{-1})$, a tridiagonal matrix whose elements are the same as the corresponding ones of T^{-1}

INV

$$A = (\Delta + L)\ \Delta^{-1}\ (\Delta + L^T)$$

$$\Delta = \begin{pmatrix} \Delta_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \Delta_n \end{pmatrix}$$

$$\begin{cases} \Delta_1 = D_1, \\ \Delta_i = D_i - A_i\ trid(\Delta_{i-1}^{-1})\ {A_i}^T. \end{cases}$$

All the Δ_i are tridiagonal matrices

INV can be done for H–matrices

INV gives better results then **IC**. Why ?

If

$$\Delta_i = L_i L_i^T$$

$$M = SS^T$$

with S

INV can also be modified $\longrightarrow$ **MINV**

INV is not vectorizable, we have to solve tridiagonal systems

$$\Delta_i w = c$$

A way to vectorize is to replace Δ_i^{-1} by a banded approximation

Taking 7 diagonals of the inverse is fine with most problems

Then, everything is in vector mode $\longrightarrow$ **INVV**

Polynomial preconditioners

$$M^{-1} = P_k(A) = \sum \alpha_i A^i$$

P_k polynomial of degree k

The eigenvalues of $M^{-1}A$ are $P_k(\lambda_i)\lambda_i$

It is natural to ask for $P_k(\lambda)\lambda$ close to 1 on $[\lambda_{min}, \lambda_{max}]$

Example: find s that minimizes

$$\int_{\lambda_{min}}^{\lambda_{max}} (1 - \lambda s(\lambda))^2 w(\lambda)\ d\lambda$$

Common choice

$$w(\lambda) = (\lambda_{max} - \lambda)^\alpha (\lambda - \lambda_{min})^\beta$$

$\alpha \geq \beta \geq -1/2$: Jacobi polynomials

Model problem, $n = 50$, $\varepsilon = 10^{-6}$

k	no. of it.	*/n
1	64	960
2	44	880
3	34	850
4	28	840
5	23	805
6	21	840
7	18	810
8	16	800
9	15	825
10	13	780

Model problem, $n = 50$, $\varepsilon = 10^{-6}$, X–MP/4, x^0 random

precond	no. of it.	Mflops	time
DIAG	110	127	0.041
IC	33	30	0.108
MIC	24	28	0.078
INV	15	19	0.067
MINV	12	19	0.053
INV2	11	19	0.067
MINV2	9	18	0.051
VDV	34	79	0.042
INVV	16	97	0.021
POLY(3)	35	122	0.041

op	no precond	IC	INV	INV2	VDV	INVV
saxpy	32.	3.3	2.5	1.8	11.9	10.4
(.,.)	19.9	2.1	1.6	1.2	7.8	6.8
A p	47.9	4.9	3.7	2.8	18.	15.8
test	0.	0.9	0.7	0.5	3.2	3.1
prec	0.	88.6	91.4	93.5	58.6	63.8