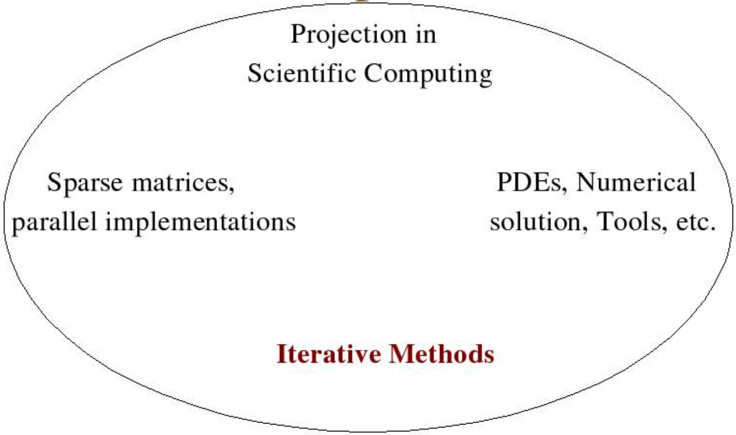
Iterative Methods in Linear Algebra

Stan Tomov

Innovative Computing Laboratory
Computer Science Department
The University of Tennessee

Wednesday April 18, 2012

Topics


Projection in
Scientific Computing
Sparse matrices,
parallel implementations
PDEs, Numerical
solution, Tools, etc.
Iterative Methods

Outline

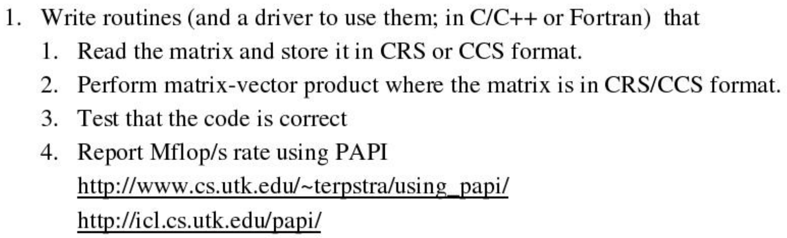
Part I

Discussion

Discussion

Homework #11

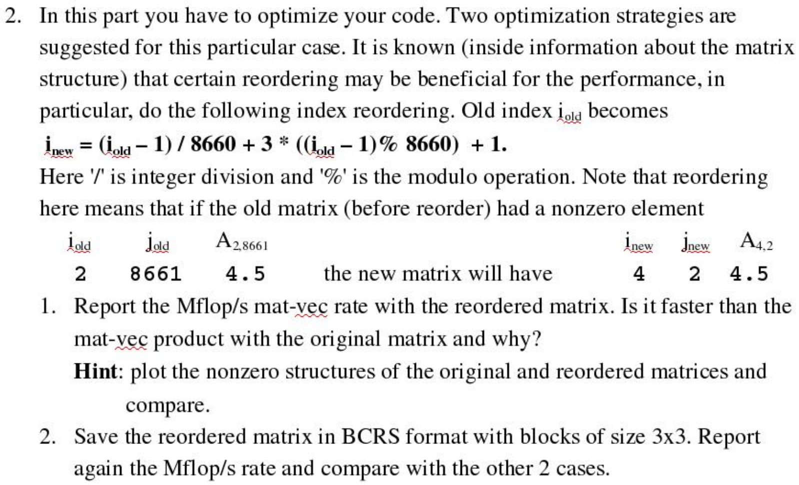
Part 1:



1. Write routines (and a driver to use them; in C/C++ or Fortran) that
   1. Read the matrix and store it in CRS or CCS format.
   2. Perform matrix-vector product where the matrix is in CRS/CCS format.
   3. Test that the code is correct
   4. Report Mflop/s rate using PAPI
      http://www.cs.utk.edu/~terpstra/using_papi/
      http://icl.cs.utk.edu/papi/

Discussion

Part 2:



2. In this part you have to optimize your code. Two optimization strategies are suggested for this particular case. It is known (inside information about the matrix structure) that certain reordering may be beneficial for the performance, in particular, do the following index reordering. Old index i_{old} becomes

   $$\mathbf{i_{new} = (i_{old} - 1) / 8660 + 3 * ((i_{old} - 1)\% \ 8660) + 1.}$$

   Here '/' is integer division and '%' is the modulo operation. Note that reordering here means that if the old matrix (before reorder) had a nonzero element

i_{old}	j_{old}	$A_{2,8661}$		i_{new}	j_{new}	$A_{4,2}$
2	8661	4.5	the new matrix will have	4	2	4.5

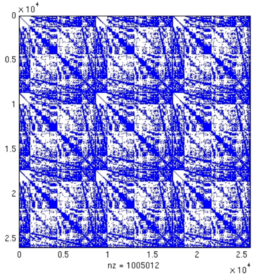
   1. Report the Mflop/s mat-vec rate with the reordered matrix. Is it faster than the mat-vec product with the original matrix and why?

      Hint: plot the nonzero structures of the original and reordered matrices and compare.
   2. Save the reordered matrix in BCRS format with blocks of size 3x3. Report again the Mflop/s rate and compare with the other 2 cases.

Discussion

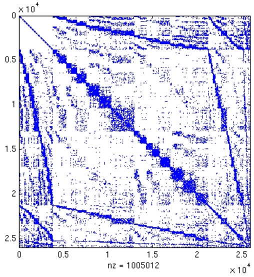
Original matrix:

```
>> load matrix.output
>> S = spconvert(matrix);
>> spy(S)
```



Reordered matrix:

```
>> load A.data
>> S = spconvert(A);
>> spy(S)
```



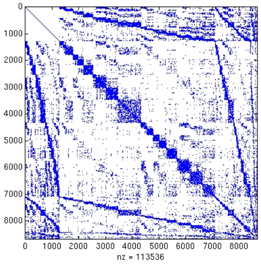
Discussion

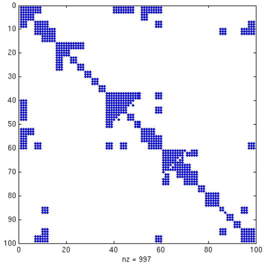
Original sub-matrix:

```
>> load matrix.output
>> S = spconvert(matrix);
>> spy(S(1:8660),1:8660))
```

Reordered sub-matrix:

```
>> load A.data
>> S = spconvert(A);
>> spy(S(8602:8700,8602:8700))
```





Discussion

Results on torc0:
Pentium III 933 MHz, 16 KB L1 and 256 KB L2 cache

- Original matrix: Mflop/s = 42.4568
- Reordered matrix: Mflop/s = 45.3954
- BCRS Mflop/s = 72.17374

Results on woodstock:
Intel Xeon 5160 Woodcrest 3.0GHz, L1 32 KB, L2 4 MB

- Original matrix: Mflop/s = 386.8
- Reordered matrix: Mflop/s = 387.6
- BCRS Mflop/s = 894.2

Homework 10

Questions?

Homework #10

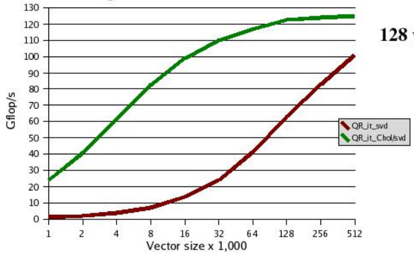
- PART I
  - PETSc
    Many iterative solvers
    We will see how is **projection** used in iterative methods
- PART II
  - hybrid CPU-GPUs computations

Hybrid CPU-GPU computations

Orthogonalization

||A-QR||		||I – Q' Q||		#iter	Cond(A)
LAPACK QR	QR_it_svd	LAPACK QR	QR_it_svd		
5.27E-015	5.40E-016	1.05E-014	1.40E-012	1	1.00E+002
7.42E-015	1.17E-015	7.82E-015	8.64E-015	2	1.00E+004
3.97E-015	2.19E-015	6.59E-015	8.40E-015	2	1.00E+006
5.93E-015	2.84E-015	5.57E-015	8.26E-015	2	1.00E+008
5.68E-015	1.35E-015	9.34E-015	6.08E-015	3	1.00E+012
5.01E-015	1.00E-015	5.57E-015	7.10E-015	3	1.00E+016

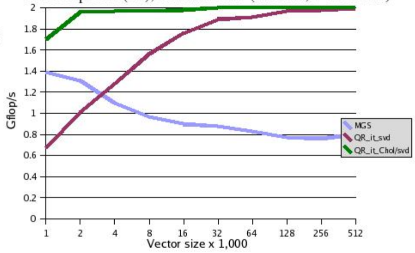
Hybrid CPU-GPU (NVIDIA Quadro FX 5600) computation as in Homework #9



128 vectors

CPU computation

AMD Opteron (tm), Processor 265 (1.8 Ghz, 1 GB cache)



Iterative Methods

Motivation

So far we have discussed and have seen:

- Many engineering and physics simulations lead to sparse matrices
  e.g. PDE based simulations and their discretizations based on
  - FEM/FVM
  - finite differences
  - Petrov-Galerkin type of conditions, etc.
- How to optimize performance on sparse matrix computations
- Some software how to solve sparse matrix problems (e.g. PETSc)

The question is:
Can we solve sparse matrix problems faster than using direct sparse methods?

- In certain cases Yes:
  using iterative methods

Sparse direct methods

Sparse direct methods

- These are based on Gaussian elimination (LU)
- Performed in sparse format

Are there limitations of the approach?

- Yes, they have fill-ins which lead to
  - more memory requirements
  - more flops being performed
- Fill-ins can become prohibitively high

Sparse direct methods

Consider LU for the matrix below

- a nonzero is represented by a *

*	*	*	*	*	*	*	*
*	*						
*		*					
*			*				
*				*			
*					*		
*						*	
*							*

1st step of LU factorization will introduce fill-ins

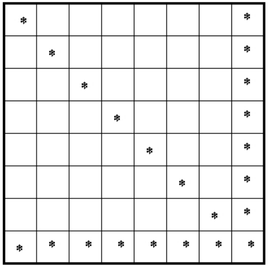
- marked by an F

*	*	*	*	*	*	*	*
*	*	F	F	F	F	F	F
*		*					
*			*				
*				*			
*					*		
*						*	
*							*

Sparse direct methods

Fill-ins can be improved by reordering

- Remember: we talked about it in slightly different context (for speed and parallelization)
- Consider the following reordering



These were extreme cases

- but still, the problem exists

Sparse methods

Sparse direct *vs* dense methods

- Dense takes $O(n^2)$ storage, $O(n^3)$ flops, runs within peak performance
- Sparse direct can take $O(\#nonz)$ storage, and $O(\#nonz)$ flops, but these can also grow due to fill-ins and performance is bad
- with $\#nonz << n^2$ and 'proper' ordering it pays off to do sparse direct

Software (table from Julien Langou)

- http://www.netlib.org/utk/people/JackDongarra/la-sw.html

Package	Support	Type		Language			Mode			
		Real	Complex	f77	c	c++	Seq	Dist	SPD	Gen
DSCPACK	yes	X			X		X	M	X	
HSL	yes	X	X	X			X		X	X
MFACT	yes	X			X		X		X	
MUMPS	yes	X	X	X	X		X	M	X	X
PSPASES	yes	X		X	X			M	X	
SPARSE	?	X	X		X		X		X	X
SPOOLES	?	X	X		X		X	M		X
SuperLU	yes	X	X	X	X		X	M		X
TAUCS	yes	X	X		X		X		X	X
UMFPACK	yes	X	X		X		X			X
Y12M	?	X		X			X		X	X

Sparse methods

What about Iterative Methods? Think for example

$$x_{i+1} = x_i + P(b - Ax_i)$$

Pluses:

- Storage is only $O(\#nonz)$ (for the matrix and a few working vectors)
- Can take only a few iterations to converge (e.g. $<< n$)
  - for $P = A^{-1}$ it takes 1 iteration (check)!
- In general easy to parallelize

Sparse methods

What about Iterative Methods? Think for example

$$x_{i+1} = x_i + P(b - Ax_i)$$

Minuses:

- Performance can be bad
  (as we saw in Lecture 13 and today's discussion)
- Convergence can be slow or even stagnate
  But can be improved with preconditioning
  - Think of P as a preconditioner, an operator/matrix $P \approx A^{-1}$
  - Optimal preconditioners (e.g. multigrid can be) lead to convergence in $O(1)$ iterations

Part II

Stationary Iterative Methods

Stationary Iterative Methods

Can be expressed in the form

$$x_{i+1} = Bx_i + c$$

where B and c do not depend on i

- older, simpler, easy to implement, but usually not as effective (as nonstationary)
- examples: Richardson, Jacobi, Gauss-Seidel, SOR, etc. (next)

Richardson Method

Richardson iteration

$$x_{i+1} = x_i + (b - Ax_i) \;= (I - A)x_i + b \tag{1}$$

i.e. B from the definition above is $B = I - A$

Denote $e_i = x - x_i$ and rewrite (1)

$$\begin{aligned} x - x_{i+1} &= x - x_i - (Ax - Ax_i) \\ e_{i+1} &= e_i - Ae_i \\ &= (I - A)e_i \\ ||e_{i+1}|| &\leq ||(I - A)e_i|| \leq ||I - A||\; ||e_i|| \leq ||I - A||^2\; ||e_{i-1}|| \\ &\leq \cdots \leq ||I - A||^i ||e_0|| \end{aligned}$$

i.e. for convergence ($e_i \longrightarrow 0$) we need

$$||I - A|| < 1$$

for some norm $|| \cdot ||$, e.g. when $\rho(A) < 1$.

Jacobi Method

Jacobi Method

$$x_{i+1} = x_i + D^{-1}(b - Ax_i) \;= (I - D^{-1}A)x_i + D^{-1}b \qquad (2)$$

where D is the diagonal of A (assumed nonzero; $B = I - D^{-1}A$ from the definition)

Denote $e_i = x - x_i$ and rewrite (2)

$$\begin{aligned} x - x_{i+1} &= x - x_i - D^{-1}(Ax - Ax_i) \\ e_{i+1} &= e_i - D^{-1}Ae_i \\ &= (I - D^{-1}A)e_i \\ ||e_{i+1}|| &\leq ||(I - D^{-1}A)e_i|| \leq ||I - D^{-1}A||\,||e_i|| \leq ||I - D^{-1}A||^2\,||e_{i-1}|| \\ &\leq \cdots \leq ||I - D^{-1}A||^i ||e_0|| \end{aligned}$$

i.e. for convergence ($e_i \rightarrow 0$) we need

$$||I - D^{-1}A|| < 1$$

for some norm $||\cdot||$, e.g. when $\rho(D^{-1}A) < 1$

Gauss-Seidel Method

Gauss-Seidel Method

$$x_{i+1} = (D - L)^{-1}(Ux_i + b) \tag{3}$$

where $-L$ is the lower and $-U$ the upper triangular part of A, and D is the diagonal.

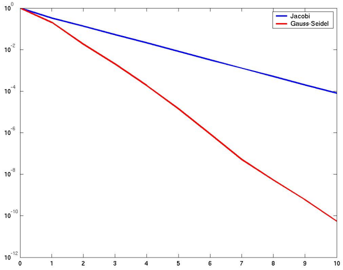
Equivalently:

$$\begin{cases} x_1^{(i+1)} = (b_1 - a_{12}x_2^{(i)} - a_{13}x_3^{(i)} - a_{14}x_4^{(i)} - \ldots - a_{1n}x_n^{(i)})/a_{11} \\ x_2^{(i+1)} = (b_2 - a_{21}x_1^{(i+1)} - a_{23}x_3^{(i)} - a_{24}x_4^{(i)} - \ldots - a_{2n}x_n^{(i)})/a_{22} \\ x_3^{(i+1)} = (b_2 - a_{31}x_1^{(i+1)} - a_{32}x_2^{(i+1)} - a_{34}x_4^{(i)} - \ldots - a_{3n}x_n^{(i)})/a_{22} \\ \vdots \\ x_n^{(i+1)} = (b_n - a_{n1}x_1^{(i+1)} - a_{n2}x_2^{(i+1)} - a_{n3}x_3^{(i+1)} - \ldots - a_{n,n-1}x_{n-1}^{(i+1)})/a_{nn} \end{cases}$$

A comparison (from Julien Langou slides)

Gauss-Seidel method

$$A = \begin{pmatrix} 10 & 1 & 2 & 0 & 1 \\ 0 & 12 & 1 & 3 & 1 \\ 1 & 2 & 9 & 1 & 0 \\ 0 & 3 & 1 & 10 & 0 \\ 1 & 2 & 0 & 0 & 15 \end{pmatrix} \quad b = \begin{pmatrix} 23 \\ 44 \\ 36 \\ 49 \\ 80 \end{pmatrix}$$



$\|b - Ax^{(i)}\|_2 / \|b\|_2$, function of i, the number of iterations

$x^{(0)}$	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$x^{(4)}$	$\dots$	
0.0000	2.3000	0.8783	0.9790	0.9978		1.0000
0.0000	3.6667	2.1548	2.0107	2.0005		2.0000
0.0000	2.9296	3.0339	3.0055	3.0006		3.0000
0.0000	3.5070	3.9502	3.9962	3.9998		4.0000
0.0000	4.6911	4.9875	5.0000	5.0001		5.0000

A comparison (from Julien Langou slides)

Jacobi Method:

$$\begin{cases} x_1^{(i+1)} = (b_1 - a_{12}x_2^{(i)} - a_{13}x_3^{(i)} - a_{14}x_4^{(i)} - \dots - a_{1n}x_n^{(i)})/a_{11} \\ x_2^{(i+1)} = (b_2 - a_{21}x_1^{(i)} - a_{23}x_3^{(i)} - a_{24}x_4^{(i)} - \dots - a_{2n}x_n^{(i)})/a_{22} \\ x_3^{(i+1)} = (b_2 - a_{31}x_1^{(i)} - a_{32}x_2^{(i)} - a_{34}x_4^{(i)} - \dots - a_{3n}x_n^{(i)})/a_{22} \\ \vdots \\ x_n^{(i+1)} = (b_n - a_{n1}x_1^{(i)} - a_{n2}x_2^{(i)} - a_{n3}x_3^{(i)} - \dots - a_{n,n-1}x_{n-1}^{(i)})/a_{nn} \end{cases}$$

Gauss-Seidel method

$$\begin{cases} x_1^{(i+1)} = (b_1 - a_{12}x_2^{(i)} - a_{13}x_3^{(i)} - a_{14}x_4^{(i)} - \dots - a_{1n}x_n^{(i)})/a_{11} \\ x_2^{(i+1)} = (b_2 - a_{21}x_1^{(i+1)} - a_{23}x_3^{(i)} - a_{24}x_4^{(i)} - \dots - a_{2n}x_n^{(i)})/a_{22} \\ x_3^{(i+1)} = (b_2 - a_{31}x_1^{(i+1)} - a_{32}x_2^{(i+1)} - a_{34}x_4^{(i)} - \dots - a_{3n}x_n^{(i)})/a_{22} \\ \vdots \\ x_n^{(i+1)} = (b_n - a_{n1}x_1^{(i+1)} - a_{n2}x_2^{(i+1)} - a_{n3}x_3^{(i+1)} - \dots - a_{n,n-1}x_{n-1}^{(i+1)})/a_{nn} \end{cases}$$

Gauss-Seidel is the method with better numerical properties (less iterations to convergence). Which is the method with better efficiency in term of implementation in sequential or parallel computer? Why?

In Gauss-Seidel, the computation of $x_{k+1}^{(i+1)}$ implies the knowledge of $x_k^{(i+1)}$. Parallelization is impossible.

Convergence

Convergence can be very slow

Consider a modified **Richardson method**:

$$x_{i+1} = x_i + \tau(b - Ax_i) \tag{4}$$

Convergence is **linear**, similarly to Richardson we get

$$||e_{i+1}|| \leq ||I - \tau A||^i \ ||e_0||$$

but can be very slow (if $||I - \tau A||$ is close to 1), e.g. let

- A be symmetric and positive definite (SPD)
- the matrix norm in $||I - \tau A||$ is induced by $||\cdot||_2$

Then the best choice for τ ($\tau = \frac{2}{\lambda_1 + \lambda_N}$) would give

$$||I - \tau A|| = \frac{k(A) - 1}{k(A) + 1}$$

where k(A) $= \frac{\lambda_N}{\lambda_1}$ is the condition number of A.

Convergence

Note:

- The rate of convergence depend on the condition number $k(A)$
- Even for the the best τ the rate of convergence

$$\frac{k(A)-1}{k(A)+1}$$

  is slow (i.e. close to 1) for large $k(A)$
- We will see convergence of nonstationary methods also depend on $k(A)$ but is better, e.g. compare with CG

$$\frac{\sqrt{k(A)}-1}{\sqrt{k(A)}+1}$$

Part III

Nonstationary Iterative Methods

Nonstationary Iterative Methods

The methods involve information that changes at every iteration

- e.g. inner-products with residuals or other vectors, etc.

The methods are

- newer, harder to understand, but more effective
- in general based on the idea of orthogonal vectors and subspace projections
- examples: Krylov iterative methods
    - CG/PCG, GMRES, CGNE, QMR, BiCG, etc.

Krylov Iterative Methods

Krylov Iterative Methods

Krylov subspaces: these are the spaces

$$K_i(A, r) = span\{r,\ Ar,\ A^2r, \ldots,\ A^{i-1}r\}$$

Krylov iterative methods find approximation x_i to x where

$$Ax = b,$$

as a minimization on $K_i(A, r)$.
We have seen how this can be done for example by projection, i.e. by the the Petrov-Galerkin conditions:

$$(Ax_i, \phi) = (b, \phi) \quad \text{for} \ \ \forall \phi \in K_i(A, r)$$

Krylov Iterative Methods

In general we

- expend the Krylov subspace by a matrix-vector product, and
- do a minimization/projection in it.

Various methods result by specific choices of expansion and minimization/projection.

Krylov Iterative Methods

An example: The **Conjugate Gradient Method (CG)**

- The method is for SPD matrices
- There is a way at iteration i to construct new 'search direction' p_i s.t.

$$span\{p_0, p_1, \ldots, p_i\} \equiv K_{i+1}(A, r_0) \text{ and } (Ap_i, p_j) = 0 \text{ for } i \neq j.$$

Note: A is SPD $\Rightarrow (Ap_i, p_j) \equiv (p_i, p_j)_A$ can be used as an inner product, i.e. $p_0, \ldots, p_i$ is an $(\cdot, \cdot)_A$ orthogonal basis for $K_{i+1}(A, r_0)$

$\Rightarrow$ we can easily find $x_{i+1} \approx x$ as

$$\begin{aligned} x_{i+1} &= x_0 + \alpha_0 p_0 + \cdots + \alpha_i p_i \quad \text{s.t.} \\ (Ax_{i+1}, p_j) &= (b, p_j) \quad \text{for } j = 0, \ldots, i \end{aligned}$$

Namely, because of the $(\cdot, \cdot)_A$ orthogonality of $p_0, \ldots, p_i$ at iteration $i+1$ we have to find only α_i

$$(Ax_{i+1}, p_i) = (A(x_i + \alpha_i p_i), p_i) = (b, p_i), \quad \Rightarrow \quad \alpha_i = \frac{(r_i, p_i)}{(Ap_i, p_i)}$$

Note: x_i above actually can be replaced by any $x_0 + v$, $v \in K_i(A, r_0)$ (Why?)

Learning Goals

A brief introduction to iterative methods

- Motivation for iterative methods and links to previous lectures, namely
  - PDE discretization and sparse matrices
  - Optimized implementations
- Stationary iterative methods
- Nonstationary iterative methods and links to building blocks that we have already covered
  - Projection/Minimization
  - Orthogonalization
- Krylov methods; an example with CG;
  to see more examples ... (next lecture)