Technische Universität München
Institut für Informatik
Prof. Dr. Thomas Huckle
Dipl.-Math. Benjamin Uekermann

WiSe 2016/17

Parallel Numerics

Exercise 8: Stationary Methods

1 Stationary Methods Revisited

To solve the equation system $Ax = b$ stationary methods split up the matrix A into $A = M - N$:

$$\begin{aligned} Ax &= b \\ (M-N)x &= b \\ Mx &= Nx + b \\ Mx^{(n+1)} &= Nx^{(n)} + b \end{aligned}$$

a) Give the Richardson, Jacobi and Gauß–Seidel method using matrix notation.
Consider splitting $A = A - I + I$ with identity I:

$$\begin{aligned} Ax &= b \\ (A-I+I)x &= b \\ (A-I)x + x &= b \\ x &= b - (A-I)x \\ x &= (I-A)x + b \end{aligned}$$

With $M = I$ and $N = I - A$ the Richardson method is $x^{(n+1)} = (I-A)x^{(n)} + b$.

Consider splitting $A = D - (-L - U)$ where D is diagonal part of A, L is strict lower and U is strict upper triangular part, respectively:

$$\begin{aligned} (D+L+U)x &= b \\ (D-(-L-U))x &= b \\ Dx - (-L-U)x &= b \\ Dx &= b + (-L-U)x \\ x &= D^{-1}(-L-U)x + D^{-1}b \end{aligned}$$

With $M = D$ and $N = -L - U$ the Jacobi method is $x^{(n+1)} = D^{-1}(-L-U)x^{(n)} + D^{-1}b$.

Consider splitting $A = (D + L) - -U$:

$$\begin{aligned} ((D+L) - U)x &= b \\ (D+L)x - (-U)x &= b \\ x &= (D+L)^{-1}(-U)x + (D+L)^{-1}b \end{aligned}$$

With $M = D+L$ and $N = -U$ the Gauß-Seidel method is $x^{(n+1)} = (D-L)^{-1}(-U)x^{(n)} + (D-L)^{-1}b$.

b) Give the Richardson, Jacobi and Gauß-Seidel method in pseudo code and identify GAXPYs, SAXPYs, ...

Richardson:

```
while not converged
    for j = 1...n
        x_j^(n+1) = Σ_{i=1}^{n} (-a_ji x_i^(n)) + x_j^(n) + b_j
                    \_____scalar product_____/
```

Pseudo code for Richardson with SAXPY for inner for-loop and GAXPY for outer for-loop:

```
x^(n+1) = 0
for j = 1, ..., n
    for i = 1, ..., n
        x_i^(n+1) = x_i^(n+1) - x_j^(n) a_ij
    end
end
x^(n+1) = x^(n+1) + x^(n) + b
```

Jacobi:

```
while not converged
    for j = 1, ..., n
        x_j^(n+1) = (1/a_jj) ( Σ_{i=1, i≠j}^{n} (-a_ji x_i^(n)) + b_j )
```

Pseudocode for Jacobi with SAXPY for inner for-loop and GAXPY for outer for-loop:

```
x^(n+1) = 0
for j = 1, ..., n
    for i = 1, ..., n
        if i ≠ j
            x_i^(n+1) = x_i^(n+1) - x_j^(n) a_ij
        end
    end
end
for i = 1, ..., n
```

$x_i^{(n+1)} = \frac{x_i^{(n+1)} + b_i}{a_{ii}}$

end

Gauß-Seidel:

while not converged

for $j = 1, ..., n$

$$x_j^{(n+1)} = \frac{1}{a_{jj}} \left(\sum_{i=j+1}^{n} (-a_{ji} x_i^{(n)}) + b_j + \sum_{i=1}^{j-1} (-a_{ji} x_i^{(n+1)}) \right)$$

In matrix notation the solution $x^{(n+1)}$ is obtained by solving

$$(D - L)x^{(n+1)} = \underbrace{Ux^{(n)}}_{GAXPY} + b.$$

Pseudocode for Gauß-Seidel. No SAXPYs in this formulation:

$x^{(n+1)} = 0$

for $j = 1, ..., n$

$c = b_j$

for $i = j + 1, ..., n$

$c = c - x_i^{(n)} a_{ji}$

end

for $i = 1, ..., j - 1$

$c = c - x_i^{(n+1)} a_{ji}$

end

$x_j^{(n+1)} = \frac{c}{a_{jj}}$

end

GAXPY available if $Ux^{(n)}$ is computed in a first step completely on its own. Afterwards the triangular solve is performed.

c) For the weighted relaxation schemes, one scales the iteration rule above with a factor ω and adds the trivial iteration $x^{(n+1)} = x^{(n)}$. Derive the $\omega - JAC$ and SOR (Successive-Over-Relaxation) scheme in matrix notation.

$\omega - JAC$ relaxation scheme by using relaxation parameter ω in convex combination:

$$\begin{aligned}
x^{(n+1)} &= (1 - \omega)x^{(n)} + \omega x_{JAC}^{(n+1)} \\
x^{(n+1)} &= (1 - \omega)x^{(n)} + \omega D^{-1}(-L - U)x^{(n)} + \omega D^{-1} b \\
x^{(n+1)} &= \underbrace{[(1 - \omega)I + \omega D^{-1}(-L - U)]}_{=:R_\omega} x^{(n)} + \omega D^{-1} b \\
x^{(n+1)} &= R_\omega x^{(n)} + \omega D^{-1} b
\end{aligned}$$

Similarly for SOR:

$$\begin{aligned}
x^{(n+1)} &= (1 - \omega)x^{(n)} + \omega x_{GS}^{(n+1)} \\
&\dots \\
x^{(n+1)} &= \left[(1 - \omega)I + \omega(D - L)^{-1}U\right] x^{(n)} + \omega(D - L)^{-1} b
\end{aligned}$$

2 Residual-based Notation

The residual is defined as

$$r \quad = \quad b - Ax$$

a) Give the Richardson, Jacobi and Gauß-Seidel method using the residual.
In general: $x^{(n+1)} = x^{(n)} + M^{-1}r^{(n)}$.
Richardson with residual: $x^{(n+1)} = x^{(n)} + r^{(n)}$
Jacobi with residual: $x^{(n+1)} = x^{(n)} + D^{-1}r^{(n)}$
Gauß-Seidel with residual: $x^{(n+1)} = x^{(n)} + (D+L)^{-1}r^{(n)}$

b) Give the $\omega - JAC$ and SOR scheme using the residual.
In general: $x^{(n+1)} = x^{(n)} + \omega M^{-1}r^{(n)}$.
$\omega - JAC$ with residual: $x^{(n+1)} = x^{(n)} + \omega D^{-1}r^{(n)}$
SOR scheme with residual: $x^{(n+1)} = x^{(n)} + \omega(D+\omega L)^{-1}r^{(n)}$

c) Give a sketch of the data dependency graph for both computing the residual and updating the solution according to the Jacobi and the GS scheme. (To simplify matters: Assume that A is tridiagonal)
For simplicity, the following figures only depict the dependencies of the current solution $x_i^{(n)}$ of the residuals $r_i^{(n-1)}$ and NOT of the previous solution $x_i^{(n-1)}$.

Figure 1 shows the data dependencies between x and r for Jacobi.

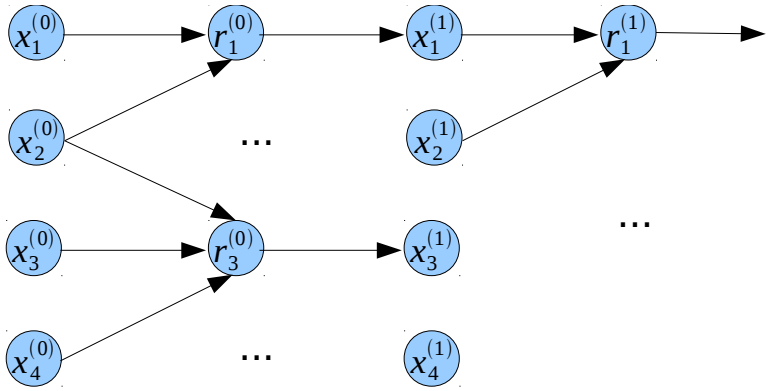

Figure 1: (Simplified) Data dependency graph of Jacobi method.

Figure 2 shows the data dependency between x and r for Gauß-Seidel with residual formulation $x_j^{(n+1)} = x_j^{(n)} + (D-L)^{-1}r_j^{(n,j-1)}$ where $j-1$ updates are considered in r_j.

d) Which parallel algorithms for matrix vector products do you know (already)?
In the residual notation all solvers are reduced to matrix-vector-products. Thus, one can use e.g. Cannon's algorithm, cyclic assignment, etc..

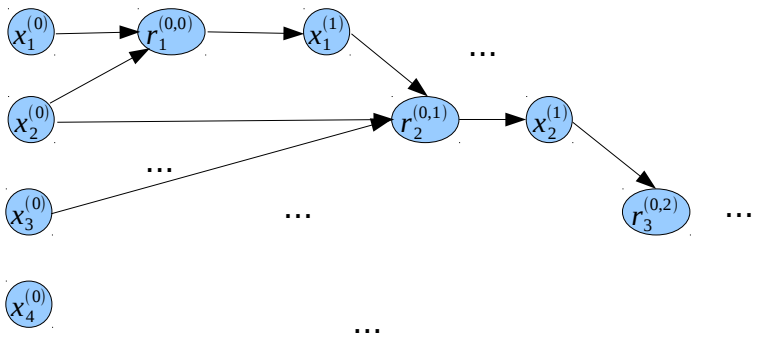

Figure 2: (Simplified) Data dependency graph of Gauß-Seidel method.

3 SOR Implementation

In this section, we want to implement the SOR. As simplification for the following algorithms we assume that the iteration index k is always running from 0 to a maximum number k_{stop}. With the definitions

$$\alpha_i := \frac{\omega}{a_{ii}} \quad \text{for } i = 1, \ldots, n \qquad \text{and} \qquad b_{ij} := \begin{cases} -a_{ij} & \text{for } i \neq j \\ \frac{1-\omega}{\omega} a_{ii} & \text{for } i = j \end{cases}$$

a serial algorithm for the SOR method can be given as follows:

```
for k = 0 to k_stop
      for i = 1 to n
          s := d_i
           for j = 1 to n
               s := s + b_ij x_j                    (*)
          x_i := α_i s
```

A parallel algorithm can be implemented in a similar way: The b_{ij} are distributed columnwise on p processors in a cyclic way (cp. the Parallel Gauss Elimination, Exercise 6). Every processor calculates only a part of the sum in $(*)$. Following, the x_i are calculated successively on different processors after receiving the parts of the sum form the other processors. Suitable communication is necessary. The parallel algorithm has the following shape:

```
for k = 0 to k_stop
      for i = 1 to n
          a := Σ_{j∈mycolumns} b_ij x_j
           if i ∈ mycolumns
                Get a from all other processors and calculate s := Σ_p a
               x_i := α_i (s + d_i)
```

Implement the serial and parallel algorithm! On which processor do you find the solution x after running the parallel algorithm? Can you observe a speedup?

See source code to the corresponding tutorial on our web page.