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.

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

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.

2 Residual-based Notation

The residual is defined as

$$r = b - Ax$$

a) Give the Richardson, Jacobi and Gauß-Seidel method using the residual.

b) Give the $\omega - JAC$ and SOR scheme using the residual.

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)

d) Which parallel algorithms for matrix vector products do you know (already)?

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

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

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
            $s := s + b_{ij} x_j$ $(*)$
        $x_i := \alpha_i s$

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
        $a := \sum_{j \in mycolumns} b_{ij} x_j$
        if $i \in mycolumns$
            Get a from all other processors and calculate $s := \sum_p a$
            $x_i := \alpha_i (s + d_i)$

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?