Technische Universität München
Institut für Informatik
Prof. Dr. Thomas Huckle
Dipl.-Inf. Christoph Riesinger

WiSe 2015/16

# Parallel Numerics

Exercise 6: Tridiagonal Matrices, Hockney/Golub method & Message Tags

## 1 Block-wise Gaussian Elimination for Tridiagonal Matrices

The solution of a linear equation system

$$Ax = d$$

with a tridiagonal matrix $A \in \mathbb{R}^{n \times n}$ and $d \in \mathbb{R}^n$ should be calculated. The notation is as follows:

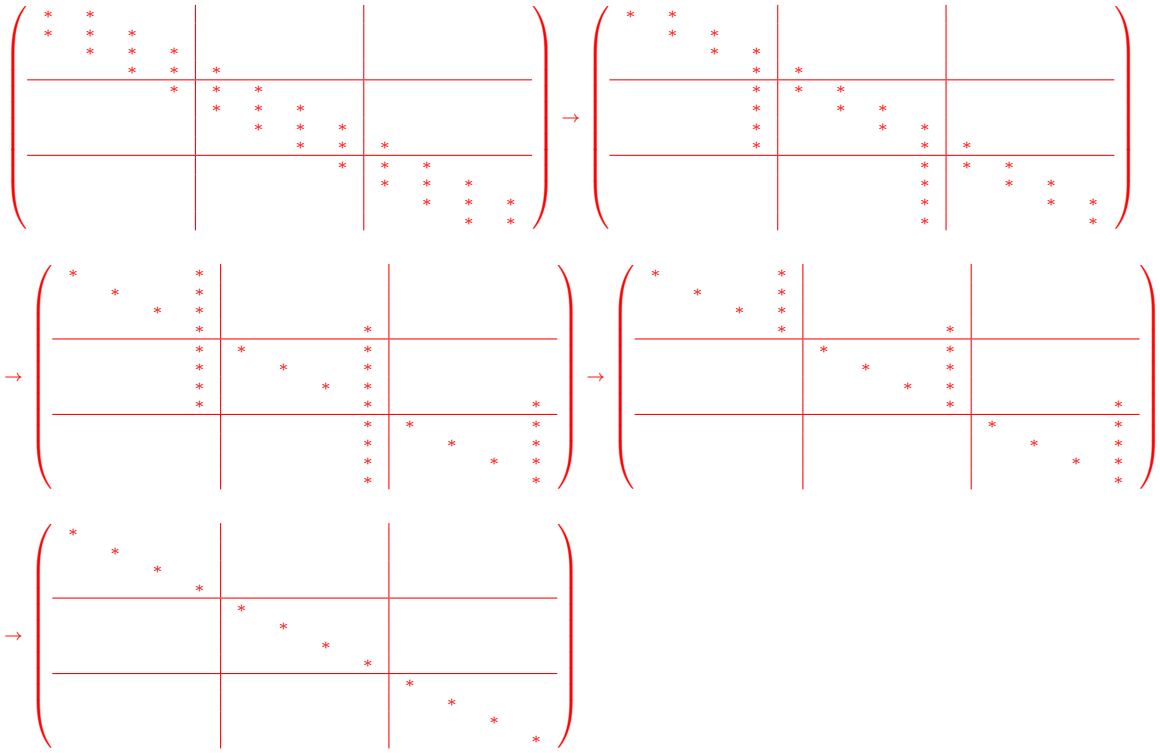
$$A = \begin{pmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & & \mathbf{0} \\ & a_3 & b_3 & c_3 & & \mathbf{0} \\ & & \ddots & \ddots & \ddots & \\ & \mathbf{0} & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{pmatrix}, \quad d = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix}$$

a) Examine the behaviour of a parallel Gaussian Elimination that assignes the columns cyclically to the different nodes of a parallel computer. What is an optimal number of nodes $p$?
Alternating column assignment of each column means that each column (processor) has to compute and broadcast only the value $L_{k+1k} = \frac{a_{k+1}}{b_k}$ to its neighboring processor. There has to be done only the update $b_{k+1} = b_{k+1} - L_{k+1k}c_k$. As there is only work for one processor, all other processors remain idle. Therefore, the optimal number of processors is 1.

Just for completeness: There is a more sophisticated approach by Ivo Babuvška (DOI: 10.1137/0709008) where the optimal number of processors is 2 (still a quite small number). There, Gaussian Elimination is done "from both sides" of the tridiagonal matrix: From the top left of the matrix (classical Gaussian Elimination) and from the bottom right.

b) Split up the rows of $A$ into $p$ blocks. Derive the sparsity pattern and the fill–ins for the following algorithm:

1. In the first step, elements of lower subdiagonal are eliminated. This is done by addition of an appropriate multiple of the row above to element to eliminate, thus classical Gaussian elimination. There are no dependencies between the partitions.
2. In the second step elements of the upper side diagonal are eliminated in a similar way: By addition of a suitable multiple of the second line from below of each partition to the rows above this row all these elements disappear but the one in the last row. For the elimination of this one the modified first row of the lower neighbouring partition is used. Note that only the elimination of the last element of each partition depends on the results of the elimination process of another partition.
3. In the third step, elements below the main diagonal are eliminated. This is done by addition of an appropriate multiple of the last row of the upper neighbouring partition to each row of the partition; the first partition is skipped naturally. This is how the elimination processes of the single partitions are connected here.
4. Finally the elements in the upper part are removed. Each partition uses its last row, whose nondiagonal element is eliminated by addition of a suitable multiple of the last row of the lower neighbouring partition. Again the elimination processes depend on each other only through the the elimination of a single element.



c) Implement the algorithm.
See sourcecode to corresponding tutorial on webpage.

# 2 MPI Message Tags

Given is the following send process running on `p0`:

```
1       int a=1;
```

```
2       int b=2;
3       int c=3;
4       MPI_Send(&a, 1, MPI_INT, p1, 0, MPI_COMM_WORLD);
5       MPI_Send(&b, 1, MPI_INT, p1, 0, MPI_COMM_WORLD);
6       MPI_Send(&c, 1, MPI_INT, p1, 1, MPI_COMM_WORLD);
```

a) What is the result of the following receive code fragment running on `p1`?

```
1       int u;
2       int v;
3       int w;
4       MPI_Recv(&u, 1, MPI_INT, p0, 0, MPI_COMM_WORLD);
5       MPI_Recv(&v, 1, MPI_INT, p0, 0, MPI_COMM_WORLD);
6       MPI_Recv(&w, 1, MPI_INT, p0, 0, MPI_COMM_WORLD);
```

$u \to 1$, $v \to 2$. This code will not terminate. There is a message left in the receive queue with tag `1` but the blocking receive waits for a message with tag `0`.

b) What is the result of the following receive code fragment running on `p1`?

```
1       int u;
2       int v;
3       int w;
4       MPI_Recv(&u, 1, MPI_INT, p0, 0, MPI_COMM_WORLD);
5       MPI_Recv(&v, 1, MPI_INT, p0, 1, MPI_COMM_WORLD);
6       MPI_Recv(&w, 1, MPI_INT, p0, 0, MPI_COMM_WORLD);
```

$u \to 1$, $v \to 3$, $w \to 2$. MPI messages with same tag from same sender are not allowed to overtake.

# 3 The Hockney/Golub Method

Again, the solution of a tridiagonal linear equation system should be calculated. With $N := \lceil \log_2 n \rceil$ and the definitions

$$
\begin{aligned}
a_i^{(0)} &:= a_i\,, \quad i = 2, \ldots, n\,, & a_1^{(0)} &:= 0 \\
b_i^{(0)} &:= b_i\,, \quad i = 1, \ldots, n\,, & & \\
c_i^{(0)} &:= c_i\,, \quad i = 1, \ldots, n-1\,, & c_n^{(0)} &:= 0 \\
d_i^{(0)} &:= d_i\,, \quad i = 1, \ldots, n\,, & &
\end{aligned}
$$

$$
a_i^{(k)} := c_i^{(k)} := d_i^{(k)} := 0\,, \quad b_i^{(k)} := 1 \qquad \text{for } i \in \mathbb{Z} \backslash \{1, \ldots, n\}\,, \quad k = 0, \ldots, N
$$

the values $a_i^{(k)}$, $b_i^{(k)}$, $c_i^{(k)}$, $d_i^{(k)}$ are calculated recursively:

$$\begin{aligned}
a_i^{(k)} &:= \alpha_i^{(k-1)}\, a_{i-2^{k-1}}^{(k-1)} \\
c_i^{(k)} &:= \gamma_i^{(k-1)}\, c_{i+2^{k-1}}^{(k-1)} \\
b_i^{(k)} &:= \alpha_i^{(k-1)}\, c_{i-2^{k-1}}^{(k-1)} + b_i^{(k-1)} + \gamma_i^{(k-1)}\, a_{i+2^{k-1}}^{(k-1)} \\
d_i^{(k)} &:= \alpha_i^{(k-1)}\, d_{i-2^{k-1}}^{(k-1)} + d_i^{(k-1)} + \gamma_i^{(k-1)}\, d_{i+2^{k-1}}^{(k-1)}
\end{aligned}$$

with

$$\begin{aligned}
\alpha_i^{(k-1)} &:= -a_i^{(k-1)} / b_{i-2^{k-1}}^{(k-1)} \\
\gamma_i^{(k-1)} &:= -c_i^{(k-1)} / b_{i+2^{k-1}}^{(k-1)}
\end{aligned}$$

Then the following equation is valid for $i \in \mathbb{Z}$:

$$a_i^{(k)}\, x_{i-2^k} + b_i^{(k)}\, x_i + c_i^{(k)}\, x_{i+2^k} = d_i^{(k)}$$

The Hockney/Golub method uses this formula to calculate the solution of the linear system and can be applied in the following way:

1. `Calculate the values` $a_i^{(k)}$, $b_i^{(k)}$, $c_i^{(k)}$, $d_i^{(k)}$ `for` $k = 1, \ldots, N$, $i = 1, \ldots, n$
2. `Calculate` $x_i = d_i^{(N)} / b_i^{(N)}$ `for` $i = 1, \ldots, n$.

Understand and implement this algorithm! Compare this method to the Gaussian elimination method! Why can the Hockney/Golub method be advantageous?
See sourcecode to corresponding tutorial on webpage.

Hockney/Golub method only possible for $b_i^{(k)} \neq 0$. This condition is not satisfied automatically for arbitrary nonsingular tridiagonal matrices. Even when GE is possible without pivoting, Hockney/Golub must not be possible. If $A$ is $H$-matrix then Hockney/Golub is possible for every rhs $d$.

Hockney/Golub can be parallelized.