Technical University of Munich
Department of Informatics
Prof. Dr. Thomas Huckle
Dr. Benjamin Uekermann

Winter term 2017/18

Parallel Numerics

Exercise 5: Parallel LU Decomposition & Collective Operations

1 LU Decomposition

To calculate the solution of a linear equation system

$$Ax = b$$

with a non singular matrix $A = (a_{i,j}) \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$ the LU decomposition algorithm can be applied. The algorithm decomposes the matrix A in a lower triangular matrix L and an upper triangular matrix U: $A = LU$ (cp. lecture notes).

A recursive procedure for calculating the upper triangular matrix $U =: A^{(n)}$ from the matrix A can be given as follows:

1. Define $A^{(1)} = (a_{i,j}^{(1)}) := A$
2. Calculate for $k = 1, \ldots, n-1$ the values $l_{i,k}$, $i = k+1, \ldots, n$ and the matrices $A^{(k+1)} = (a_{i,j}^{(k+1)})$ iteratively with

$$l_{i,k} := a_{i,k}^{(k)} / a_{k,k}^{(k)}$$

$$a_{i,j}^{(k+1)} := \begin{cases} a_{i,j}^{(k)} - l_{i,k} a_{k,j}^{(k)} & \text{for } i \in \{k+1, \ldots, n\},\ j \in \{k, \ldots, n\} \\ a_{i,j}^{(k)} & \text{otherwise} \end{cases}$$

A serial implementation of this iterative algorithm could have the following form ("kij-loop-arrangement"):

```
for k = 1 to n − 1
    i = k + 1 to n
        l_{i,k} := a_{i,k} / a_{k,k}
        for j = k to n
            a_{i,j} := a_{i,j} − l_{i,k} a_{k,j}
```

a) Define the term *in situ* storage scheme.
In situ (Latin: in the place) storage scheme means to store both L and R within one

matrix as $L + R - I$. Example for $L, R, I \in \mathbb{R}^{5\times5}$:

$$\begin{pmatrix} r & r & r & r & r \\ l & r & r & r & r \\ l & l & r & r & r \\ l & l & l & r & r \\ l & l & l & l & r \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ l & 1 & 0 & 0 & 0 \\ l & l & 1 & 0 & 0 \\ l & l & l & 1 & 0 \\ l & l & l & l & 1 \end{pmatrix} + \begin{pmatrix} r & r & r & r & r \\ 0 & r & r & r & r \\ 0 & 0 & r & r & r \\ 0 & 0 & 0 & r & r \\ 0 & 0 & 0 & 0 & r \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

b) Define the term *Forward Elimination* and the term *Backward Elimination* mathematically.
Solving $Ax = b$ via LR decomposition:

$$Ax = b \Leftrightarrow L(Ux) = b \Leftrightarrow Ly = b, Ux = y.$$

Forward elimination: $Ly = b \Leftrightarrow y = L^{-1}b$
Backward elimination: $Ux = y \Leftrightarrow x = U^{-1}y$

c) Given is

$$M = \begin{pmatrix} 2 & 2 & 1 \\ 4 & 2 & 3 \\ 2 & 2 & 2 \end{pmatrix}$$

Compute the LU decomposition $M = L \cdot U$.
Performing the two row operations $M_{2,:} = M_{2,:} - 2M_{1,:}$ and $M_{3,:} = M_{3,:} - 1 \cdot M_{1,:}$ yields the solution:

$$M = L \cdot U = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 2 & 1 \\ 0 & -2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

d) Compute the LU decomposition using a pivot search (move the entry with the largest absolute value to the pivot position).
First performing row permutations on M:

$$M = \begin{pmatrix} 4 & 2 & 3 \\ 2 & 2 & 1 \\ 2 & 2 & 2 \end{pmatrix}.$$

Via row operations $M_{2,:} = M_{2,:} - \frac{1}{2}M_{1,:}$, $M_{3,:} = M_{3,:} - \frac{1}{2}M_{1,:}$, and $M_{3,:} = M_{3,:} - M_{2,:}$ we receive:

$$M = L \cdot U = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{2} & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 2 & 3 \\ 0 & 1 & -\frac{1}{2} \\ 0 & 0 & 1 \end{pmatrix}$$

2 Collective Operations

a) Define the MPI term *collective operation*.
Collective communication is defined as communication that involves a group or groups of processes. Collective operations are everything besides Point-To-Point or One-To-One operations: One-To-All, All-To-One, or All-To-All operations.

b) Are there any non-blocking collective operations?
In the new MPI standards (since version 3) there are non-blocking collectives. Examples: MPI_IBcast, MPI_IAllreduce, MPI_IReduce, ...

c) Give examples for this three different types of collective operations:

Collective Computations	...
Data movement	...
Synchronisation	...

	All-To-All	All-To-One	One-To-All
Collective computations	Allreduce	Reduce	
Data movement	Allgather	Gather	Broadcast, Scatter
Synchronization	Barrier		

d) The `MPI_Reduce` command supports lots of predefined computations. List and explain them. What is the difference to `MPI_Allreduce`?
Syntax: `MPI_Reduce(sendbuf, recvbuf, count, datatype, op, root, comm)`.

The operation `op` either equals a built-in or a user-defined operator.

Built-in operators are:

Operator	Meaning
`MPI_MAX`	maximum
`MPI_MIN`	minimum
`MPI_SUM`	sum
`MPI_PROD`	product
`MPI_LAND`	logical and
`MPI_BAND`	bit-wise and
`MPI_LOR`	logical or
`MPI_BOR`	bit-wise or
`MPI_LXOR`	logical exclusive or (xor)
`MPI_BXOR`	bit-wise exclusive or (xor)
`MPI_MAXLOC`	max value and location
`MPI_MINLOC`	min value and location

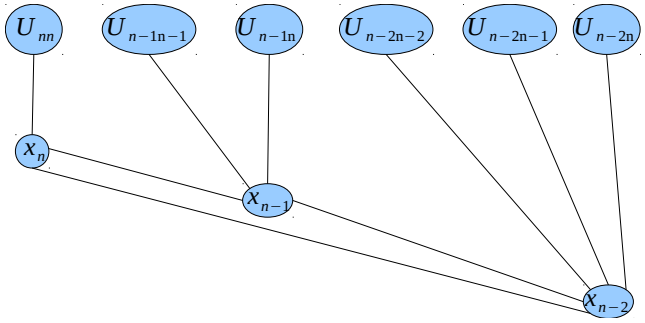
`Allreduce` stores the result on every node = reduce followed by broadcast operation.

3 Parallel LU Decomposition

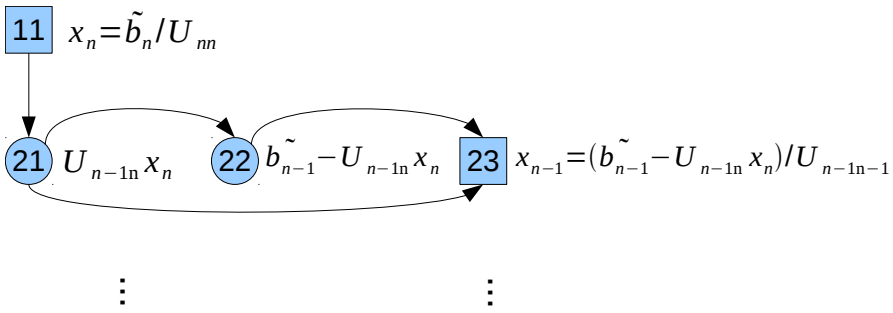
a) Give the data dependency graph for the first two equations of the backward elimination. Is there a parallel algorithm to compute the backward elimination?
First two eliminations:

1. $U_{nn}x_n = (L^{-1}b)_n \Leftrightarrow x_n = \frac{1}{U_{nn}}(L^{-1}b)_n$
2. $x_{n-1} = \frac{1}{U_{n-1n-1}}((L^{-1}b)_{n-1} - U_{n-1n}x_n)$
3. ...

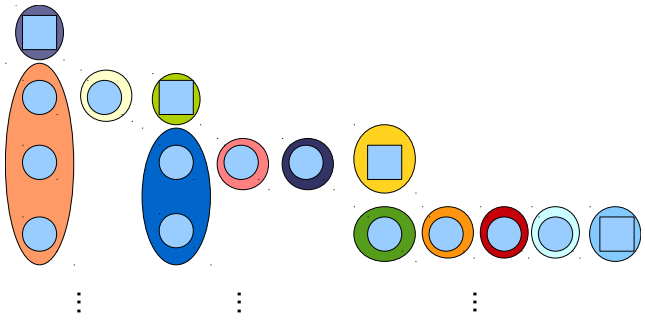
Dependency scetch for backward elimination for x and U:



Dependency graph for first two equations:



There is no parallel algorithm to compute the backward elimination. But with reasonable colouring you can compute the backward elimination with parallelism. Compared to the lecture, where $a_{ii} = 1$, it is possible to compute the backward elimination in $2n - 1$ time steps. In general, assuming no parallelization is performed along the "rows" of the dependency graph, the dependency graph has the following scetch:



For an n-by-n matrix it is possible to compute the backward elimination recursively via

$$steps(1) = 1$$
$$steps(n) = steps(n-1) + n + 1$$

where n is the matrix size.

b) Divide the matrix into blocks of rows. Scatter the blocks among the nodes of a parallel machine. Give a parallel algorithm. What collective operations are used? How many nodes are idle troughout the computation?
Short pseudocode (similar to right looking GE):

- `Split up` A, L, U `into chunks (blocks) of rows.`
- `Broadcast` a_{kk} `to all nodes. Afterwards, the computation of the` l_{ik} `can be done in parallel on each node.`
- `Update of local rows on each node can be done in parallel if row elements (rows) of other nodes were received.`

As collective operations One-To-All operations (Broadcast) are used. Throughout the computation more and more nodes become idle → poor load balancing.

See sourcecode to corresponding tutorial on webpage.

c) Divide the matrix into blocks of columns. Scatter the blocks among the nodes of a parallel machine. Give a parallel algorithm. What collective operations are used? How many nodes are idle throughout the computation?
Left to yourself for training (do not give up ;-)).

d) Assign the columns of the matrix alternating to the different machines. Reformulate a parallel algorithm in words (or pseudocode). What about the load balancing and load difference, respectively?
Alternating assignment of columns to processors. Short pseudocode

```
Scatter A among processors.
for k=1 to n-1
     Compute l_ik of column L_:,1
     Broadcast L_:,1
     Update all entries on all nodes (parallel part).
```

The computation of L is performed sequentially but the update of A is done in parallel. At most one column load difference between two nodes.