Technische Universität München
Institut für Informatik
Prof. Dr. Thomas Huckle
Michael Lieb, M. Sc.

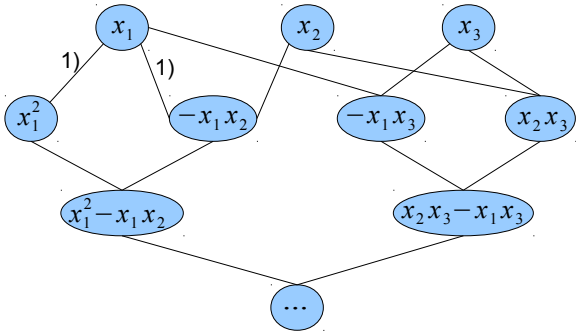
WT 2013/2014

Parallel Numerics

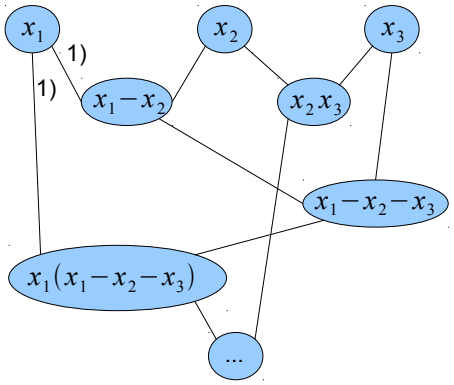
Exercise 4: Matrix–Matrix Operations & P2P Communication III

1) DAGs for Function Evaluation

Given is $f(x_1, x_2, x_3, x_4) = x_1^2 + x_2x_3 - x_1x_3 - x_1x_2 = x_2x_3 + x_1(x_1 - x_3 - x_2)$.
Give the DAGs for both evaluation schemes in parallel and compare the number of required time steps.



(a) *DAG for* $x_1^2 + x_2x_3 - x_1x_3 - x_1x_2$



(b) *DAG for* $x_2x_3 + x_1(x_1 - x_3 - x_2)$

Number of required time steps for b) is 4. If the negation is not considered as an operation for a), the number of time steps is 3, otherwise 4. Shared variables e.g. for 1) in a) and b).

2) Amdahl's Law

Given is an algorithm working on a fixed amount of data. f is the fraction of a calculation that is sequential and, thus, cannot benefit from parallelisation. The algorithm has execution time t_1.

i) What is the execution time t_p, if exactly this problem is ported to a parallel machine with p processors.
$t_p = ft_1 + (1-f)t_1 \cdot \frac{1}{p}$

ii) What is the speedup?
$S = \frac{t_1}{t_p} = \frac{t_1}{ft_1+(1-f)t_1\cdot\frac{1}{p}} = \frac{1}{f+\frac{1}{p}(1-f)}$

iii) What is the efficiency?
$E = \frac{S}{p} = \frac{1}{pf+1-f}$

iv) What happens if the number of processors is increased ($\lim_{p \mapsto \infty}$)?
If the number of processors is increased to ∞, the efficiency becomes 0.
$\lim_{p \to \infty} E = \lim_{p \to \infty} \frac{1}{pf + 1 - f} = 0$
If the number of operations is bounded (see computation Tutorial 3) more and more nodes become idle.

3) Gustafson's Law (Gustafson-Barsis' Law)

You are given a parallel algorithm running on p nodes that needs the time t_p to finish. Again, f is the fraction of a calculation that is sequential and, thus, cannot benefit from parallelisation.

i) Derive the time t_1, one processor would need to solve this problem.
$t_1 = t_p f + p(1 - f)t_p$

ii) What is the speedup?
$S = \frac{t_1}{t_p} = f + p(1 - f)$

iii) What is the efficiency?
$E = \frac{S}{p} = \frac{1}{p}f + 1 - f$

iv) What happens, if the number of processors is increased ($\lim_{p \mapsto \infty}$)?
$\lim_{p \to \infty} E = 1 - f$

v) Compare these formulas to Amdahl's law. In today's problems, often not the execution but the pure amount of data is the limiting factor for computations. Thus, an algorithm is parallelised because one wants to compute bigger problems, and a speedup is only a (nice) side–effect. What speedup law is the appropriate one for this scenario?
Today one wants to make many problems as big as one node can handle → Gustafson's law is appropriate.

4) P2P Communication III: Synchronised Operations

Given is the following pseudocode fragment:

```
# node 0:
int a = 0;
send a to node 1
a = 1;
send a to node 1

#node 1:
int b;
receive b from node 0
printf("%i", b);
```

```
receive b from node 0
printf("%i", b);
```

i) Implement the send and receive operations using blocking and non–blocking operations. What might be the difference in the results?

```
//blocking pseudocode
//# node 0:
int a = 0;
MPI_Send(&a,1,MPI_INT,1,...);
...
//# node 1:
MPI_Recv(&b,1,MPI_INT,0,...);
```

Output for blocking operations will be:

0
1

```
//non-blocking pseudocode
//# node 0:
MPI_Request request;
int a = 0;
MPI_Isend(&a,1,MPI_INT,1,0,MPI_COMM_WORLD,&request);
...
a = 1;
...
```

Output for non-blocking operations will be:

0 $\perp$ $\perp$ *0* ...
1 $\perp$ *0* *0* ...

ii) Define the MPI term *synchronised*.
A send can be started whether or not a matching receive was posted. However, the send will complete only if a matching receive is posted and the receive operation has started to receive the message sent by the synchronous send (handshake semantics). Thus, the completion of a synchronous send not only indicates that the send buffer can be reused, but it also indicates that the receiver has started executing the matching receive.

iii) Implement the send and receive operations using synchronised non–blocking operations. Insert a wait operation directly after the next integer assignment on node 0. What might be the difference in the results?

```
//Pseudocode
int a = 0;
MPI_Issend(&a,...);
a = 1;
MPI_Wait(...);
```

It is sure that the message has been sent and that the receive has been started, but, it is not sure what the message contained.

iv) Implement the send and receive operations using synchronised blocking operations. What might be the drawback?

Advantage of synchronous and blocking mode: Deterministic communication behaviour and guarantee that message has been received.
Disadvantage: slow.

v) Make yourself familiar with the MPI terms *buffered send* and *ready send*.
Buffered send: If buffered send is executed and no matching receive is posted, MPI must buffer the outgoing message, so as to allow the send call to complete. Additional buffer (memory) assigend to MPI.
Ready send: Send may be started only if the matching receive is already posted. Receiver has to tell sender that everything is prepared to receive information (e.g. streaming).

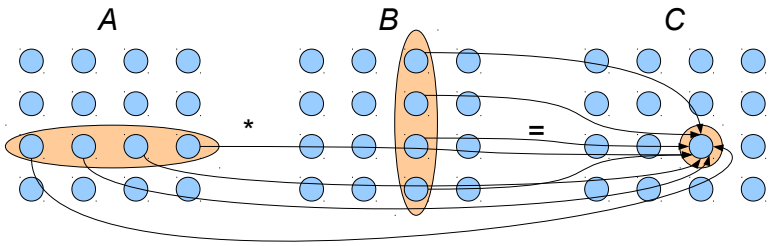
5) Matrix–Matrix Multiplication

The product $A \cdot B$ of two $N \times N$-matrices A and B is calculated as follows:

$$C = A \cdot B \qquad \text{where} \qquad (C)_{ij} = \sum_{k=1}^{N} (A)_{ik}(B)_{kj}.$$

A program that calculates this product has to use three independent loops (over i, j, k). The execution speed of the program depends on the arrangement of these loops.

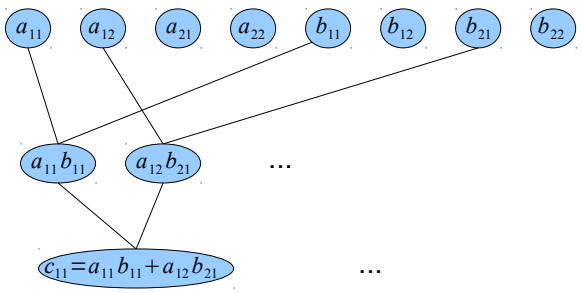
i) Give a scetch of the data dependency graph for one element of this problem.
A scetch of the data dependency graph is shown for one single element of C for a 4-by-4 matrix:



For a 2-by-2 matrix-matrix multiplication of the form

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

the direct acyclic graph has the form



ii) Assume N is a multiple of $\sqrt{p}$. All the three matrices are split up into p quadratic submatrices. Every node has to compute one submatrix of the result matrix C. Illustrate this fact!
$A \cdot B = C$ is computed by p nodes. E.g. node #0 computes block C_{11}, node #1 computes block C_{12}, etc.:

$$\left(\begin{array}{c|c|c} A_{11} & A_{12} & A_{13} \\ \hline A_{21} & A_{22} & A_{23} \\ \hline A_{31} & A_{32} & A_{33} \end{array}\right) \cdot \left(\begin{array}{c|c|c} B_{11} & B_{12} & B_{13} \\ \hline B_{21} & B_{22} & B_{23} \\ \hline B_{31} & B_{32} & B_{33} \end{array}\right) = \left(\begin{array}{c|c|c} \#0 & \#1 & \#2 \\ \hline \#3 & \#4 & \#5 \\ \hline \#6 & \#7 & \#8 \end{array}\right)$$

iii) Pick out three different nodes. What parts of A and B are required by them to compute the result?
First three nodes have to compute following blocks:
node #0 : $C_{11} = A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{31}$
node #1 : $C_{12} = A_{11}B_{12} + A_{12}B_{22} + A_{13}B_{32}$
node #2 : $C_{13} = A_{11}B_{13} + A_{12}B_{23} + A_{13}B_{33}$

iv) Have a look at the node computing the upper left submatrix of C. Interprete the multiplication formular $(C)_{ij} = \sum_{k=1}^{\sqrt{p}} (A)_{ik}(B)_{kj}$ block–wise. Let's assume every summand of the result is computed in one time step. What different parts of A and B are required subsequently? How many time steps are required?
See iii) node #0 : $C_{11} = A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{31}$. *There are $\sqrt{p}$ time steps required.*

v) Every node is allowed to communicate with the node that computes the left, right, top or bottom submatrix of C only. Assume this cartesian topology is cyclic. Furthermore, every node is allowed to hold only one submatrix of A and B at one time. Derive a communication scheme.
Use Cannon's algorithm for matrix–matrix multiplication. A cyclic rotation scheme is used (pseudocode):

- *Determine blocks*
- *Scatter blocks on processors*
- *Loop over blocks:*
  - *Compute part of sum*
  - *Shift A_{local} to left*
  - *Shift B_{local} up*
- *Gather blocks*

See sourcecode to corresponding tutorial on webpage.