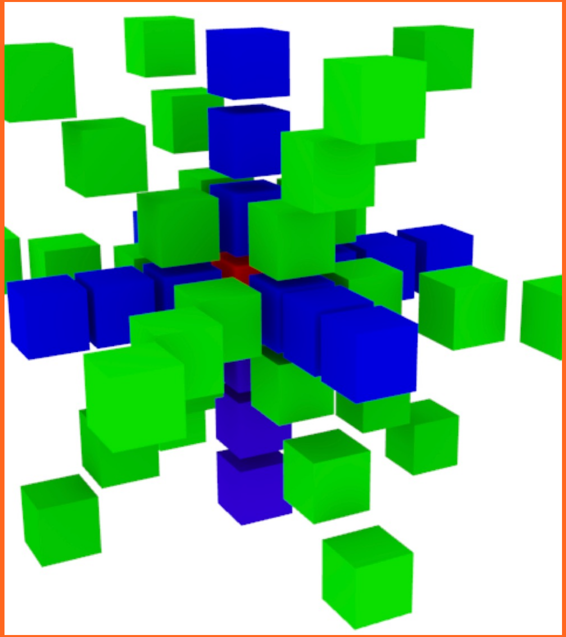
# CS-E4690 – Programming parallel supercomputers D

4th lecture



**Maarit Korpi-Lagg**

**maarit.korpi-lagg@aalto.fi**

**14.11.2023**

Aalto University
School of Science

# Lecture 4

Collectives (finalizing basic MPI)
One-sided communication
(entering the advanced domain)

- **Course practicalities: 5 min**
- **Key concepts recap (40 min) – old and new**
- **Break (**max 15 mins**)**
- **Example codes cntd.** (20 mins)
- **Exercise sheet tasks tips (**20 mins**)**
- **Wrap up (**poll & feedback; 5 mins**)**

# Break-down of learning objectives

## Lecture1

Introduction to the current HPC landscape

Understanding how this course fits into that

Establishing understanding of the learning outcomes, specifically answering the question: “What are programming during this course?”

## Lecture2

Learning basic definitions and taxonomies

Understanding the importance of the “network”

Learning basic performance models

Understanding the concept of a well-performing software in large-scale computing.

## Lecture3

Becoming knowledgeable of the modern landscape of distributed memory programming

Understanding why in this course we will concentrate on low-level programming models

Getting acquainted with MPI: basics and synchronous and asynchronous point-to-point communication

# Break-down of learning objectives

**Lecture4**

Learning more about MPI:

One-sided point-to-point communications

Collective communications

**Lecture5**

Programming MP hybrid architectures

Becoming knowledgeable of the spectrum of options

Understanding efficiency issues

**Lecture6**

Programming hybrid architectures with accelerators

Acquiring knowledge of CUDA-MPI programming model

# Repetition of key concepts - old and new

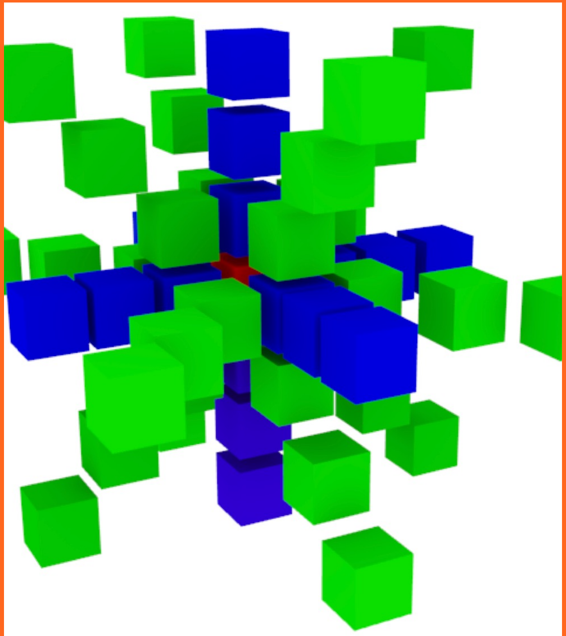
Aim: to explain the key concept in short

Discuss 1-5 mins in row-wise groups

Randomly selected group(s) present(s) and another one comment(s)

Model answer

Feedback with post-its

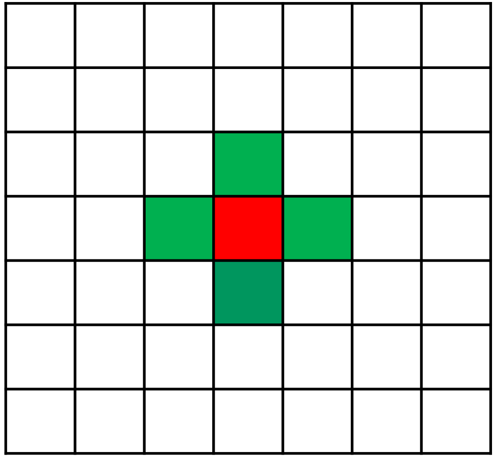


Aalto University
School of Science

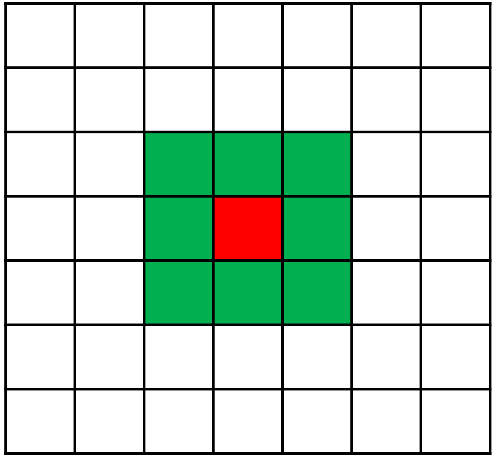
# Iterative stencil loop

# Recurring update pattern of array elements based on their neighbors.

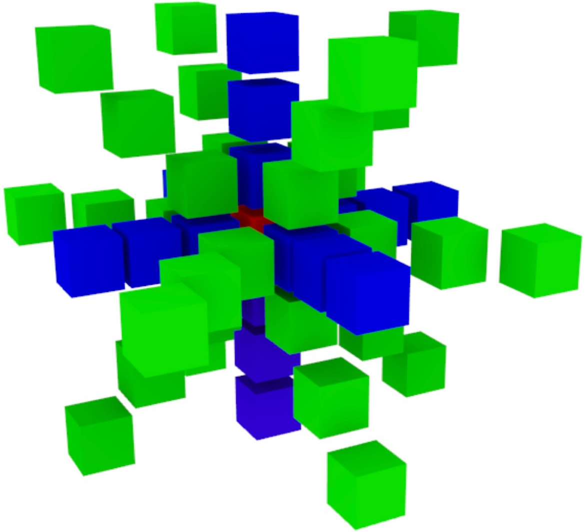
# Iterative stencil loop



?



?



?

# Draw the stencil of Sheet 3 ex. 2

$$\frac{\partial c}{\partial x}(x_i, y_j, t_n) \approx \frac{+3c^n_{i,j} - 4c^n_{i-1,j} + c^n_{i-2,j}}{2\Delta x} \qquad \text{for} \qquad v_x > 0$$

$$\frac{\partial c}{\partial x}(x_i, y_j, t_n) \approx \frac{-c^n_{i+2,j} + 4c^n_{i+1,j} - 3c^n_{i,j}}{2\Delta x} \qquad \text{for} \qquad v_x < 0,$$

… and the same for y velocity in j direction.

# Two-sided p2p communication; Examples belonging/not belonging (at least one per group)?

# Two parties – corresponding sender and receiver

MPI_Irecv

MPI_Send

MPI_Ssend

MPI_Isend

MPI_Bsend

MPI_Recv

MPI_Sendrecv

…

MPI_Put

MPI_Get

MPI_Bcast

MPI_Scatter …

MPI_Accumulate

MPI_Get_accumulate

# What does ”MPI messages are non-overtaking” mean?

**If a sender sends two messages in succession to the same destination, and both match the same receive, then this operation cannot receive the second message if the first one is still pending.**

# Does this apply to one-sided communication?

**It depends. Many MPI_Put and MPI_Get mixed within an epoch do not guarantee the order and can lead to race conditions.**

# Are there ways to avoid race conditions in one-sided communication?

Yes!

Remember to synchronize

OR

Use MPI_Accumulate with MPI_REPLACE and MPI_Get_accumulate with MPI_NO_OP which implement atomic operations for the same target – origin pairs.

# What is the difference with rooted and all-to-all collectives?

Results of reductions are collected to ROOT’s receive buffer versus to all ranks’ receive buffers

**MPI_Reduce vs. MPI_Allreduce**

# Are user defined ops allowed in RMA reductions?

# Not yet.

| MPI type | meaning | applies to\ |
|---|---|---|
| MPI_MAX | maximum | integer, floating point |
| MPI_MIN | minimum | |
| MPI_SUM | sum | integer, floating point, complex, multilanguage types |
| MPI_REPLACE | overwrite | |
| MPI_NO_OP | no change | |
| MPI_PROD | product | |
| MPI_LAND | logical and | C integer, logical |
| MPI_LOR | logical or | |
| MPI_LXOR | logical xor | |
| MPI_BAND | bitwise and | integer, byte, multilanguage types |
| MPI_BOR | bitwise or | |
| MPI_BXOR | bitwise xor | |
| MPI_MAXLOC | max value and location | MPI_DOUBLE_INT and such |
| MPI_MINLOC | min value and location | |

# Is there a way to make blocking two-sided ops safe?

**Not fully. Using MPI_Sendrecv, MPI_Ssend and MPI_Bsend may help to write safe code.**

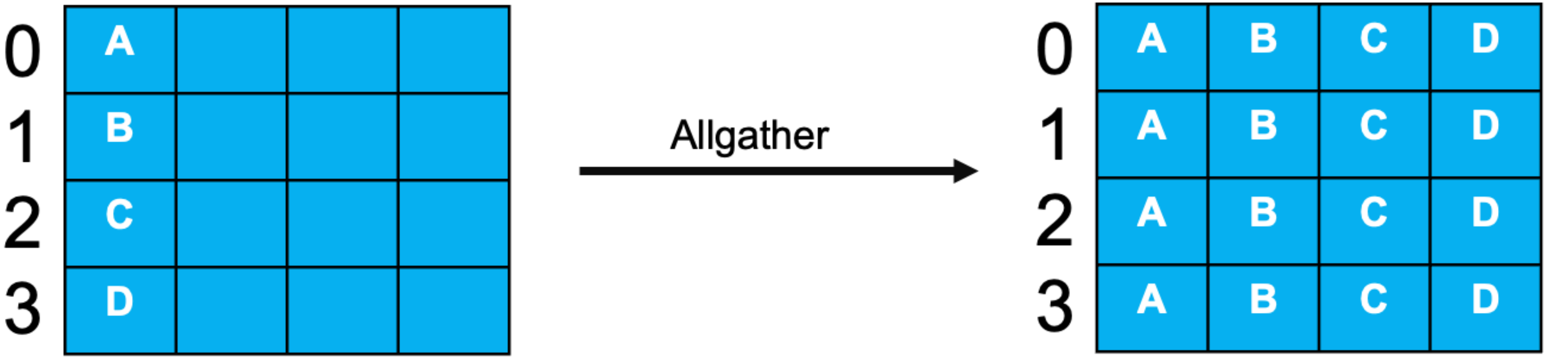
# Will blocking two-sided ops give you the optimum performance?

**No, as these functions do not allow for concurrency in computation and communication.**

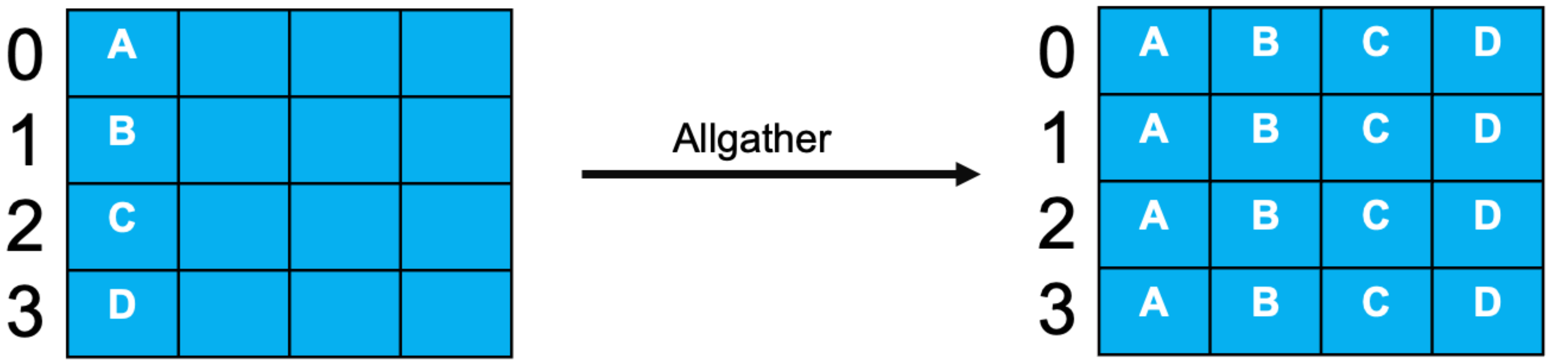
# Will non-blocking two-sided ops give you the optimum performance?

**It depends: these functions allow for concurrency in computation and communication, but still require some implicit synchronization, buffers, and activity from both sender and receiver.**

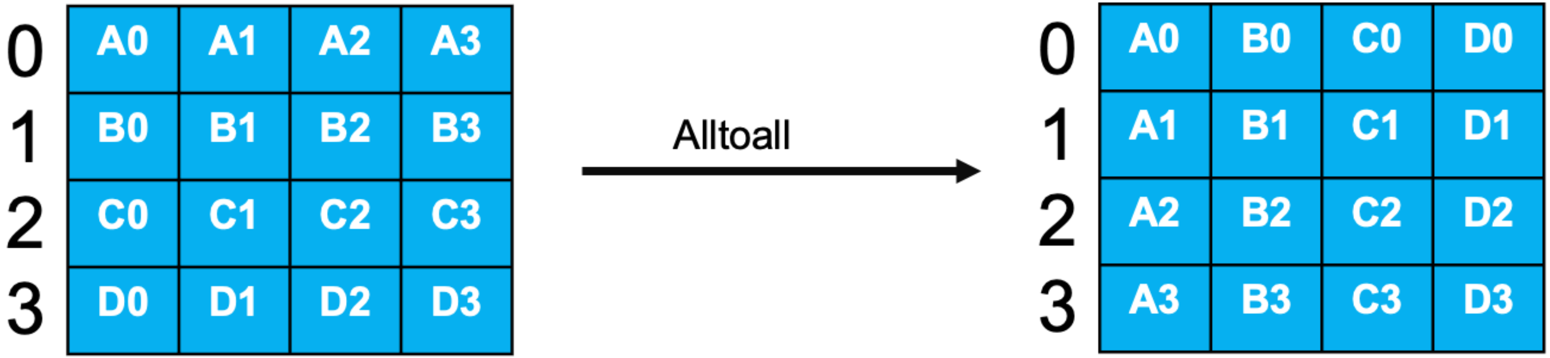
# How could you replace Allgather with simpler MPI collective functions?



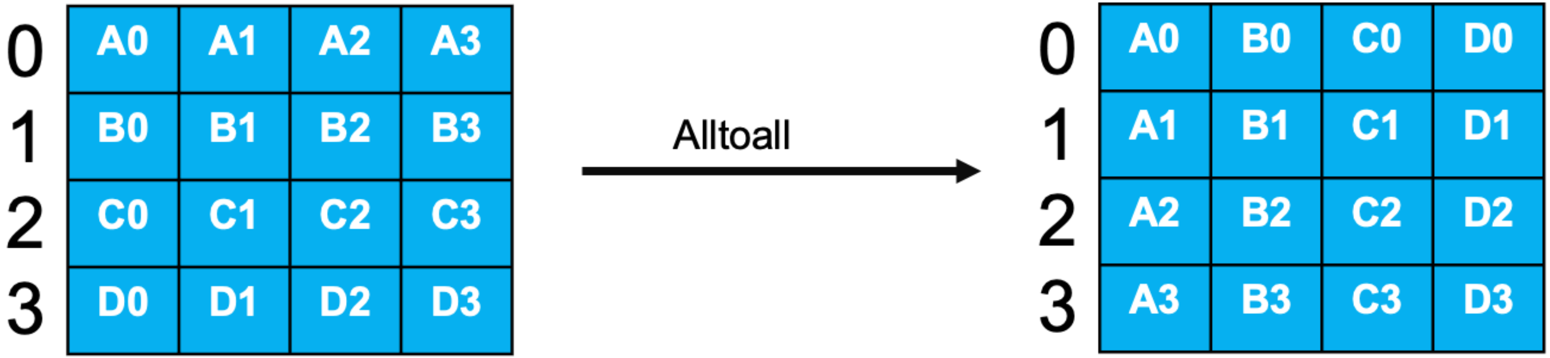
# First Gather, and then Broadcast

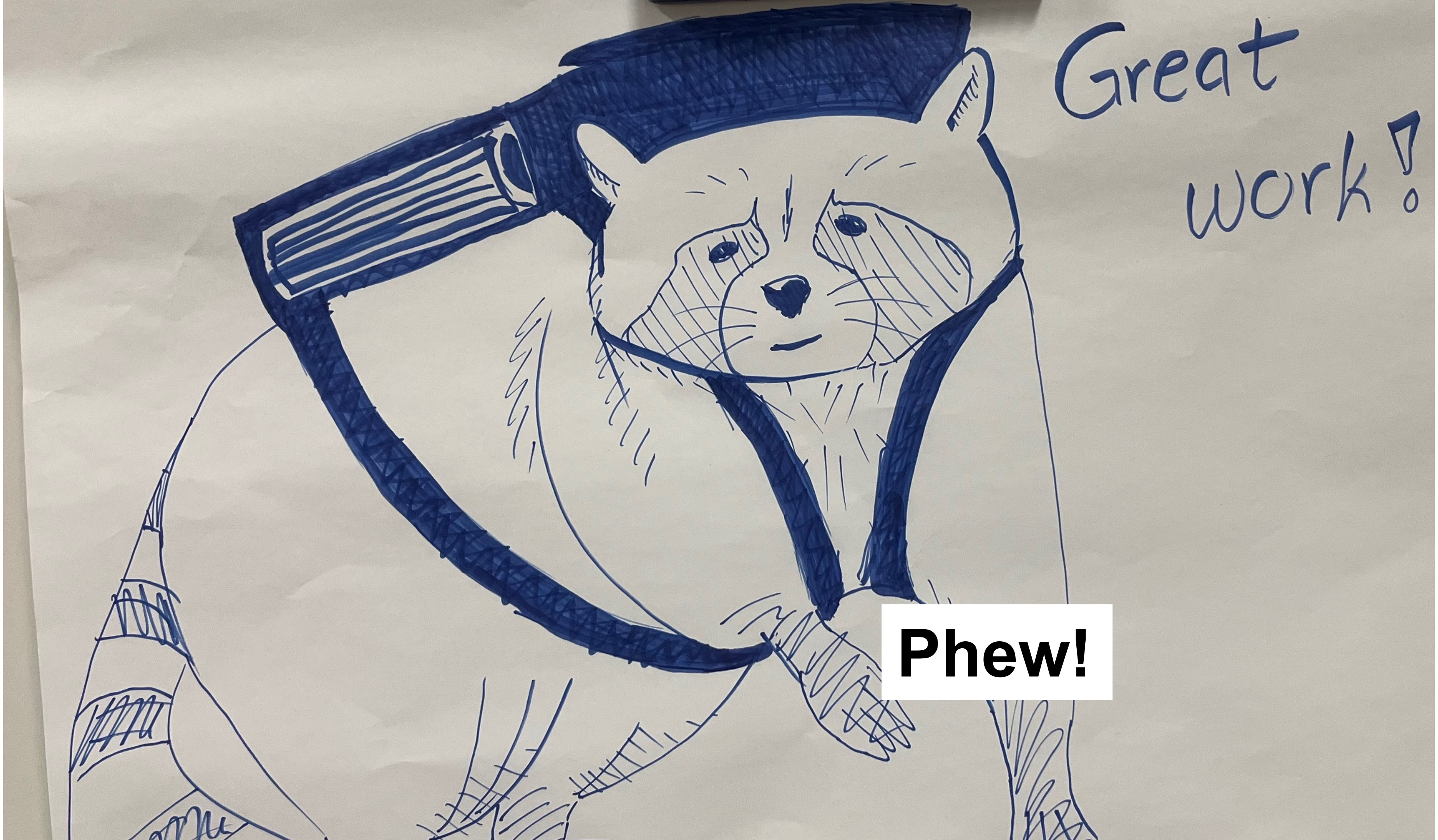


# What, in terms of matrix operations, is done here?



# Transpose


0
1
2
3
A0 A1 A2 A3
B0 B1 B2 B3
C0 C1 C2 C3
D0 D1 D2 D3
Alltoall
0
1
2
3
A0 B0 C0 D0
A1 B1 C1 D1
A2 B2 C2 D2
A3 B3 C3 D3


Great work!
Phew!