FARABI UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE



Lecture 5.
Parallel Algorithms for Matrix Operations

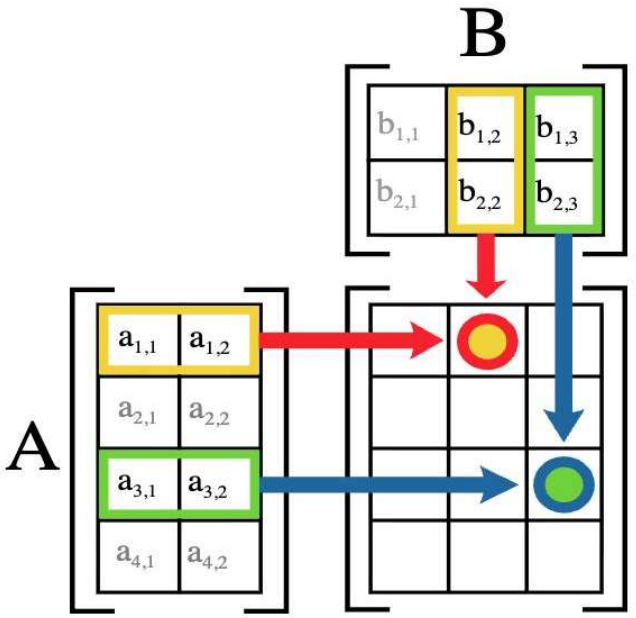
Dr. Bazargul Matkerim, Professor-Researcher

Outline

1. **Matrix-vector multiplication**
   columnwise decomposition, MPI program

2. **Rowwise block-striped algorithm**
   broadcast, scatter, dot products

3. **Matrix-matrix multiplication**
   scatter, broadcast, gather



Source: Matrix multiplication, commons.wikimedia.org (CC BY-SA 3.0)

Matrix-Vector Multiplication

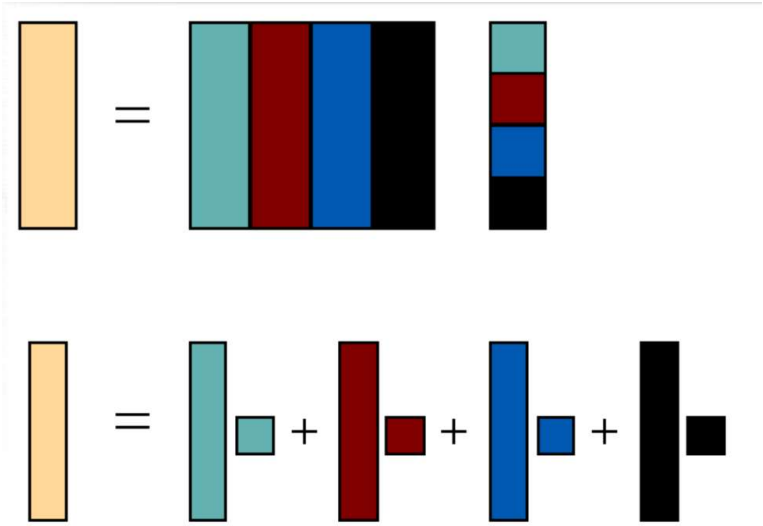
Product **A = B · C**

Decomposed into independent computations:
a column of B times a scalar element of C

Computations are **independent**
no communication, which usually reduces performance

$$a_i = \sum_{j=0}^{n-1} b_{ij} c_j$$

$$A = \sum_{j=0}^{n-1} c_j \cdot B_{*,j}$$



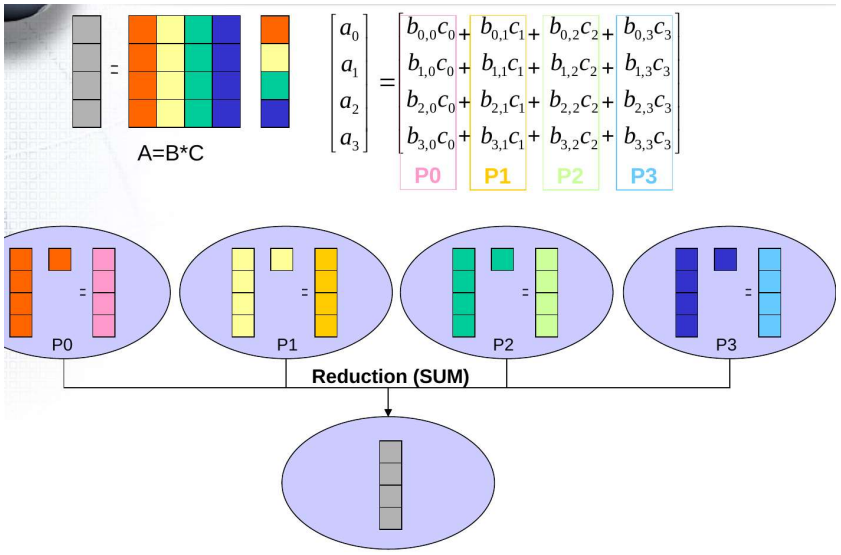
Source: Lecture 10 material, slide 5

Columnwise Algorithm with MPI

Columns of B and elements of C: **scatter**
each process gets Bpart and Cpart

Each process computes Apart = Cpart · Bpart

Parts of A are added: **reduction (SUM)**
result on the root process



Source: Lecture 10 material, slide 6

MPI Program: Data Distribution

```
/* Scatter matrix B by rows */
MPI_Scatter(B, NCOLS, MPI_FLOAT, Bpart, NCOLS, MPI_FLOAT,
            root, MPI_COMM_WORLD);
/* Scatter vector C */
MPI_Scatter(C, 1, MPI_FLOAT, Cpart, 1, MPI_FLOAT,
            root, MPI_COMM_WORLD);
/* vector-scalar multiplication */
for (j = 0; j < NCOLS; j++)
    Apart[j] = Cpart[0] * Bpart[j];
/* Reduce to vector A */
MPI_Reduce(Apart, A, NCOLS, MPI_FLOAT, MPI_SUM,
           root, MPI_COMM_WORLD);
```

Source: Lecture 10 material, slide 10

- MPI_Scatter: distribute parts
- MPI_Reduce + MPI_SUM: add parts
- MPI_Barrier: wait for I/O
- Check: compare with the **serial** result

C Stores Arrays by Rows

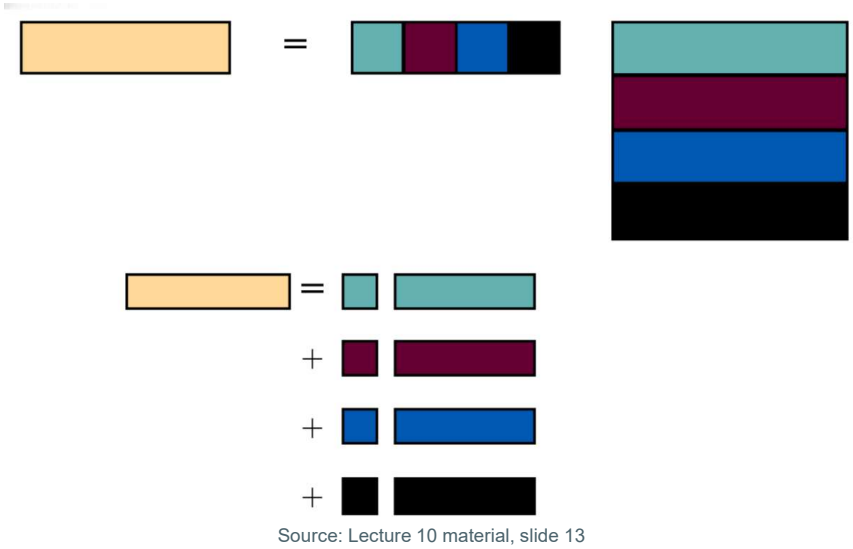
C: arrays stored by **rows**

Fortran: arrays stored by **columns**

A direct translation to Fortran would fail

Solutions:

- derived datatypes
- decompose into scalar-row computations



Source: Lecture 10 material, slide 13

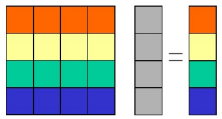
Rowwise Block-Striped Multiplication

Broadcast vector C to all processes

Scatter the rows of B

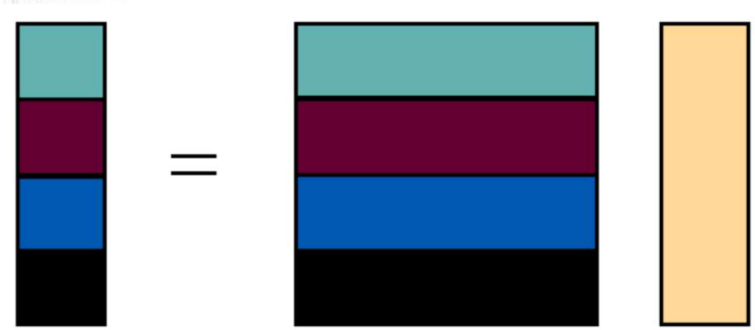
Each process: **dot product** row · C
gives elements of A

$$a_i = \sum_{j=0}^{n-1} b_{ij} c_j$$



B*C=A

Source: Lecture 10 material, slide 14



Source: Lecture 10 material, slide 16

Collective Routines Used

Routine	What it does
MPI_Barrier	synchronizes all processes
MPI_Bcast	sends the same data from root to all
MPI_Scatter	sends a different part to each process
MPI_Gather	collects parts on the root
MPI_Reduce	combines parts, e.g. with MPI_SUM

All routines are called by **all processes** of the communicator

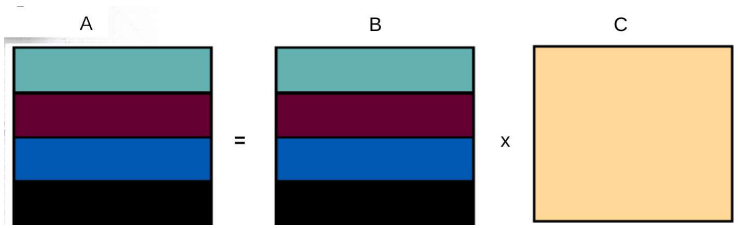
Matrix-Matrix Multiplication

Naive decomposition of **A = B · C**

4×4 example: **four independent** vector-matrix multiplications

Rows of B: one part for each process

Matrix C (yellow) is **broadcast** to all



Source: Lecture 10 material, slide 19

$$a_{ij} = \sum_{k=0}^{n-1} b_{ik} c_{kj}$$

Matrix-Matrix Algorithm

1. **Scatter** the rows of B
2. **Broadcast** the matrix C
3. Multiply: rows of B times columns of C gives the corresponding rows of A
4. **Gather** the rows of A on one process

```
MPI_Scatter(B, rows*N, MPI_FLOAT, Bpart, rows*N,
            MPI_FLOAT, root, MPI_COMM_WORLD);
MPI_Bcast(C, N*N, MPI_FLOAT, root, MPI_COMM_WORLD);
for (i = 0; i < rows; i++)
  for (j = 0; j < N; j++) {
    Apart[i][j] = 0;
    for (k = 0; k < N; k++)
      Apart[i][j] += Bpart[i][k] * C[k][j];
  }
MPI_Gather(Apart, rows*N, MPI_FLOAT, A, rows*N,
           MPI_FLOAT, root, MPI_COMM_WORLD);
```

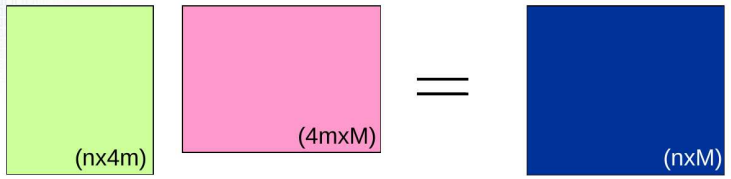
Code sketch of the steps from slide 20 (rows = N / number of processes)

Questions and Homework

Think about:

how to generalize to an (n × 4m) by (4m × M) matrix on 4 processors?

Homework: write the code for the **rowwise** matrix-matrix decomposition in C



Source: Lecture 10 material, slide 17