Parallel programming

MPI Interface

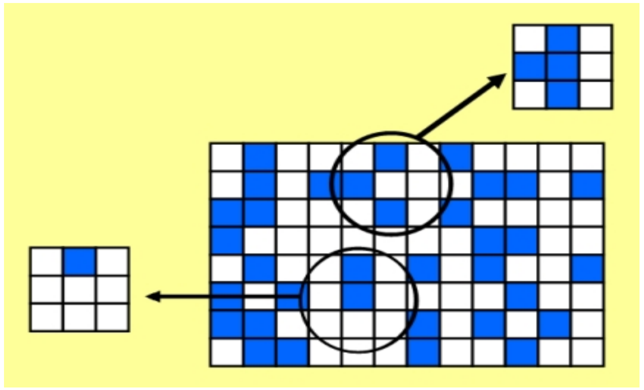
13 - Game of Life

A simple simulation developed by John Conway. In the "Game of Life" the domain is a 2-dimensional array of cells, and each cell can have one of two possible states, usually referred to as "alive" or "dead." The array is usually intialized using random numbers and the system then evolves in time. At each time step, each cell may or may not change its state, based on the number of adjacent alive cells, including diagonals. There are three rules:

- If a cell has three neighbors that are alive, the cell will be alive. If it was already alive, it will remain so, and if it was dead, it will become alive.
- If a cell has two neighbors that are alive, there is no change to the cell. If it was dead, it will remain dead, and if it was alive, it will remain alive.
- In all other cases — the cell will be dead. If it was alive it becomes dead and if it was dead it remains dead.

13 - Game of Life

An example of the changes that occur for two cells in a given time step are shown in Figure below where "live" cells are blue, and "dead" cells are white.



The idea behind the "Game of Life" is that it crudely simulates populations of organisms. The organisms need the support of other organisms to live. If there is not a significant population, i.e., zero or one neighbor, the organism will die. If there are two neighbors, the case is neutral, with no change. If there are three neighbors, the organisms will multiply. With four or more neighbors, there is overcrowding, and the organisms will die.

DERIVED DATATYPES (MPI_Datatype)

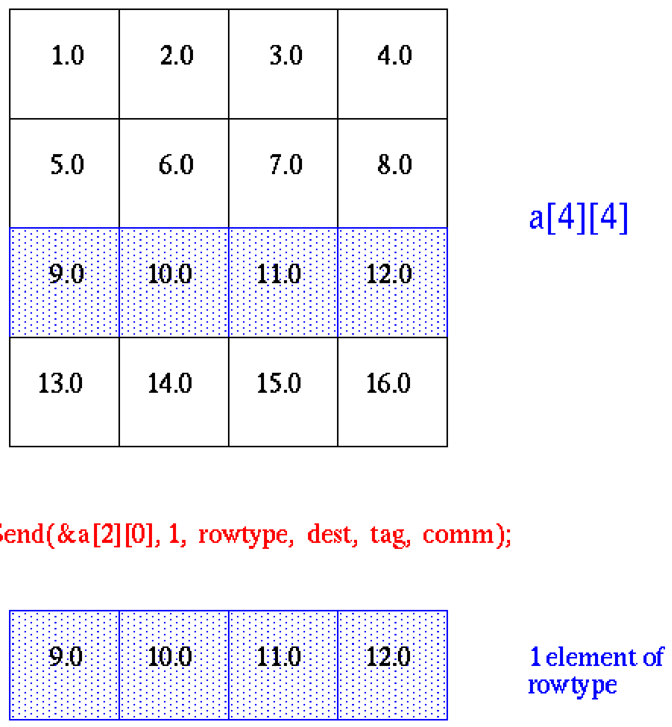
- Derived datatypes:
  - MPI_Type_contiguous
  - MPI_Type_vector
  - MPI_Type_struct

- MPI_Type_commit
- MPI_Type_free
- MPI_Type_extent

MPI_Type_contiguous

- MPI_Type_contiguous (count, oldtype, *newtype)
  - **count** - Number of blocks to be added
  - **oldtype** - Datatype of each element
  - **newtype** - Handle (pointer) for new derived type

```
count = 4;
MPI_Type_contiguous(count, MPI_FLOAT, &rowtype);
```

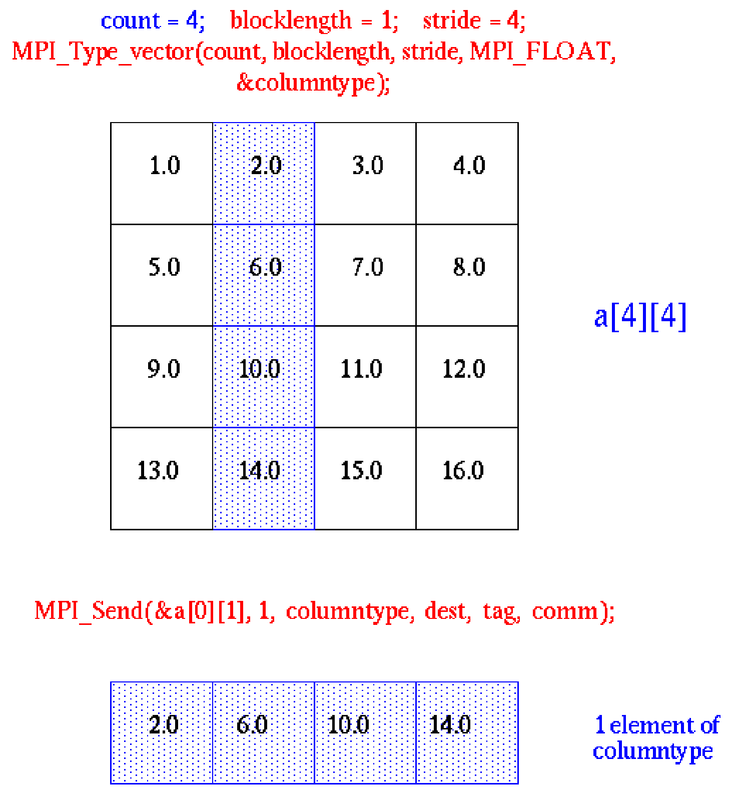


```
MPI_Send(&a[2][0], 1, rowtype, dest, tag, comm);
```

07 - Example - MPI_Type_contigouos

MPI_Type_vector

- MPI_Type_vector (count,blocklength,stride,oldtype,*newtype)
  - **count** - Number of blocks to be added
  - **blocklen** - Number of elements in block
  - **stride** - Number of elements (NOT bytes) between start of each block
  - **oldtype** - Datatype of each element
  - **newtype** - Handle (pointer) for new derived type



count = 4; blocklength = 1; stride = 4;
MPI_Type_vector(count, blocklength, stride, MPI_FLOAT, &columntype);
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
a[4][4]
MPI_Send(&a[0][1], 1, columntype, dest, tag, comm);
2.0
6.0
10.0
14.0
1 element of columntype

08 - Example - MPI_Type_vector

MPI_Type_struct

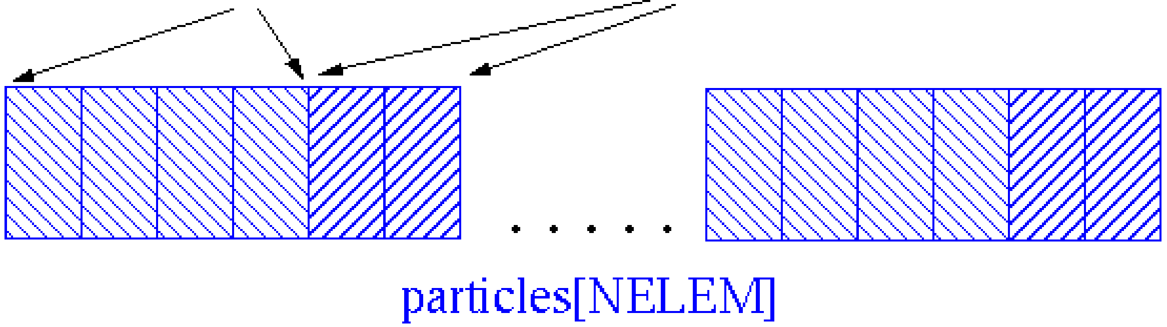
- MPI_Type_struct (count, blocklens[], offsets[], old_types[], *newtype)
  - **count** - Number of blocks to be added
  - **blocklens** - Number of elements in block -- an array of length count
  - **offset** - Number of bytes from start for each block. Displacements (an array of length count) for each block
  - **oldtype** - Datatype of each element
  - **newtype** - Handle (pointer) for new derived type

MPI_Type_struct

```
typedef struct { float x, y, z, velocity;  int  n, type; }  Particle;
Particle  particles[NELEM];

MPI_Type_extent(MPI_FLOAT, &extent);

count = 2;  oldtypes[0] = MPI_FLOAT;     oldtypes[1] = MPI_INT
            offsets[0] = 0;              offsets[1] = 4 * extent;
            blockcounts[0] = 4;          blockcounts[1] = 2;
```



```
MPI_Type_struct(count, blockcounts, offsets, oldtypes, &particletype);

MPI_Send(particles,  NELEM, particletype,  dest,  tag,  comm);
```

Sends entire (NELEM) array of particles, each particle being comprised four floats and two integers.

09 - Example - MPI_Type_struct

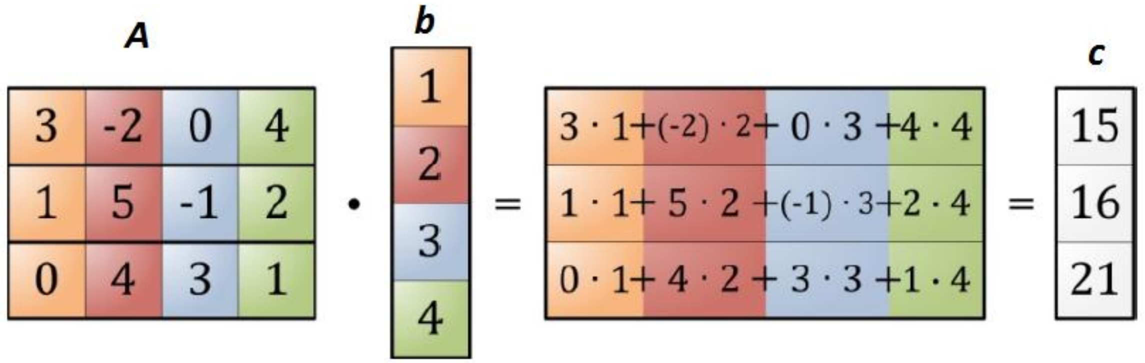
Množenje matrice vektorom

Input : a [0 . .m-1 ,0. .n-1] - matrica dimenzije m x n

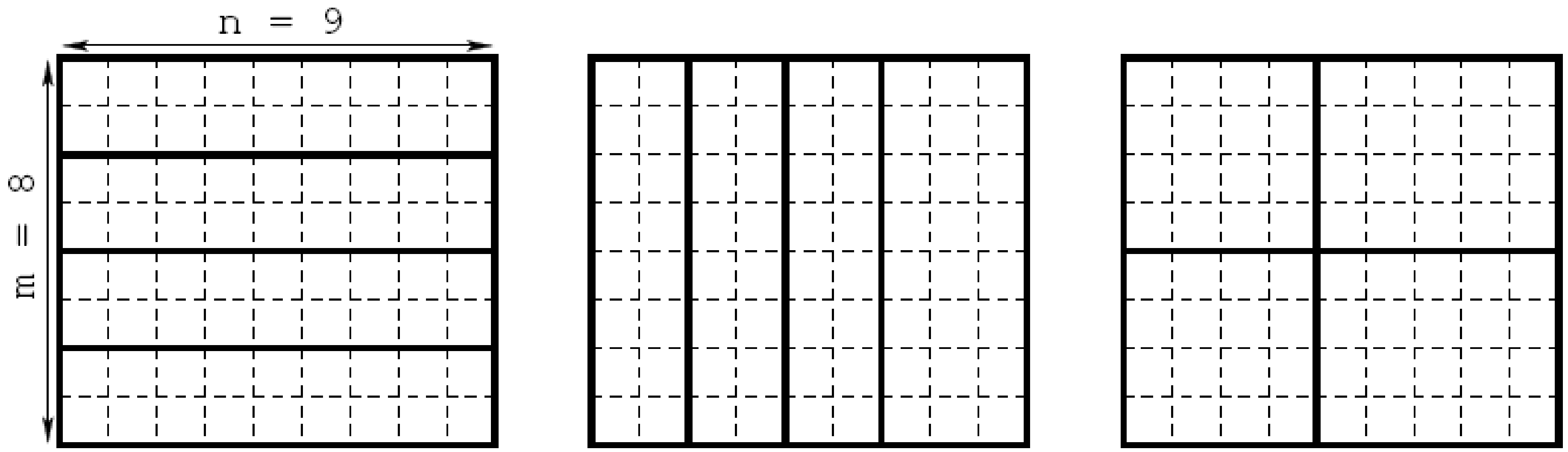
b [0 . . N-1] - vektor dimenzije n x 1

Output : c [0 . .m-1] - vektor dimenzije m x 1

```
for i :=0 to m-1
c[i] :=0
   for j :=0 to n-1
   c[i] := c[i] + a[i, j] x b[j]
   endfor
endfor
```



Množenje matrice vektorom - particionisanje

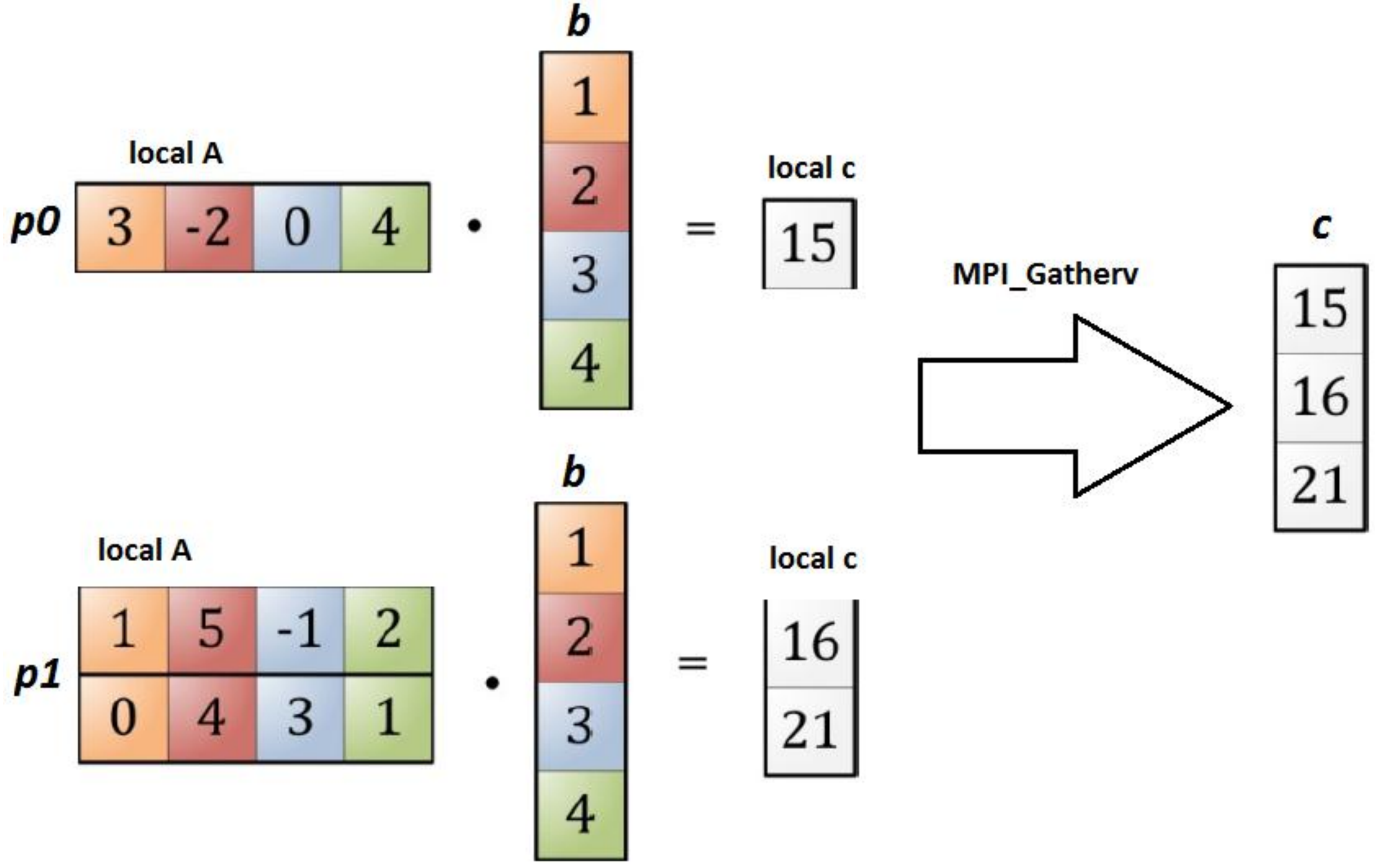


a) Rowwise block–striped decomposition

b) Columnwise block–striped decomposition

c) Checkerboard block decomposition

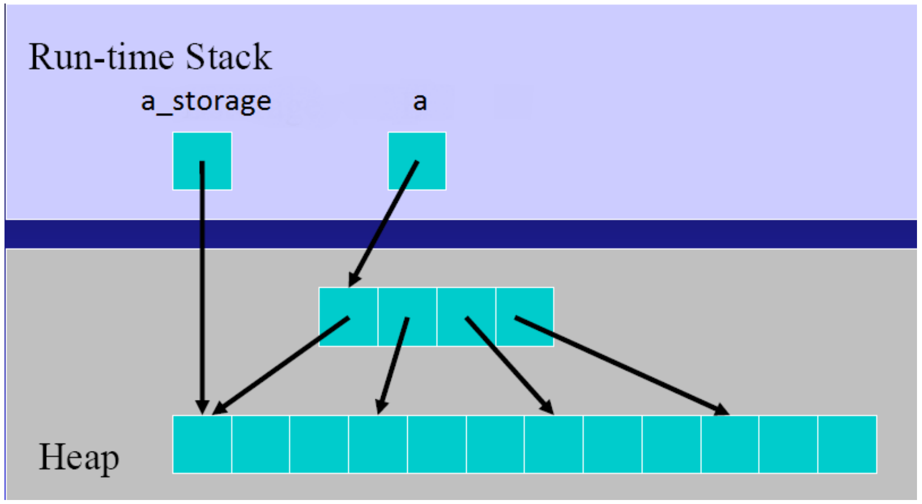
Dekompozicija na blokove redova



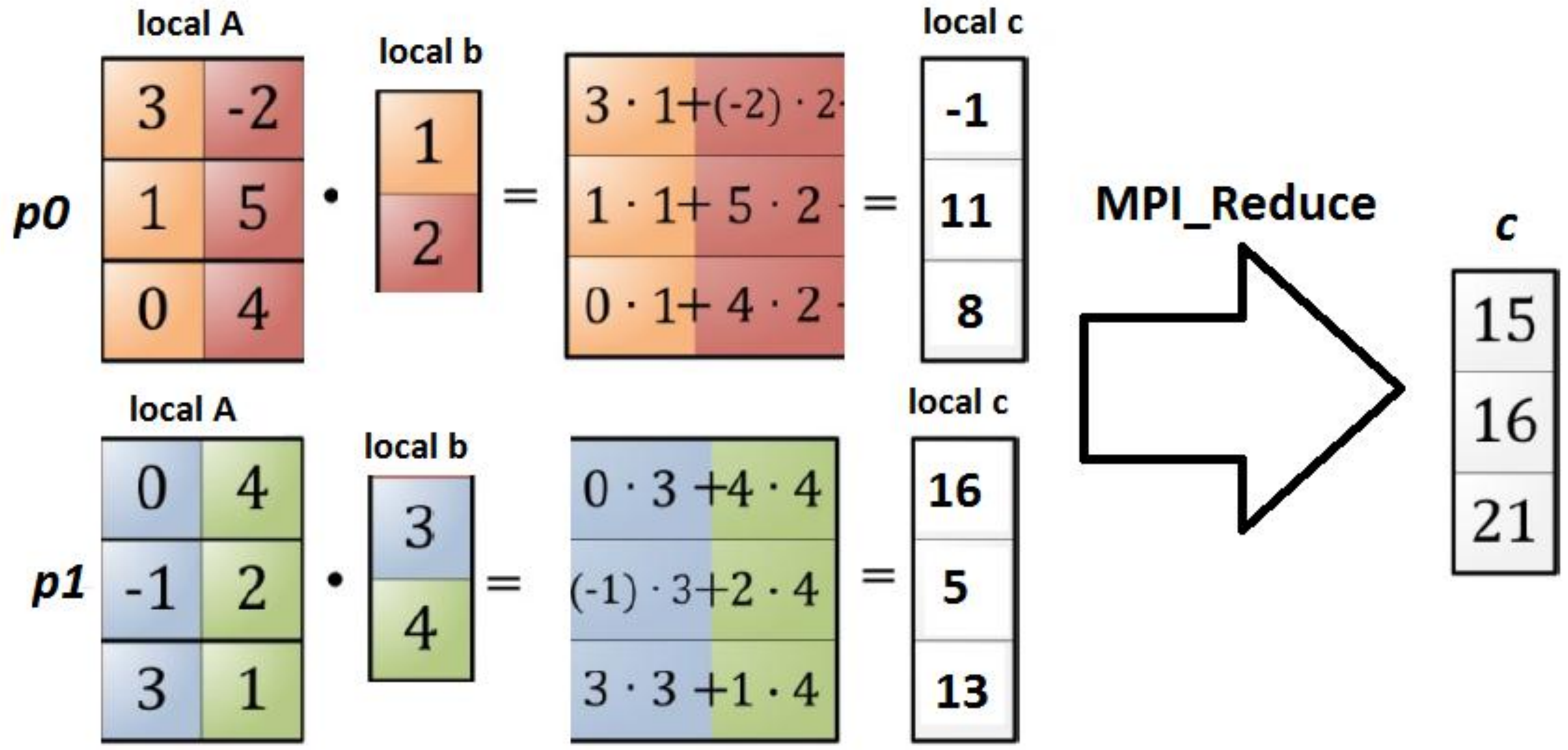
10 - Example - Rowwise

int **MPI_Gatherv** (void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf, int *recvcnts, int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm);

sendbuf	- početna adresa bafera za slanje
sendcnt	- broj elemenata za slanje
sendtype	- tip podataka elemenata koji se šalju
recvbuf	- adresa bafera za primanje podataka
recvcnts	- broj elemenata koje treba da primi
displs	- niz koji čuva informaciju o tome od koje adrese treba da smesti podatke sa određenog procesa
recvtype	- tip podataka elemenata koji se primaju
root	- rang procesa koji prima podatke
comm	- komunikator



Dekompozicija na blokove kolona



11 - Example - Columnwise

int **MPI_Scatterv**(void *sendbuf, int *sendcnts, int *displs, MPI Datatype sendtype, void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm);

sendbuf	- pocetna adresa bafera za slanje
sendcnt	- broj elemenata za slanje
displs	- niz koji čuva informaciju o tome od koje adrese treba da pošalju podaci za određeni proces
sendtype	- tip podataka elemenata koji se šalju
recvbuf	- adresa bafera za primanje podataka
recvcnts	- broj elemenata koje treba da primi
recvtype	- tip podataka elemenata koji se primaju
root	- rang procesa koji šalje podatke
comm	- komunikator