Matrix Multiplication

Thoai Nam

Outline

- Sequential matrix multiplication
- Algorithms for processor arrays
  - Matrix multiplication on 2-D mesh SIMD model
  - Matrix multiplication on hypercube SIMD model
- Matrix multiplication on UMA multiprocessors
- Matrix multiplication on multicomputers

Sequential Matrix Multiplication

```
Global  a[0..l-1,0..m-1], b[0..m-1][0..n-1],  {Matrices to be multiplied}
        c[0..l-1,0..n-1],                     {Product matrix}
        t,                                    {Accumulates dot product}
        i, j, k;
Begin
   for i:=0 to l-1 do
     for j:=0 to n-1 do
         t:=0;
        for k:=0 to m-1 do
           t:=t+a[i][k]*b[k][j];
         endfor k;
        c[i][j]:=t;
     endfor j;
    endfor i;
End.
```

Algorithms for Processor Arrays

- ❑ Matrix multiplication on 2-D mesh SIMD model
- ❑ Matrix multiplication on Hypercube SIMD model

Matrix Multiplication on 2D-Mesh SIMD Model

- Gentleman(1978) has shown that multiplication of two n*n matrices on the 2-D mesh SIMD model requires 0(n) routing steps
- We will consider a multiplication algorithm on a 2-D mesh SIMD model with wraparound connections

Matrix Multiplication on 2D-Mesh SIMD Model (cont'd)

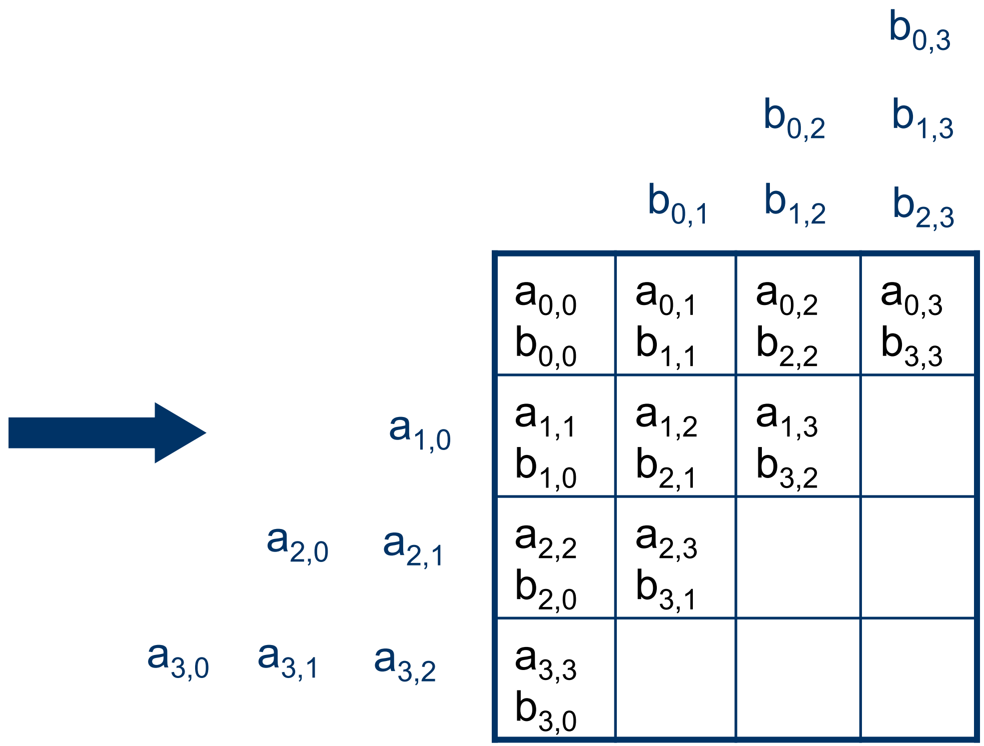
- For simplicity, we suppose that
  - Size of the mesh is n*n
  - Size of each matrix (A and B) is n*n
  - Each processor $P_{i,j}$ in the mesh (located at row i, column j) contains $a_{i,j}$ and $b_{i,j}$
- At the end of the algorithm, $P_{i,j}$ will hold the element $c_{i,j}$ of the product matrix

Matrix Multiplication on 2D-Mesh SIMD Model (cont'd)

- Major phases

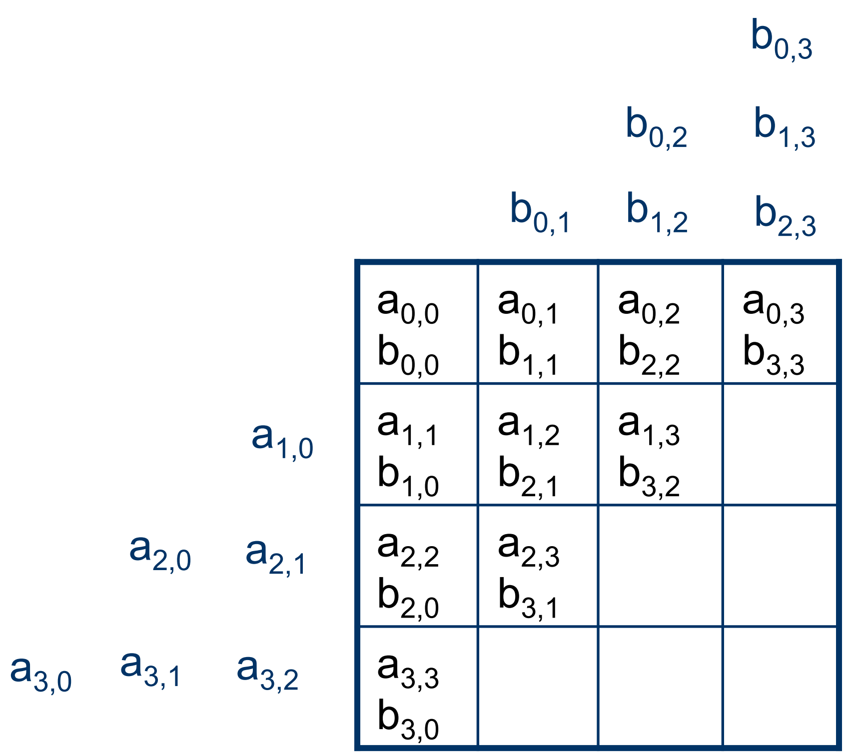
$a_{0,0}$ $b_{0,0}$	$a_{0,1}$ $b_{0,1}$	$a_{0,2}$ $b_{0,2}$	$a_{0,3}$ $b_{0,3}$
$a_{1,0}$ $b_{1,0}$	$a_{1,1}$ $b_{1,1}$	$a_{1,2}$ $b_{1,2}$	$a_{1,3}$ $b_{1,3}$
$a_{2,0}$ $b_{2,0}$	$a_{2,1}$ $b_{2,1}$	$a_{2,2}$ $b_{2,2}$	$a_{2,3}$ $b_{2,3}$
$a_{3,0}$ $b_{3,0}$	$a_{3,1}$ $b_{3,1}$	$a_{3,2}$ $b_{3,2}$	$a_{3,3}$ $b_{3,3}$

(a) Initial distribution of matrices A and B



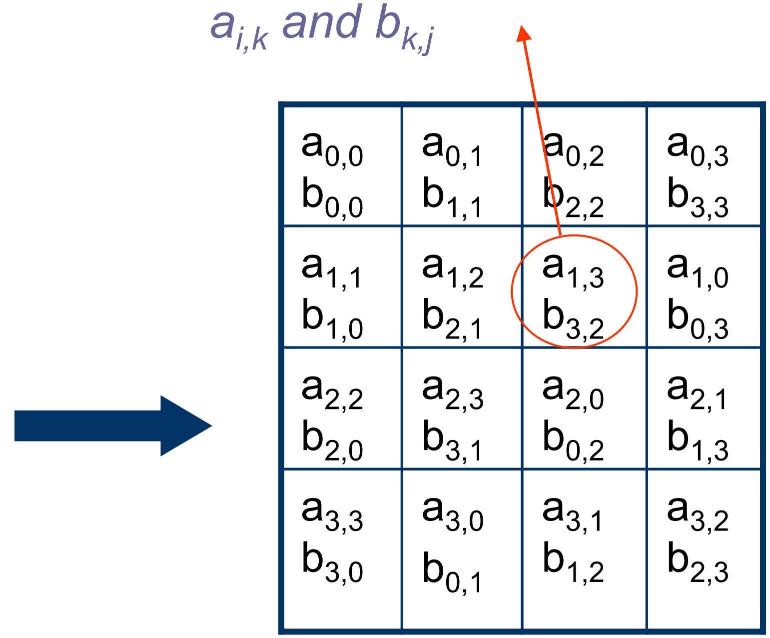
(b) Staggering all A's elements in row i to the left by i positions and all B's elements in col j upwards by i positions

Matrix Multiplication on 2D-Mesh SIMD Model (cont'd)



(b) *Staggering all A's elements in row i to the left by i positions and all B's elements in col j upwards by i positions*

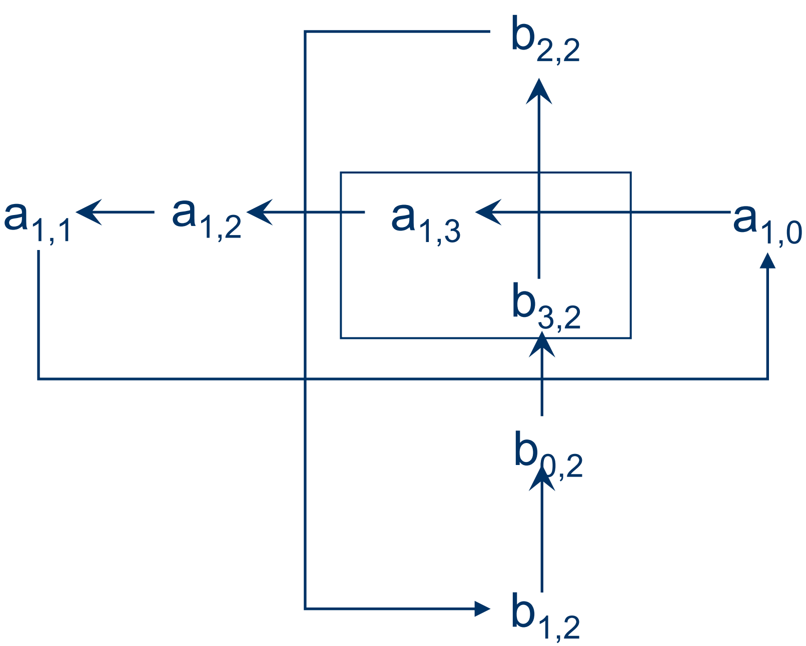
Each processor P(i,j) has a pair of elements to multiply $a_{i,k}$ and $b_{k,j}$



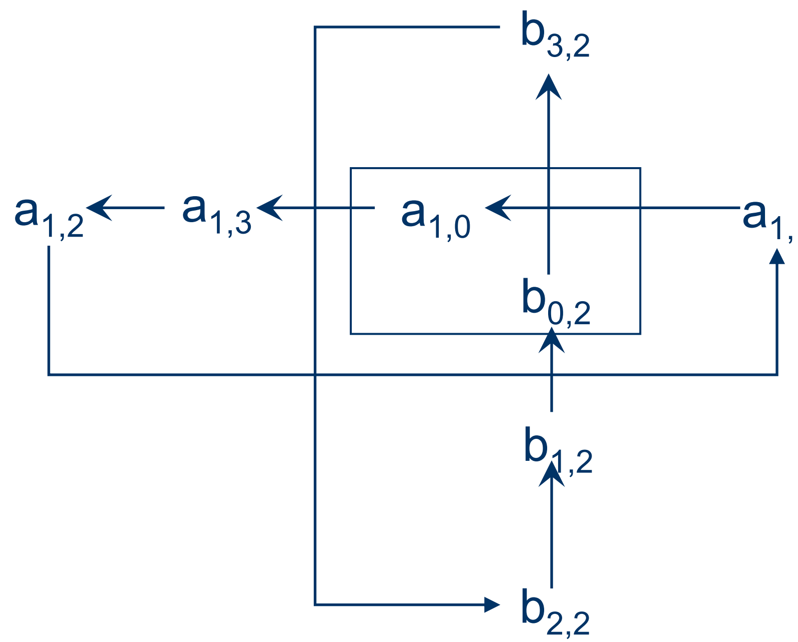
(c) *Distribution of 2 matrices A and B after staggering in a 2-D mesh with wraparound connection*

Matrix Multiplication on 2D-Mesh SIMD Model (cont'd)

- The rest steps of the algorithm from the viewpoint of processor P(1,2)

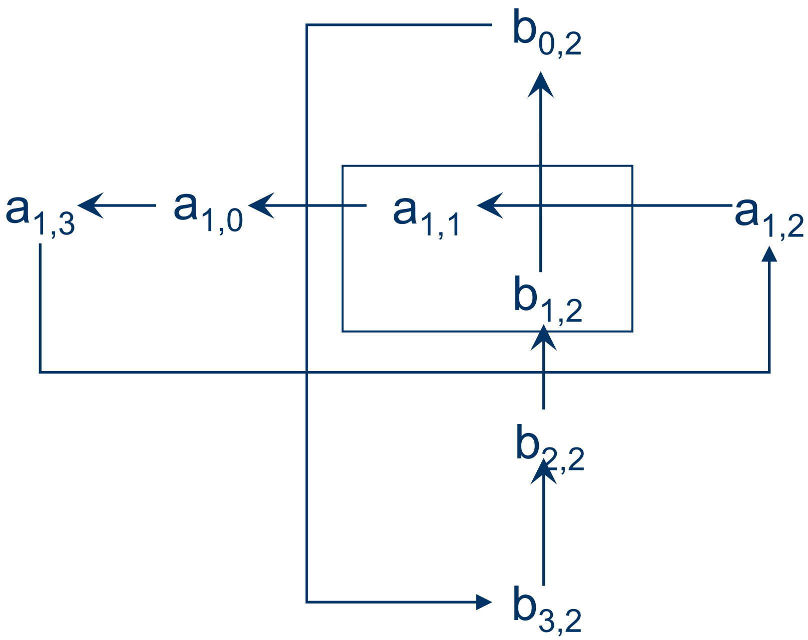


(a) First scalar multiplication step

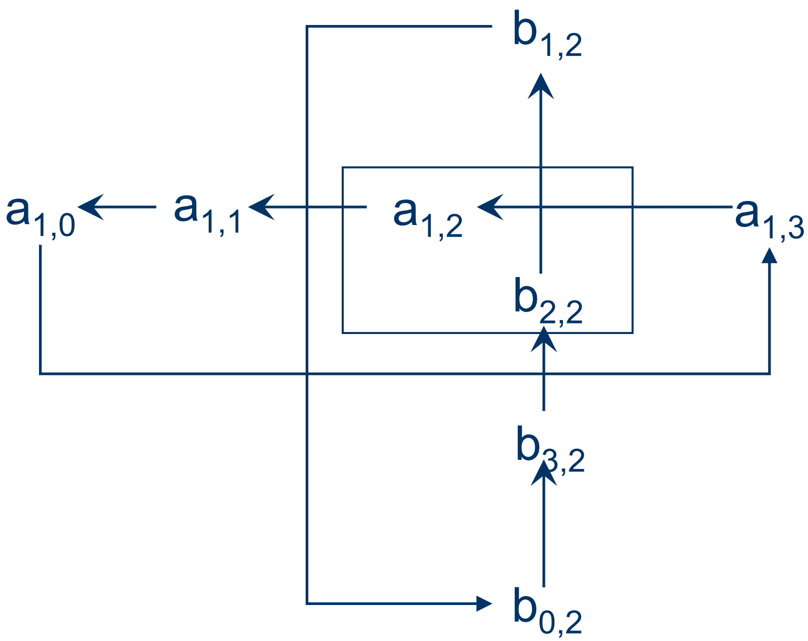


(b) Second scalar multiplication step after elements of A are cycled to the left and elements of B are cycled upwards

Matrix Multiplication on 2D-Mesh SIMD Model (cont'd)



(c) Third scalar multiplication step after second cycle step



(d) Third scalar multiplication step after second cycle step. At this point processor P(1,2) has computed the dot product $\mathbf{c}_{1,2}$

Matrix Multiplication on 2D-Mesh SIMD Model (cont'd)

Detailed Algorithm

Stagger 2 matrices
a[0..n-1,0..n-1] and b[0..n-1,0..n-1]

```
Global   n, {Dimension of matrices}
         k ;
Local    a, b, c;
Begin
   for k:=1 to n-1 do
       forall P(i,j) where 1 ≤ i,j < n do
          if i ≥ k then a:= fromleft(a);
          if j ≥ k  then b:=fromdown(b);
       end forall;
    endfor k;
```

Matrix Multiplication on 2D-Mesh SIMD Model (cont'd)

Compute dot product

```
forall P(i,j) where 0 ≤ i,j < n do
        c:= a*b;
    end forall;
    for k:=1 to n-1 do
        forall P(i,j) where 0 ≤ i,j < n do
           a:= fromleft(a);
           b:=fromdown(b);
           c:= c + a*b;
        end forall;
     endfor k;
End.
```

Matrix Multiplication on 2D-Mesh SIMD Model (cont'd)

- Can we implement the above mentioned algorithm on a 2-D mesh SIMD model without wrapparound connection?

Matrix Multiplication Algorithm for Multiprocessors

- **Design strategy 5**
  - If load balancing is not a problem, maximize grain size
    - Grain size: the amount of work performed between processor interactions
- Things to be considered
  - Parallelizing the most outer loop of the sequential algorithm is a good choice since the attained grain size ($0(n^3/p)$) is the biggest
  - Resolving memory contention as much as possible

Matrix Multiplication Algorithm for UMA Multiprocessors

Algorithm using p processors

```
Global   n,                                        {Dimension of matrices}
         a[0..n-1,0..n-1], b[0..n-1,0..n-1];       {Two input matrices}
         c[0..n-1,0..n-1];                         {Product matrix}
Local    i,j,k,t;
Begin
    forall Pm where 1 ≤ m ≤ p  do
      for i:=m to n step p do
         for j:= 1 to n to
            t:=0;
            for k:=1 to n do t:=t+a[i,k]*b[k,j];
         endfor j;
         c[i][j]:=t;
       endfor i;
    end forall;
End.
```

Matrix Multiplication Algorithm for NUMA Multiprocessors

- Things to be considered
  - Try to resolve memory contention as much as possible
  - Increase the locality of memory references to reduce memory access time
- **Design strategy 6**
  - Reduce average memory latency time by increasing locality
- The block matrix multiplication algorithm is a reasonable choice in this situation
  - Section 7.3, p.187, Parallel Computing: Theory and Practice

Matrix Multiplication Algorithm for Multicomputers

- We will study 2 algorithms on multicomputers
  - Row-Column-Oriented Algorithm
  - Block-Oriented Algorithm

Row-Column-Oriented Algorithm

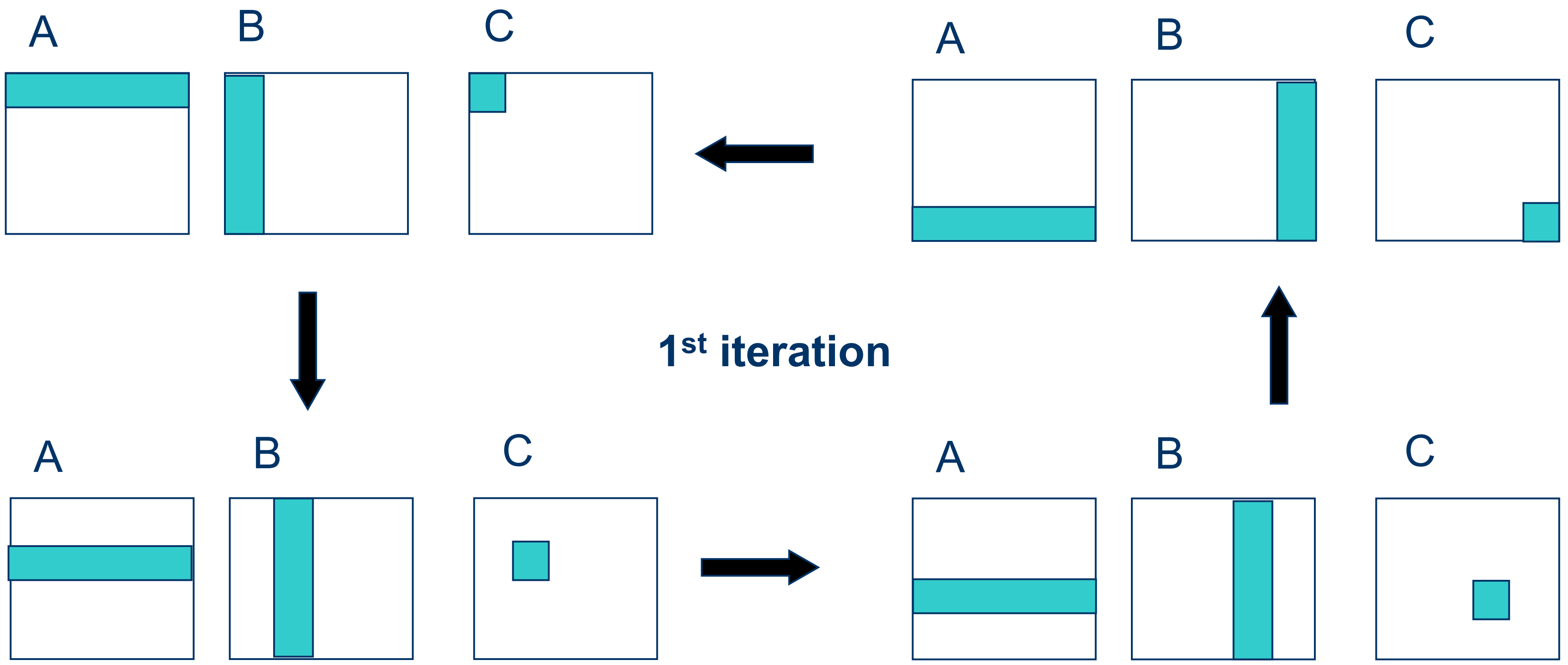
- The processes are organized as a ring
  - Step 1: Initially, each process is given 1 row of the matrix A and 1 column of the matrix B
  - Step 2: Each process uses vector multiplication to get 1 element of the product matrix C.
  - Step 3: After a process has used its column of matrix B, it fetches the next column of B from its successor in the ring
  - Step 4: If all rows of B have already been processed, quit. Otherwise, go to step 2

Row-Column-Oriented Algorithm (cont'd)

- Why do we have to organize processes as a ring and make them use B's rows in turn?
- **Design strategy 7:**
  - Eliminate contention for shared resources by changing the order of data access

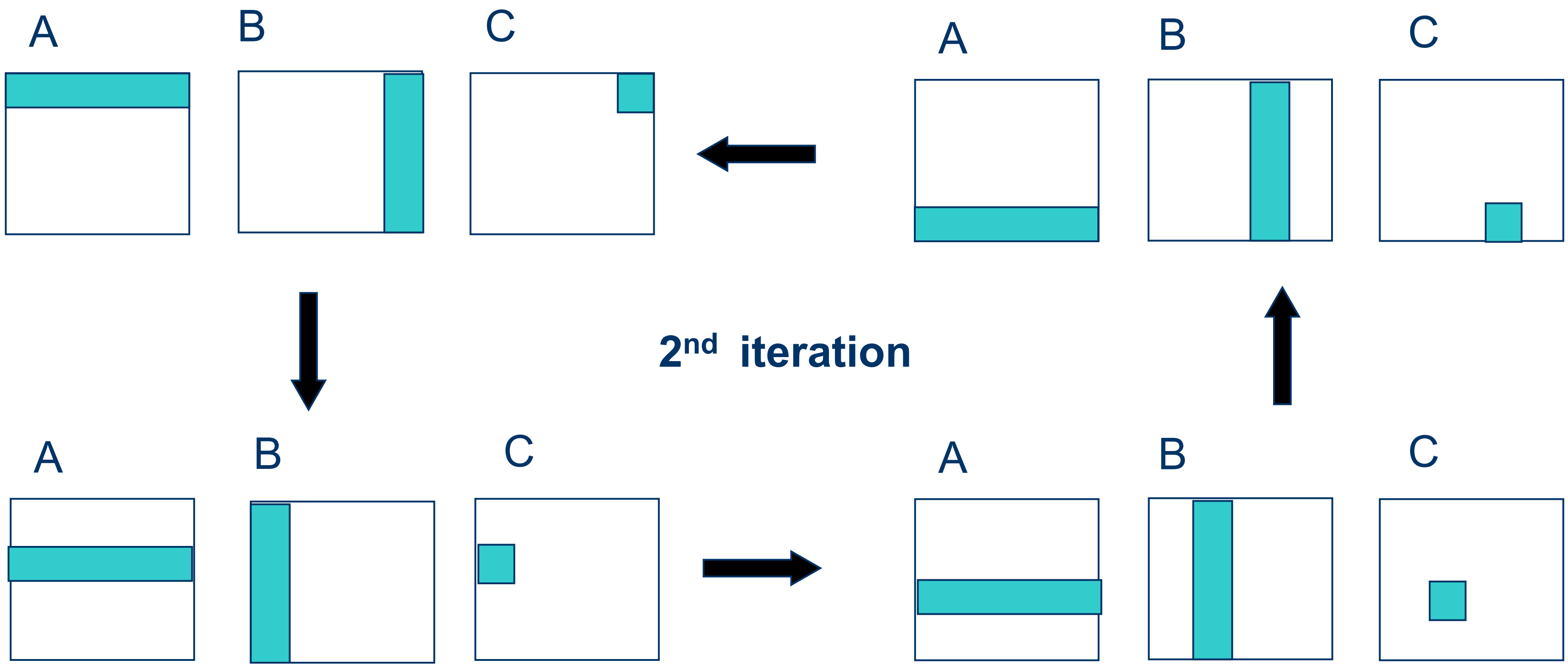
Row-Column-Oriented Algorithm (cont'd)

Example: Use 4 processes to mutliply two matrices A_{4*4} and B_{4*4}



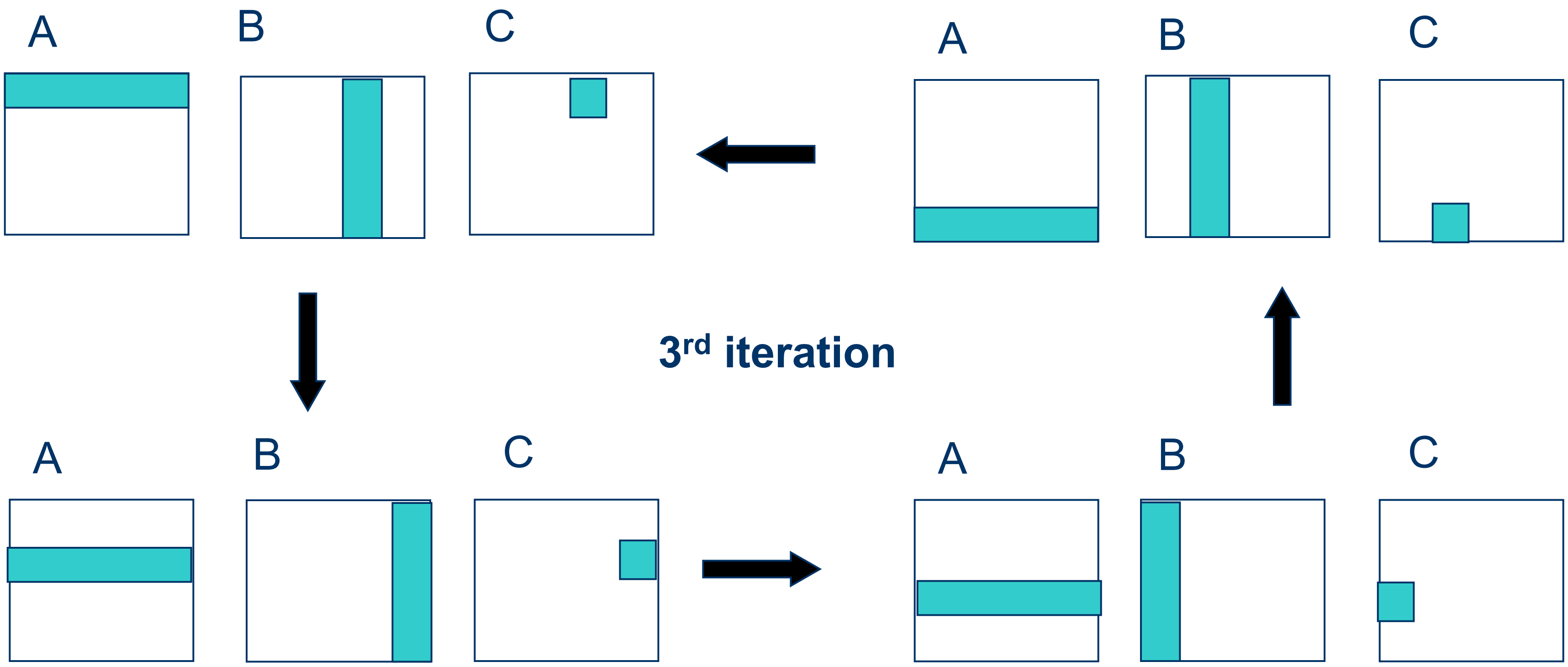
Row-Column-Oriented Algorithm (cont'd)

<u>Example</u>: Use 4 processes to mutliply two matrices A_{4*4} and B_{4*4}



Row-Column-Oriented Algorithm (cont’d)

<u>Example</u>: Use 4 processes to mutliply two matrices A_{4*4} and B_{4*4}



Row-Column-Oriented Algorithm (cont'd)

Example: Use 4 processes to mutliply two matrices A_{4*4} and B_{4*4}