2.9 PBLAS routines for matrix-matrix operations (Level 3) Scalable Universal Matrix Multiply Algorithm (SUMMA)

An alternative for Cannon algorithm represents the SUMMA algorithm:

- SUMMA = Scalable Universal Matrix Multiply Algorithm
- Slightly less efficient, but simpler and easier to generalize
- Presentation from *van de Geijn* and *Watts [1997]*
  - www.netlib.org/lapack/lawns/lawn96.ps •
  - Similar ideas appeared many times
- Used in practice in PBLAS = Parallel BLAS
  - www.netlib.org/lapack/lawns/lawn100.ps

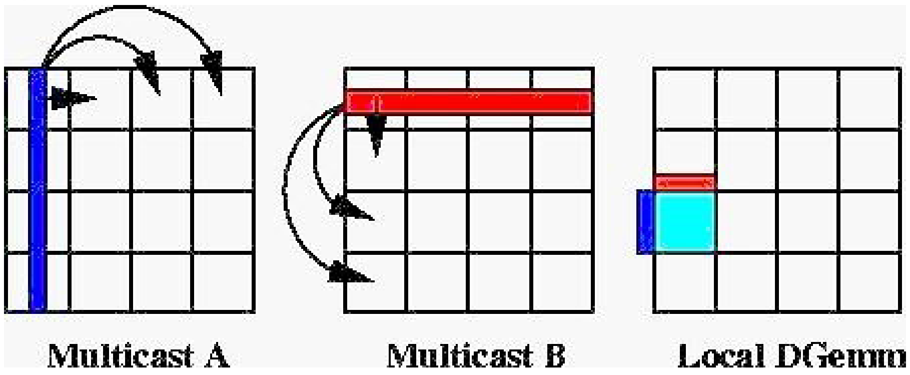
Naive matrix multiply

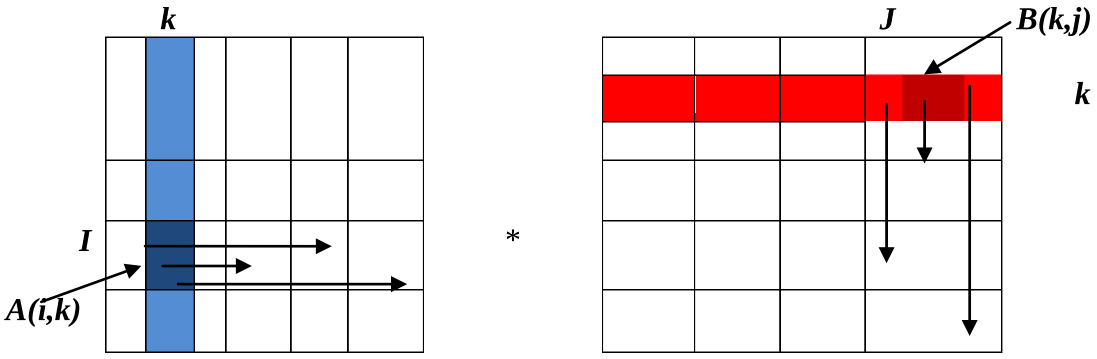
- For i = 0 to n
- For j = 0 to n
- For k = 0 to n
- C[i,j] += A[i,k]*B[k,j]

Calculates n^2 dot products (inner products) C[i,j] = A[i,:]*B[:,j]

The main steps of the algorithm:

- For each *k* (between 0 and n-1),
  - Owner of partial row *k* broadcasts that row along its process column;
  - Owner of partial column *k* broadcasts that column along its process row





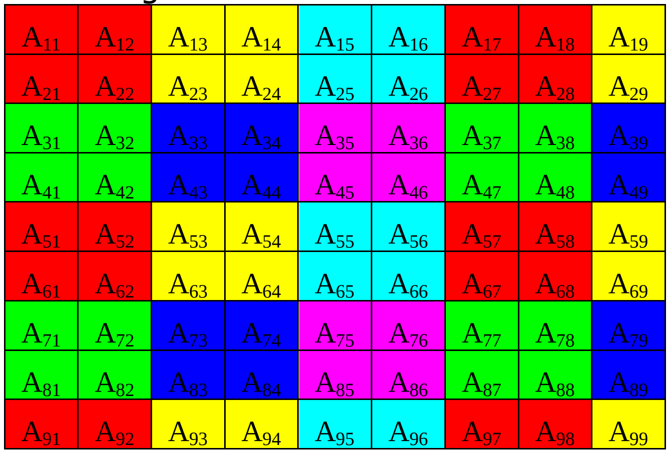
			C(I,J)

- ***I, J*** represent all rows, columns owned by a processor
- ***k*** is a single row or column (or a block of ***b*** rows or columns)
- ***$C(I,J) = C(I,J) + \sum_k A(I,k)*B(k,J)$***

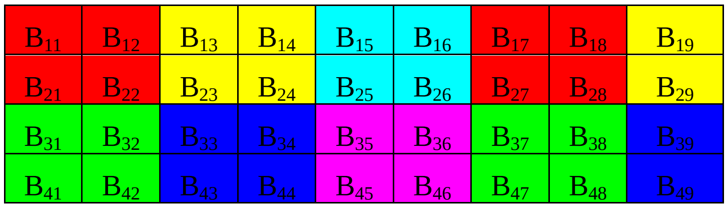
Assume a p_r by p_c processor grid ($p_r = p_c = 4$ above)
Complete pseudo code of the algorithm.

```
// On each process P(i,j):
• For k=0 to n-1     //… or n/b-1 where b is the block
                     //size
                     //…= # cols in A(I,k) and # rows in
                       B(k,J)
    • for all I = 1 to $p_r$      //… in parallel
    • owner of A(I,k) broadcasts it to whole processor row
          [adica, procesul $p_r$ transmite A(I,k) tuturor proceselor de pe linia r]
    • for all J = 1 to $p_c$      //… in parallel
    • owner of B(k,J) broadcasts it to whole processor column
          [adica, procesul $p_c$ transmite B(k,J) tuturor proceselor de pe coloana c]
    • Receive A(I,k) into Acol
    • Receive B(k,J) into Brow
    • C(myproc,myproc) = C(myproc,myproc)+Acol*Brow
• Endfor
```

We consider the following two matrices



A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	A_{17}	A_{18}	A_{19}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}	A_{27}	A_{28}	A_{29}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}	A_{36}	A_{37}	A_{38}	A_{39}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}	A_{46}	A_{47}	A_{48}	A_{49}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}	A_{56}	A_{57}	A_{58}	A_{59}
A_{61}	A_{62}	A_{63}	A_{64}	A_{65}	A_{66}	A_{67}	A_{68}	A_{69}
A_{71}	A_{72}	A_{73}	A_{74}	A_{75}	A_{76}	A_{77}	A_{78}	A_{79}
A_{81}	A_{82}	A_{83}	A_{84}	A_{85}	A_{86}	A_{87}	A_{88}	A_{89}
A_{91}	A_{92}	A_{93}	A_{94}	A_{95}	A_{96}	A_{97}	A_{98}	A_{99}



B_{11}	B_{12}	B_{13}	B_{14}	B_{15}	B_{16}	B_{17}	B_{18}	B_{19}
B_{21}	B_{22}	B_{23}	B_{24}	B_{25}	B_{26}	B_{27}	B_{28}	B_{29}
B_{31}	B_{32}	B_{33}	B_{34}	B_{35}	B_{36}	B_{37}	B_{38}	B_{39}
B_{41}	B_{42}	B_{43}	B_{44}	B_{45}	B_{46}	B_{47}	B_{48}	B_{49}

B_{51}	B_{52}	B_{53}	B_{54}	B_{55}	B_{56}	B_{57}	B_{58}	B_{59}
B_{61}	B_{62}	B_{63}	B_{64}	B_{65}	B_{66}	B_{67}	B_{68}	B_{69}
B_{71}	B_{72}	B_{73}	B_{74}	B_{75}	B_{76}	B_{77}	B_{78}	B_{79}
B_{81}	B_{82}	B_{83}	B_{84}	B_{85}	B_{86}	B_{87}	B_{88}	B_{89}
B_{91}	B_{92}	B_{93}	B_{94}	B_{95}	B_{96}	B_{97}	B_{98}	B_{99}

As a result of distribution of these matrices using “2-D ciclic” algorithm on the grid of 2x3 processes (2 lines and 3 columns) with block dimensions 2x2, we obtain:

Matrix ***A***	Matrix ***B***
$\mathbf{A}_{(0,0)}=\begin{pmatrix} A_{11} & A_{12} & A_{17} & A_{18} \\ A_{21} & A_{22} & A_{27} & A_{28} \\ A_{51} & A_{52} & A_{57} & A_{58} \\ A_{61} & A_{62} & A_{67} & A_{68} \\ A_{91} & A_{92} & A_{97} & A_{98} \end{pmatrix}$	$\mathbf{B}_{(0,0)}=\begin{pmatrix} B_{11} & B_{12} & B_{17} & B_{18} \\ B_{21} & B_{22} & B_{27} & B_{28} \\ B_{51} & B_{52} & B_{57} & B_{58} \\ B_{61} & B_{62} & B_{67} & B_{68} \\ B_{91} & B_{92} & B_{97} & B_{98} \end{pmatrix}$
$\mathbf{A}_{(0,1)}=\begin{pmatrix} A_{13} & A_{14} & A_{19} \\ A_{23} & A_{24} & A_{29} \\ A_{53} & A_{54} & A_{59} \\ A_{63} & A_{64} & A_{69} \\ A_{93} & A_{94} & A_{99} \end{pmatrix}$	$\mathbf{B}_{(0,1)}=\begin{pmatrix} B_{13} & B_{14} & B_{19} \\ B_{23} & B_{24} & B_{29} \\ B_{53} & B_{54} & B_{59} \\ B_{63} & B_{64} & B_{69} \\ B_{93} & B_{94} & B_{99} \end{pmatrix}$
$\mathbf{A}_{(0,2)}=\begin{pmatrix} A_{15} & A_{16} \\ A_{25} & A_{26} \\ A_{55} & A_{56} \\ A_{65} & A_{66} \\ A_{95} & A_{96} \end{pmatrix}$	$\mathbf{B}_{(0,2)}=\begin{pmatrix} B_{15} & B_{16} \\ B_{25} & B_{26} \\ B_{55} & B_{56} \\ B_{65} & B_{66} \\ B_{95} & B_{96} \end{pmatrix}$
$\mathbf{A}_{(1,0)}=\begin{pmatrix} A_{31} & A_{32} & A_{37} & A_{38} \\ A_{41} & A_{42} & A_{47} & A_{48} \\ A_{71} & A_{72} & A_{77} & A_{78} \\ A_{81} & A_{82} & A_{87} & A_{88} \end{pmatrix}$	$\mathbf{B}_{(1,0)}=\begin{pmatrix} B_{31} & B_{32} & B_{37} & B_{38} \\ B_{41} & B_{42} & B_{47} & B_{48} \\ B_{71} & B_{72} & B_{77} & B_{78} \\ B_{81} & B_{82} & B_{87} & B_{88} \end{pmatrix}$

$\mathbf{A}_{(1,1)}=\begin{pmatrix} A_{33} & A_{34} & A_{39} \\ A_{43} & A_{44} & A_{49} \\ A_{73} & A_{74} & A_{79} \\ A_{83} & A_{84} & A_{89} \end{pmatrix}$	$\mathbf{B}_{(1,1)}=\begin{pmatrix} B_{33} & B_{34} & B_{39} \\ B_{43} & B_{44} & B_{49} \\ B_{73} & B_{74} & B_{79} \\ B_{83} & B_{84} & B_{89} \end{pmatrix}$
$\mathbf{A}_{(1,2)}=\begin{pmatrix} A_{35} & A_{36} \\ A_{45} & A_{46} \\ A_{75} & A_{76} \\ A_{85} & A_{86} \end{pmatrix}$	$\mathbf{B}_{(1,2)}=\begin{pmatrix} B_{35} & B_{36} \\ B_{45} & B_{46} \\ B_{75} & B_{76} \\ B_{85} & B_{86} \end{pmatrix}$

So in a compact form we have

	0	**1**	**2**
0	$\boldsymbol{A_{(0,0)}}$	$\boldsymbol{A_{(0,1)}}$	$\boldsymbol{A_{(0,2)}}$
1	$\boldsymbol{A_{(1,0)}}$	$\boldsymbol{A_{(1,1)}}$	$\boldsymbol{A_{(1,2)}}$

	0	**1**	**2**
0	$\boldsymbol{B_{(0,0)}}$	$\boldsymbol{B_{(0,1)}}$	$\boldsymbol{B_{(0,2)}}$
1	$\boldsymbol{B_{(1,0)}}$	$\boldsymbol{B_{(1,1)}}$	$\boldsymbol{B_{(1,2)}}$

To determine the product **C=AB** we use the algorithm SUMMA**.** Processes will determine the following submatrices of the matrix **C:**

$$\mathbf{C}_{(0,0)}=\begin{pmatrix} C_{11} & C_{12} & C_{17} & C_{18} \\ C_{21} & C_{22} & C_{27} & C_{28} \\ C_{51} & C_{52} & C_{57} & C_{68} \\ C_{61} & C_{62} & C_{67} & C_{68} \\ C_{91} & C_{92} & C_{97} & C_{98} \end{pmatrix} \quad \mathbf{C}_{(0,1)}=\begin{pmatrix} C_{13} & C_{14} & C_{19} \\ C_{23} & C_{24} & C_{29} \\ C_{53} & C_{54} & C_{59} \\ C_{63} & C_{64} & C_{69} \\ C_{93} & C_{94} & C_{99} \end{pmatrix} \quad \mathbf{C}_{(0,2)}=\begin{pmatrix} C_{15} & C_{16} \\ C_{25} & C_{26} \\ C_{55} & C_{56} \\ C_{65} & C_{66} \\ C_{95} & C_{96} \end{pmatrix}$$

$$\mathbf{C}_{(1,0)}=\begin{pmatrix} C_{31} & C_{32} & C_{37} & C_{38} \\ C_{41} & C_{42} & C_{47} & C_{48} \\ C_{71} & C_{72} & C_{77} & C_{78} \\ C_{81} & C_{82} & C_{87} & C_{88} \end{pmatrix} \quad \mathbf{C}_{(1,1)}=\begin{pmatrix} C_{33} & C_{34} & C_{39} \\ C_{43} & C_{44} & C_{49} \\ C_{73} & C_{74} & C_{79} \\ C_{83} & C_{84} & C_{89} \end{pmatrix} \quad \mathbf{C}_{(1,2)}=\begin{pmatrix} C_{35} & C_{36} \\ C_{45} & C_{46} \\ C_{75} & C_{76} \\ C_{85} & C_{86} \end{pmatrix}$$

Here, for example, the process (0,0) can do:

$C_{11}=A_{11}B_{11} + A_{12}B_{21}+ A_{13}B_{31} +A_{14}B_{41} +A_{15}B_{51}+ A_{16}B_{61}+ A_{17}B_{71} +A_{18}B_{81} +A_{19}B_{91}$

$C_{12}=A_{11}B_{12} + A_{12}B_{22}+ A_{13}B_{32} +A_{14}B_{42} +A_{15}B_{52}+ A_{16}B_{62}+ A_{17}B_{72} +A_{18}B_{82} +A_{19}B_{92}$

$\mathrm{C}_{17}=\mathrm{A}_{11}\mathrm{B}_{17} + \mathrm{A}_{12}\mathrm{B}_{27}+ \mathrm{A}_{13}\mathrm{B}_{37} +\mathrm{A}_{14}\mathrm{B}_{47} +\mathrm{A}_{15}\mathrm{B}_{57}+ \mathrm{A}_{16}\mathrm{B}_{67}+ \mathrm{A}_{17}\mathrm{B}_{77} +\mathrm{A}_{18}\mathrm{B}_{87} +\mathrm{A}_{19}\mathrm{B}_{97}$

$C_{18}=A_{11}B_{18} + A_{12}B_{28}+ A_{13}B_{38} +A_{14}B_{48} +A_{15}B_{58}+ A_{16}B_{68}+ A_{17}B_{78} +A_{18}B_{88} +A_{19}B_{98}$

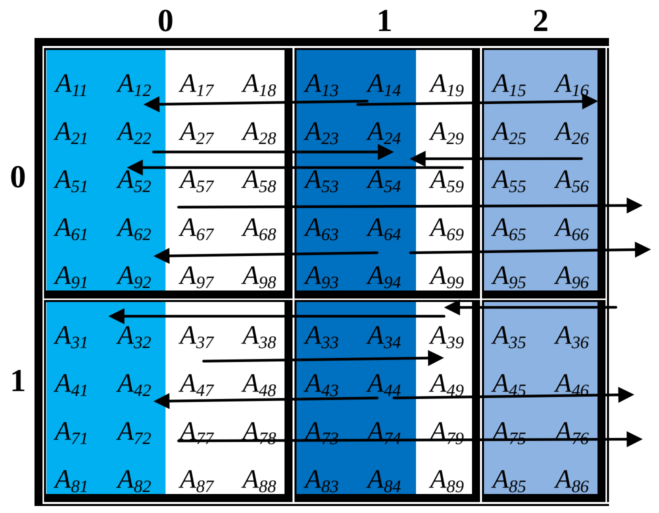
$C_{21}=A_{21}B_{11} + A_{22}B_{21}+ A_{23}B_{31} +A_{24}B_{41} +A_{25}B_{51}+ A_{26}B_{61}+ A_{27}B_{71} +A_{28}B_{81} +A_{29}B_{91}$
$C_{22}=A_{21}B_{12} + A_{22}B_{22}+ A_{23}B_{32} +A_{24}B_{42} +A_{25}B_{52}+ A_{26}B_{62}+ A_{27}B_{72} +A_{28}B_{82} +A_{29}B_{92}$
$C_{27}=A_{21}B_{17} + A_{22}B_{27}+ A_{23}B_{37} +A_{24}B_{47} +A_{25}B_{57}+ A_{26}B_{67}+ A_{27}B_{77} +A_{28}B_{87} +A_{29}B_{97}$
$C_{28}=A_{21}B_{18}+ A_{22}B_{28}+ A_{23}B_{38}+A_{24}B_{48} +A_{25}B_{58}+ A_{26}B_{68}+ A_{27}B_{77} +A_{28}B_{88} +A_{29}B_{98}$

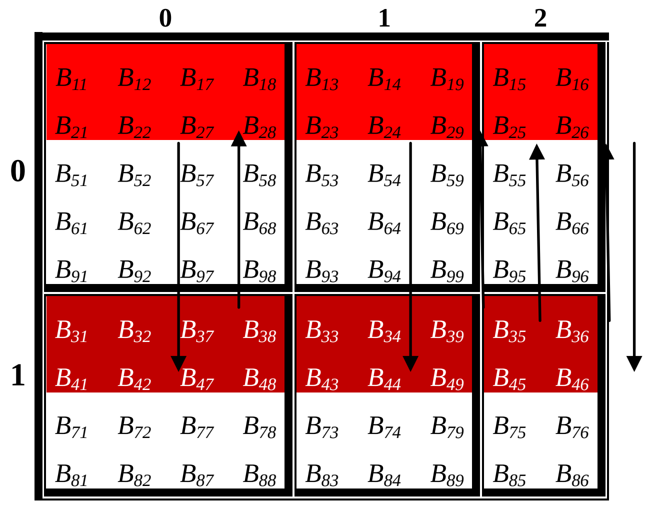
$C_{51}=A_{51}B_{11} + A_{52}B_{21}+ A_{53}B_{31} +A_{54}B_{41} +A_{55}B_{51}+ A_{56}B_{61}+ A_{57}B_{71} +A_{58}B_{81} +A_{59}B_{91}$
$C_{52}=A_{51}B_{12} + A_{52}B_{22}+ A_{53}B_{32} +A_{54}B_{42} +A_{55}B_{52}+ A_{56}B_{62}+ A_{57}B_{72} +A_{58}B_{82} +A_{59}B_{92}$
$C_{57}=A_{51}B_{17} + A_{52}B_{27}+ A_{53}B_{37} +A_{54}B_{47} +A_{55}B_{57}+ A_{56}B_{67}+ A_{57}B_{77} +A_{58}B_{87} +A_{59}B_{97}$
$C_{58}=A_{51}B_{18}+ A_{52}B_{28}+ A_{53}B_{38}+A_{54}B_{48} +A_{55}B_{58}+ A_{56}B_{68}+ A_{57}B_{78} +A_{58}B_{88} +A_{59}B_{98}$

$C_{61}=A_{61}B_{11} + A_{62}B_{21}+ A_{63}B_{31} +A_{64}B_{41} +A_{65}B_{51}+ A_{66}B_{61}+ A_{67}B_{71} +A_{68}B_{81} +A_{69}B_{91}$
$C_{62}=A_{61}B_{12} + A_{62}B_{22}+ A_{63}B_{32} +A_{64}B_{42} +A_{65}B_{52}+ A_{66}B_{62}+ A_{67}B_{72} +A_{68}B_{82} +A_{69}B_{92}$
$C_{67}=A_{61}B_{17} + A_{62}B_{27}+ A_{63}B_{37} +A_{64}B_{47} +A_{65}B_{57}+ A_{66}B_{67}+ A_{67}B_{77} +A_{68}B_{87} +A_{69}B_{97}$
$C_{68}=A_{61}B_{18}+ A_{62}B_{28}+ A_{63}B_{38}+A_{64}B_{48} +A_{65}B_{58}+ A_{66}B_{68}+ A_{67}B_{77} +A_{68}B_{88} +A_{69}B_{98}$

$C_{91}=A_{91}B_{11} + A_{92}B_{21}+ A_{93}B_{31} +A_{94}B_{41} +A_{95}B_{51}+ A_{96}B_{61}+ A_{97}B_{71} +A_{98}B_{81} +A_{99}B_{91}$
$C_{92}=A_{91}B_{12} + A_{92}B_{22}+ A_{93}B_{32} +A_{94}B_{42} +A_{95}B_{52}+ A_{96}B_{62}+ A_{97}B_{72} +A_{98}B_{82} +A_{99}B_{92}$
$C_{97}=A_{91}B_{17} + A_{92}B_{27}+ A_{93}B_{37} +A_{94}B_{47} +A_{95}B_{57}+ A_{96}B_{67}+ A_{97}B_{77} +A_{98}B_{87} +A_{99}B_{97}$
$C_{98}=A_{91}B_{18}+ A_{92}B_{28}+ A_{93}B_{38}+A_{94}B_{48} +A_{95}B_{58}+ A_{96}B_{68}+ A_{97}B_{78} +A_{98}B_{88} +A_{99}B_{98}$

We apply the algorithm SUMMA. The initial situation is:


0 1 2
0
A11 A12 A17 A18 A13 A14 A19 A15 A16
A21 A22 A27 A28 A23 A24 A29 A25 A26
A51 A52 A57 A58 A53 A54 A59 A55 A56
A61 A62 A67 A68 A63 A64 A69 A65 A66
A91 A92 A97 A98 A93 A94 A99 A95 A96
1
A31 A32 A37 A38 A33 A34 A39 A35 A36
A41 A42 A47 A48 A43 A44 A49 A45 A46
A71 A72 A77 A78 A73 A74 A79 A75 A76
A81 A82 A87 A88 A83 A84 A89 A85 A86

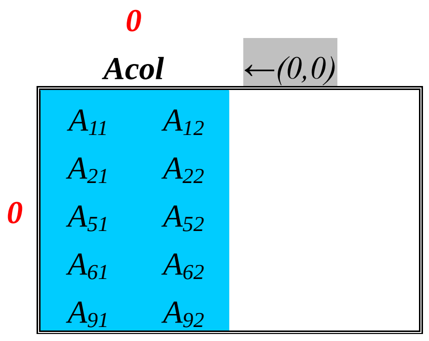

0 1 2
0
B11 B12 B17 B18 B13 B14 B19 B15 B16
B21 B22 B27 B28 B23 B24 B29 B25 B26
B51 B52 B57 B58 B53 B54 B59 B55 B56
B61 B62 B67 B68 B63 B64 B69 B65 B66
B91 B92 B97 B98 B93 B94 B99 B95 B96
1
B31 B32 B37 B38 B33 B34 B39 B35 B36
B41 B42 B47 B48 B43 B44 B49 B45 B46
B71 B72 B77 B78 B73 B74 B79 B75 B76
B81 B82 B87 B88 B83 B84 B89 B85 B86

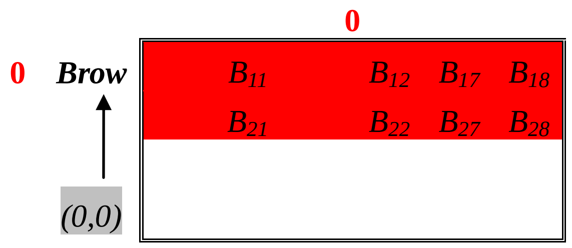
The process (0,0)

On the base of algorithm SUMMA we obtain:

Let *k*=2 (the block length, i.e. there are transmitted two rows and two columns).

- Iteration 0. There is no data transmission




0
0
Brow
(0,0)
B_{11} B_{12} B_{17} B_{18}
B_{21} B_{22} B_{27} B_{28}

Then it is possible to calculate the product $C_{(0,0)}= C_{(0,0)}+$*Acol*Brow*. So we obtain

$C^0_{(0,0)}=$	$C^0_{11}=A_{11}B_{11}+A_{12}B_{21}$	$C^0_{12}=A_{11}B_{12}+A_{12}B_{22}$	$C^0_{17}=A_{11}B_{17}+A_{12}B_{27}$	$C^0_{18}=A_{11}B_{18}+A_{12}B_{28}$
	$C^0_{21}=A_{21}B_{11}+A_{22}B_{21}$	$C^0_{22}=A_{21}B_{12}+A_{22}B_{22}$	$C^0_{27}=A_{21}B_{17}+A_{22}B_{27}$	$C^0_{28}=A_{21}B_{18}+A_{22}B_{28}$
	$C^0_{51}=A_{51}B_{11}+A_{52}B_{21}$	$C^0_{52}=A_{51}B_{12}+A_{52}B_{22}$	$C^0_{57}=A_{51}B_{17}+A_{52}B_{27}$	$C^0_{58}=A_{51}B_{18}+A_{52}B_{28}$
	$C^0_{61}=A_{61}B_{11}+A_{62}B_{21}$	$C^0_{62}=A_{61}B_{12}+A_{62}B_{22}$	$C^0_{67}=A_{61}B_{17}+A_{62}B_{27}$	$C^0_{68}=A_{61}B_{18}+A_{62}B_{28}$
	$C^0_{91}=A_{91}B_{11}+A_{92}B_{21}$	$C^0_{92}=A_{91}B_{12}+A_{92}B_{22}$	$C^0_{97}=A_{91}B_{17}+A_{92}B_{27}$	$C^0_{98}=A_{91}B_{18}+A_{92}B_{28}$

Finally, $C_{(0,0)}= C^0_{(0,0)}$

Therefore, at the end of this iteration the process (0,0) already has calculated

$C_{11}=A_{11}B_{11} + A_{12}B_{21}$
$C_{12}=A_{11}B_{12} + A_{12}B_{22}$
$C_{17}=A_{11}B_{17} + A_{12}B_{27}$
$C_{18}=A_{11}B_{18} + A_{12}B_{28}$

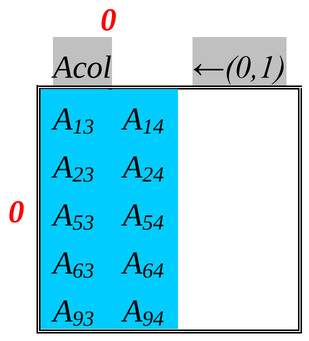
$C_{21}=A_{21}B_{11} + A_{22}B_{21}$
$C_{22}=A_{21}B_{12} + A_{22}B_{22}$
$C_{27}=A_{21}B_{17} + A_{22}B_{27}$
$C_{28}=A_{21}B_{18}+ A_{22}B_{28}$

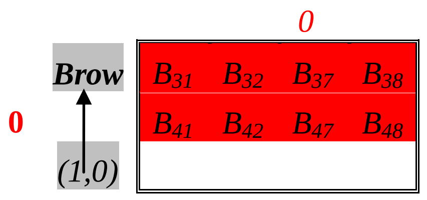
$C_{51}=A_{51}B_{11} + A_{52}B_{21}$
$C_{52}=A_{51}B_{12} + A_{52}B_{22}$
$C_{57}=A_{51}B_{17} + A_{52}B_{27}$
$C_{58}=A_{51}B_{18}+ A_{52}B_{28}$

$C_{61}=A_{61}B_{11} + A_{62}B_{21}$
$C_{62}=A_{61}B_{12} + A_{62}B_{22}$
$C_{67}=A_{61}B_{17} + A_{62}B_{27}$
$C_{68}=A_{61}B_{18}+ A_{62}B_{28}$

$C_{91}=A_{91}B_{11} + A_{92}B_{21}$
$C_{92}=A_{91}B_{12} + A_{92}B_{22}$
$C_{97}=A_{91}B_{17} + A_{92}B_{27}$
$C_{98}=A_{91}B_{18}+ A_{92}B_{28}$

- Iteration 1. Here we have for *Acol* and *Bcol* the following:





Then it is possible to calculate the product $C_{(0,0)}= C_{(0,0)}$+Acol*Brow. So we obtain

	$C^1_{1,1}=A_{13}B_{31}+A_{14}B_{41}$	$C^1_{1,2}= A_{13}B_{32}+A_{14}B_{42}$	$C^1_{1,7}= A_{13}B_{37}+A_{14}B_{47}$	$C^1_{1,8}= A_{13}B_{38}+A_{14}B_{48}$
	$C^1_{2,1}=A_{23}B_{31}+A_{24}B_{41}$	$C^1_{22}= A_{23}B_{32}+A_{24}B_{42}$	$C^1_{2,7}= A_{23}B_{37}+A_{24}B_{47}$	$C^1_{2,8}= A_{23}B_{38}+A_{24}B_{48}$
$C^1_{(0,0)}=$	$C^1_{5,1}=A_{53}B_{31}+A_{54}B_{41}$	$C^1_{52}= A_{53}B_{32}+A_{54}B_{42}$	$C^1_{57}= A_{53}B_{37}+A_{54}B_{47}$	$C^1_{58}= A_{53}B_{38}+A_{54}B_{48}$
	$C^1_{61}=A_{63}B_{31}+A_{64}B_{41}$	$C^1_{62}= A_{63}B_{32}+A_{64}B_{42}$	$C^1_{67}= A_{63}B_{37}+A_{64}B_{47}$	$C^1_{68}= A_{63}B_{38}+A_{64}B_{48}$
	$C^1_{91}=A_{93}B_{31}+A_{94}B_{41}$	$C^1_{92}= A_{93}B_{32}+A_{94}B_{42}$	$C^1_{97}= A_{93}B_{37}+A_{94}B_{47}$	$C^1_{98}= A_{93}B_{38}+A_{94}B_{48}$

Therefore, at the end of this iteration the process (0,0) already has calculated

$C_{11}=A_{11}B_{11} + A_{12}B_{21}+ A_{13}B_{31} +A_{14}B_{41}$
$C_{12}=A_{11}B_{12} + A_{12}B_{22}+ A_{13}B_{32} +A_{14}B_{42}$
$C_{17}=A_{11}B_{17} + A_{12}B_{27}+ A_{13}B_{37} +A_{14}B_{47}$
$C_{18}=A_{11}B_{18} + A_{12}B_{28}+ A_{13}B_{38} +A_{14}B_{48}$

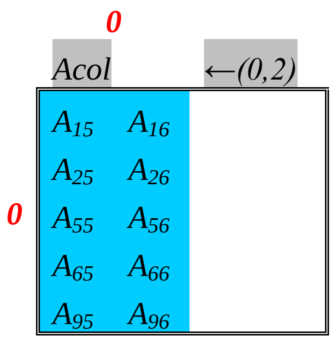
$C_{21}=A_{21}B_{11} + A_{22}B_{21}+ A_{23}B_{31} +A_{24}B_{41}$
$C_{22}=A_{21}B_{12} + A_{22}B_{22}+ A_{23}B_{32} +A_{24}B_{42}$
$C_{27}=A_{21}B_{17} + A_{22}B_{27}+ A_{23}B_{37} +A_{24}B_{47}$
$C_{28}=A_{21}B_{18}+ A_{22}B_{28}+ A_{23}B_{38}+A_{24}B_{48}$

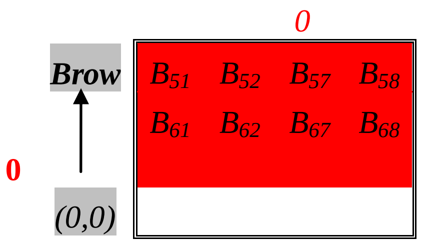
$C_{51}=A_{51}B_{11} + A_{52}B_{21}+ A_{53}B_{31} +A_{54}B_{41}$
$C_{52}=A_{51}B_{12} + A_{52}B_{22}+ A_{53}B_{32} +A_{54}B_{42}$
$C_{57}=A_{51}B_{17} + A_{52}B_{27}+ A_{53}B_{37} +A_{54}B_{47}$
$C_{58}=A_{51}B_{18}+ A_{52}B_{28}+ A_{53}B_{38}+A_{54}B_{48}$

$C_{61}=A_{61}B_{11} + A_{62}B_{21}+ A_{63}B_{31} +A_{64}B_{41}$
$C_{62}=A_{61}B_{12} + A_{62}B_{22}+ A_{63}B_{32} +A_{64}B_{42}$
$C_{67}=A_{61}B_{17} + A_{62}B_{27}+ A_{63}B_{37} +A_{64}B_{47}$
$C_{68}=A_{61}B_{18}+ A_{62}B_{28}+ A_{63}B_{38}+A_{64}B_{48}$

$C_{91}=A_{91}B_{11} + A_{92}B_{21}+ A_{93}B_{31} +A_{94}B_{41}$
$C_{92}=A_{91}B_{12} + A_{92}B_{22}+ A_{93}B_{32} +A_{94}B_{42}$
$C_{97}=A_{91}B_{17} + A_{92}B_{27}+ A_{93}B_{37} +A_{94}B_{47}$
$C_{98}=A_{91}B_{18}+ A_{92}B_{28}+ A_{93}B_{38}+A_{94}B_{48}$

- Iteration 2. Here we have for *Acol* and *Bcol* the following:





0
Brow
0
(0,0)
B_{51} B_{52} B_{57} B_{58}
B_{61} B_{62} B_{67} B_{68}

Then it is possible to calculate the product $C_{(0,0)}= C_{(0,0)}$+Acol*Brow. So we obtain

$C^2_{(0,0)}=$	$C^2_{1,1}= A_{15}B_{51}+ A_{16}B_{61}$	$C^2_{1,2} = A_{15}B_{52}+ A_{16}B_{62}$	$C^2_{1,7}= A_{15}B_{57}+ A_{16}B_{67}$	$C^2_{1,8}= A_{15}B_{58}+ A_{16}B_{68}$
	$C^2_{2,1}=A_{25}B_{51}+A_{26}B_{61}$	$C^2_{22}= A_{25}B_{52}+A_{26}B_{62}$	$C^2_{2,7}= A_{25}B_{57}+A_{26}B_{67}$	$C^2_{2,8}= A_{25}B_{58}+A_{26}B_{68}$
	$C^2_{5,1}=A_{55}B_{51}+A_{56}B_{61}$	$C^2_{52}= A_{55}B_{52}+A_{56}B_{62}$	$C^2_{57}= A_{55}B_{57}+A_{56}B_{67}$	$C^2_{58}= A_{55}B_{58}+A_{56}B_{68}$
	$C^2_{61}=A_{65}B_{51}+A_{66}B_{61}$	$C^2_{62}= A_{65}B_{52}+A_{66}B_{62}$	$C^2_{67}= A_{65}B_{57}+A_{66}B_{67}$	$C^2_{68}= A_{65}B_{58}+A_{66}B_{68}$
	$C^2_{91}=A_{95}B_{51}+A_{96}B_{61}$	$C^2_{92}= A_{95}B_{52}+A_{96}B_{62}$	$C^2_{97}= A_{95}B_{57}+A_{96}B_{67}$	$C^2_{98}= A_{95}B_{58}+A_{96}B_{68}$

Therefore, at the end of this iteration the process (0,0) already has calculated

$C_{11}=A_{11}B_{11} + A_{12}B_{21}+ A_{13}B_{31} +A_{14}B_{41} +A_{15}B_{51}+ A_{16}B_{61}$
$C_{12}=A_{11}B_{12} + A_{12}B_{22}+ A_{13}B_{32} +A_{14}B_{42} +A_{15}B_{52}+ A_{16}B_{62}$
$C_{17}=A_{11}B_{17} + A_{12}B_{27}+ A_{13}B_{37} +A_{14}B_{47} +A_{15}B_{57}+ A_{16}B_{67}$
$C_{18}=A_{11}B_{18} + A_{12}B_{28}+ A_{13}B_{38} +A_{14}B_{48} +A_{15}B_{58}+ A_{16}B_{68}$

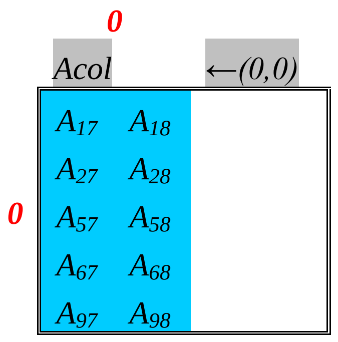
$C_{21}=A_{21}B_{11} + A_{22}B_{21}+ A_{23}B_{31} +A_{24}B_{41} +A_{25}B_{51}+ A_{26}B_{61}$
$C_{22}=A_{21}B_{12} + A_{22}B_{22}+ A_{23}B_{32} +A_{24}B_{42} +A_{25}B_{52}+ A_{26}B_{62}$
$C_{27}=A_{21}B_{17} + A_{22}B_{27}+ A_{23}B_{37} +A_{24}B_{47} +A_{25}B_{57}+ A_{26}B_{67}$
$C_{28}=A_{21}B_{18}+ A_{22}B_{28}+ A_{23}B_{38}+A_{24}B_{48} +A_{25}B_{58}+ A_{26}B_{68}$

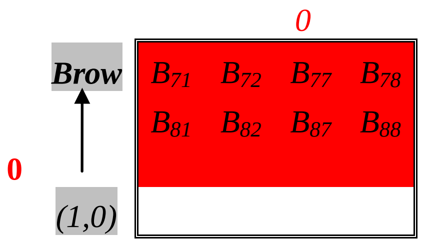
$C_{51}=A_{51}B_{11} + A_{52}B_{21}+ A_{53}B_{31} +A_{54}B_{41} +A_{55}B_{51}+ A_{56}B_{61}$
$C_{52}=A_{51}B_{12} + A_{52}B_{22}+ A_{53}B_{32} +A_{54}B_{42} +A_{55}B_{52}+ A_{56}B_{62}$
$C_{57}=A_{51}B_{17} + A_{52}B_{27}+ A_{53}B_{37} +A_{54}B_{47} +A_{55}B_{57}+ A_{56}B_{67}$
$C_{58}=A_{51}B_{18}+ A_{52}B_{28}+ A_{53}B_{38}+A_{54}B_{48} +A_{55}B_{58}+ A_{56}B_{68}$

$C_{61}=A_{61}B_{11} + A_{62}B_{21}+ A_{63}B_{31} +A_{64}B_{41} +A_{65}B_{51}+ A_{66}B_{61}$
$C_{62}=A_{61}B_{12} + A_{62}B_{22}+ A_{63}B_{32} +A_{64}B_{42} +A_{65}B_{52}+ A_{66}B_{62}$
$C_{67}=A_{61}B_{17} + A_{62}B_{27}+ A_{63}B_{37} +A_{64}B_{47} +A_{65}B_{57}+ A_{66}B_{67}$
$C_{68}=A_{61}B_{18}+ A_{62}B_{28}+ A_{63}B_{38}+A_{64}B_{48} +A_{65}B_{58}+ A_{66}B_{68}$

$C_{91}=A_{91}B_{11} + A_{92}B_{21}+ A_{93}B_{31} +A_{94}B_{41} +A_{95}B_{51}+ A_{96}B_{61}$
$C_{92}=A_{91}B_{12} + A_{92}B_{22}+ A_{93}B_{32} +A_{94}B_{42} +A_{95}B_{52}+ A_{96}B_{62}$
$C_{97}=A_{91}B_{17} + A_{92}B_{27}+ A_{93}B_{37} +A_{94}B_{47} +A_{95}B_{57}+ A_{96}B_{67}$
$C_{98}=A_{91}B_{18}+ A_{92}B_{28}+ A_{93}B_{38}+A_{94}B_{48} +A_{95}B_{58}+ A_{96}B_{68}$

- Iteration 3. Here we have for Acol and Bcol the following:




0
Brow
0
(1,0)
B_{71} B_{72} B_{77} B_{78}
B_{81} B_{82} B_{87} B_{88}

Then it is possible to calculate the product $C_{(0,0)}= C_{(0,0)}$+Acol*Brow. So we obtain

	$C^3_{1,1}=A_{17}B_{71}+A_{18}B_{81}$	$C^3_{1,2}=A_{17}B_{72}+A_{18}B_{82}$	$C^3_{1,7}=A_{17}B_{77}+A_{18}B_{87}$	$C^3_{1,8}=A_{17}B_{78}+A_{18}B_{88}$
	$C^3_{2,1}=A_{27}B_{71}+A_{28}B_{81}$	$C^3_{22}=A_{27}B_{72}+A_{28}B_{82}$	$C^3_{2,7}=A_{27}B_{77}+A_{28}B_{87}$	$C^3_{2,8}=A_{27}B_{78}+A_{28}B_{88}$
$C^3_{(0,0)}=$	$C^3_{5,1}=A_{57}B_{71}+A_{58}B_{81}$	$C^3_{52}=A_{57}B_{72}+A_{58}B_{82}$	$C^3_{57}=A_{57}B_{77}+A_{58}B_{87}$	$C^3_{58}=A_{57}B_{78}+A_{58}B_{88}$
	$C^3_{61}=A_{67}B_{71}+A_{68}B_{81}$	$C^3_{62}=A_{67}B_{72}+A_{68}B_{82}$	$C^3_{67}=A_{67}B_{77}+A_{68}B_{87}$	$C^3_{68}=A_{67}B_{78}+A_{68}B_{88}$
	$C^3_{91}=A_{97}B_{71}+A_{98}B_{81}$	$C^3_{92}=A_{97}B_{72}+A_{98}B_{82}$	$C^3_{97}=A_{97}B_{77}+A_{98}B_{87}$	$C^3_{98}=A_{97}B_{78}+A_{98}B_{88}$

Therefore, at the end of this iteration the process (0,0) already has calculated

$C_{11}=A_{11}B_{11} + A_{12}B_{21}+ A_{13}B_{31} +A_{14}B_{41} +A_{15}B_{51}+ A_{16}B_{61}+ A_{17}B_{71} +A_{18}B_{81}$
$C_{12}=A_{11}B_{12} + A_{12}B_{22}+ A_{13}B_{32} +A_{14}B_{42} +A_{15}B_{52}+ A_{16}B_{62}+ A_{17}B_{72} +A_{18}B_{82}$
$C_{17}=A_{11}B_{17} + A_{12}B_{27}+ A_{13}B_{37} +A_{14}B_{47} +A_{15}B_{57}+ A_{16}B_{67}+ A_{17}B_{77} +A_{18}B_{87}$
$C_{18}=A_{11}B_{18} + A_{12}B_{28}+ A_{13}B_{38} +A_{14}B_{48} +A_{15}B_{58}+ A_{16}B_{68}+ A_{17}B_{78} +A_{18}B_{88}$

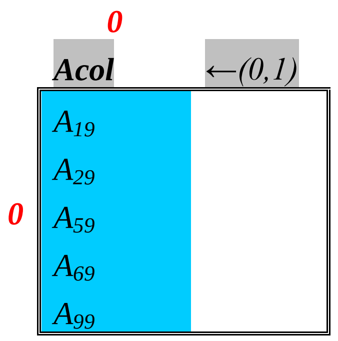
$C_{21}=A_{21}B_{11} + A_{22}B_{21}+ A_{23}B_{31} +A_{24}B_{41} +A_{25}B_{51}+ A_{26}B_{61}+ A_{27}B_{71} +A_{28}B_{81}$
$C_{22}=A_{21}B_{12} + A_{22}B_{22}+ A_{23}B_{32} +A_{24}B_{42} +A_{25}B_{52}+ A_{26}B_{62}+ A_{27}B_{72} +A_{28}B_{82}$
$C_{27}=A_{21}B_{17} + A_{22}B_{27}+ A_{23}B_{37} +A_{24}B_{47} +A_{25}B_{57}+ A_{26}B_{67}+ A_{27}B_{77} +A_{28}B_{87}$
$C_{28}=A_{21}B_{18}+ A_{22}B_{28}+ A_{23}B_{38}+A_{24}B_{48} +A_{25}B_{58}+ A_{26}B_{68}+ A_{27}B_{77} +A_{28}B_{88}$

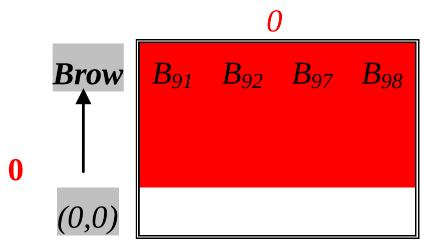
$C_{51}=A_{51}B_{11} + A_{52}B_{21}+ A_{53}B_{31} +A_{54}B_{41} +A_{55}B_{51}+ A_{56}B_{61}+ A_{57}B_{71} +A_{58}B_{81}$
$C_{52}=A_{51}B_{12} + A_{52}B_{22}+ A_{53}B_{32} +A_{54}B_{42} +A_{55}B_{52}+ A_{56}B_{62}+ A_{57}B_{72} +A_{58}B_{82}$
$C_{57}=A_{51}B_{17} + A_{52}B_{27}+ A_{53}B_{37} +A_{54}B_{47} +A_{55}B_{57}+ A_{56}B_{67}+ A_{57}B_{77} +A_{58}B_{87}$
$C_{58}=A_{51}B_{18}+ A_{52}B_{28}+ A_{53}B_{38}+A_{54}B_{48} +A_{55}B_{58}+ A_{56}B_{68}+ A_{57}B_{78} +A_{58}B_{88}$

$C_{61}=A_{61}B_{11} + A_{62}B_{21}+ A_{63}B_{31} +A_{64}B_{41} +A_{65}B_{51}+ A_{66}B_{61}+ A_{67}B_{71} +A_{68}B_{81}$
$C_{62}=A_{61}B_{12} + A_{62}B_{22}+ A_{63}B_{32} +A_{64}B_{42} +A_{65}B_{52}+ A_{66}B_{62}+ A_{67}B_{72} +A_{68}B_{82}$
$C_{67}=A_{61}B_{17} + A_{62}B_{27}+ A_{63}B_{37} +A_{64}B_{47} +A_{65}B_{57}+ A_{66}B_{67}+ A_{67}B_{77} +A_{68}B_{87}$
$C_{68}=A_{61}B_{18}+ A_{62}B_{28}+ A_{63}B_{38}+A_{64}B_{48} +A_{65}B_{58}+ A_{66}B_{68}+ A_{67}B_{78} +A_{68}B_{88}$

$C_{91}=A_{91}B_{11} + A_{92}B_{21}+ A_{93}B_{31} +A_{94}B_{41} +A_{95}B_{51}+ A_{96}B_{61}+ A_{97}B_{71} +A_{98}B_{81}$
$C_{92}=A_{91}B_{12} + A_{92}B_{22}+ A_{93}B_{32} +A_{94}B_{42} +A_{95}B_{52}+ A_{96}B_{62}+ A_{97}B_{72} +A_{98}B_{82}$
$C_{97}=A_{91}B_{17} + A_{92}B_{27}+ A_{93}B_{37} +A_{94}B_{47} +A_{95}B_{57}+ A_{96}B_{67}+ A_{97}B_{77} +A_{98}B_{87}$
$C_{98}=A_{91}B_{18}+ A_{92}B_{28}+ A_{93}B_{38}+A_{94}B_{48} +A_{95}B_{58}+ A_{96}B_{68}+ A_{97}B_{78} +A_{98}B_{88}$

- Iteration 4. Here we have for *Acol* and *Bcol* the following:





Then it is possible to calculate the product C(0,0)= C(0,0)+Acol*Brow. So we obtain

	$C^4_{1,1}= A_{19}B_{91}$	$C^4_{1,2} = A_{19}B_{92}$	$C^4_{1,7}= A_{19}B_{97}$	$C^4_{1,8}= A_{19}B_{98}$
	$C^4_{2,1}=A_{29}B_{91}$	$C^4_{22}= A_{29}B_{92}$	$C^4_{2,7}= A_{29}B_{97}$	$C^4_{2,8}= A_{29}B_{98}$
$C^4_{(0,0)}=$	$C^4_{5,1}=A_{59}B_{91}$	$C^4_{52}= A_{59}B_{92}$	$C^4_{57}= A_{59}B_{97}$	$C^4_{58}= A_{59}B_{98}$
	$C^4_{61}=A_{69}B_{91}$	$C^4_{62}= A_{69}B_{92}$	$C^4_{67}= A_{69}B_{97}$	$C^4_{68}= A_{69}B_{98}$
	$C^4_{91}=A_{99}B_{91}$	$C^4_{92}= A_{99}B_{92}$	$C^4_{97}= A_{99}B_{97}$	$C^4_{98}= A_{99}B_{98}$

Therefore, at the end of this iteration the process (0,0) already has calculated

$$C_{11}=A_{11}B_{11} + A_{12}B_{21}+ A_{13}B_{31} +A_{14}B_{41} +A_{15}B_{51}+ A_{16}B_{61}+ A_{17}B_{71} +A_{18}B_{81} +A_{19}B_{91}$$
$$C_{12}=A_{11}B_{12} + A_{12}B_{22}+ A_{13}B_{32} +A_{14}B_{42} +A_{15}B_{52}+ A_{16}B_{62}+ A_{17}B_{72} +A_{18}B_{82} +A_{19}B_{92}$$
$$C_{17}=A_{11}B_{17} + A_{12}B_{27}+ A_{13}B_{37} +A_{14}B_{47} +A_{15}B_{57}+ A_{16}B_{67}+ A_{17}B_{77} +A_{18}B_{87} +A_{19}B_{97}$$
$$C_{18}=A_{11}B_{18} + A_{12}B_{28}+ A_{13}B_{38} +A_{14}B_{48} +A_{15}B_{58}+ A_{16}B_{68}+ A_{17}B_{78} +A_{18}B_{88} +A_{19}B_{98}$$

$$C_{21}=A_{21}B_{11} + A_{22}B_{21}+ A_{23}B_{31} +A_{24}B_{41} +A_{25}B_{51}+ A_{26}B_{61}+ A_{27}B_{71} +A_{28}B_{81} +A_{29}B_{91}$$
$$C_{22}=A_{21}B_{12} + A_{22}B_{22}+ A_{23}B_{32} +A_{24}B_{42} +A_{25}B_{52}+ A_{26}B_{62}+ A_{27}B_{72} +A_{28}B_{82} +A_{29}B_{92}$$
$$C_{27}=A_{21}B_{17} + A_{22}B_{27}+ A_{23}B_{37} +A_{24}B_{47} +A_{25}B_{57}+ A_{26}B_{67}+ A_{27}B_{77} +A_{28}B_{87} +A_{29}B_{97}$$
$$C_{28}=A_{21}B_{18}+ A_{22}B_{28}+ A_{23}B_{38}+A_{24}B_{48} +A_{25}B_{58}+ A_{26}B_{68}+ A_{27}B_{77} +A_{28}B_{88} +A_{29}B_{98}$$

$$C_{51}=A_{51}B_{11} + A_{52}B_{21}+ A_{53}B_{31} +A_{54}B_{41} +A_{55}B_{51}+ A_{56}B_{61}+ A_{57}B_{71} +A_{58}B_{81} +A_{59}B_{91}$$
$$C_{52}=A_{51}B_{12} + A_{52}B_{22}+ A_{53}B_{32} +A_{54}B_{42} +A_{55}B_{52}+ A_{56}B_{62}+ A_{57}B_{72} +A_{58}B_{82} +A_{59}B_{92}$$
$$C_{57}=A_{51}B_{17} + A_{52}B_{27}+ A_{53}B_{37} +A_{54}B_{47} +A_{55}B_{57}+ A_{56}B_{67}+ A_{57}B_{77} +A_{58}B_{87} +A_{59}B_{97}$$
$$C_{58}=A_{51}B_{18}+ A_{52}B_{28}+ A_{53}B_{38}+A_{54}B_{48} +A_{55}B_{58}+ A_{56}B_{68}+ A_{57}B_{78} +A_{58}B_{88} +A_{59}B_{98}$$

$$C_{61}=A_{61}B_{11} + A_{62}B_{21}+ A_{63}B_{31} +A_{64}B_{41} +A_{65}B_{51}+ A_{66}B_{61}+ A_{67}B_{71} +A_{68}B_{81} +A_{69}B_{91}$$
$$C_{62}=A_{61}B_{12} + A_{62}B_{22}+ A_{63}B_{32} +A_{64}B_{42} +A_{65}B_{52}+ A_{66}B_{62}+ A_{67}B_{72} +A_{68}B_{82} +A_{69}B_{92}$$
$$C_{67}=A_{61}B_{17} + A_{62}B_{27}+ A_{63}B_{37} +A_{64}B_{47} +A_{65}B_{57}+ A_{66}B_{67}+ A_{67}B_{77} +A_{68}B_{87} +A_{69}B_{97}$$
$$C_{68}=A_{61}B_{18}+ A_{62}B_{28}+ A_{63}B_{38}+A_{64}B_{48} +A_{65}B_{58}+ A_{66}B_{68}+ A_{67}B_{77} +A_{68}B_{88} +A_{69}B_{98}$$

$$C_{91}=A_{91}B_{11} + A_{92}B_{21}+ A_{93}B_{31} +A_{94}B_{41} +A_{95}B_{51}+ A_{96}B_{61}+ A_{97}B_{71} +A_{98}B_{81} +A_{99}B_{91}$$
$$C_{92}=A_{91}B_{12} + A_{92}B_{22}+ A_{93}B_{32} +A_{94}B_{42} +A_{95}B_{52}+ A_{96}B_{62}+ A_{97}B_{72} +A_{98}B_{82} +A_{99}B_{92}$$
$$C_{97}=A_{91}B_{17} + A_{92}B_{27}+ A_{93}B_{37} +A_{94}B_{47} +A_{95}B_{57}+ A_{96}B_{67}+ A_{97}B_{77} +A_{98}B_{87} +A_{99}B_{97}$$
$$C_{98}=A_{91}B_{18}+ A_{92}B_{28}+ A_{93}B_{38}+A_{94}B_{48} +A_{95}B_{58}+ A_{96}B_{68}+ A_{97}B_{78} +A_{98}B_{88} +A_{99}B_{98}$$

Performance Analysis

On a 2-dimensional *PxQ* processor grid, the communication time of SUMMA is doubled in order to broadcast T_B rows as well as T_A columns *[aici indicele semnifica matricea]*. Assume that the time for sending a column T_A and a row T_B to the next processor are t_{ca} and t_{cb}, respectively, and the time for multiplying T_A with T_B and adding the product to ***C*** is t_p . Using the Hockney model, transferring time is modeled by **$t_s=\alpha+\beta m$**, where **α** is the latency for each message, and **β** is the transfer time per byte (or reciprocal of network bandwidth) we obtained that

$$t_{ca} = \alpha + \left(\frac{M}{P} k_b\right) \cdot \beta \text{ , } t_{cb} = \alpha + \left(\frac{N}{Q} k_b\right) \cdot \beta \text{ .}$$

The $t_p = 2\left(\frac{M}{P} \times \frac{N}{Q} k_b\right)\gamma$.

So,

$$t^{2D}_{summa} = K_g\left(2t_{ca} + 2t_{cb} + t_p\right) - t_{ca} + (Q-2)t_{ca} - t_{cb} + (P-2)t_{cb} =$$

$$= K_g\left(2t_{ca} + 2t_{cb} + t_p\right) + (Q-3)t_{ca} + (P-3)t_{cb} \text{ .}$$

Here where α is a communication start-up time, β is a data transfer time, and γ is a time for multiplication or addition, k_b is a block sizes, *N*-number of the rows in the matrix ***C***, M-number of the columns in the matrix ***C***, $K_g = \lceil K/k_b \rceil$ -columns of blocks of ***A*** and K_g rows of blocks of ***B***.

Can we reduce the communication cost somehow? Obvious improvement [*îmbunătățire evidentă*]: instead of broadcasting single rows and columns, do block rows and columns.

```
PvGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,IA,JA,DESCA,B,IB,JB,DESCB,BET
        A,C,IC,JC,DESCC)
void pdgemm_(char *TRANSA,char *TRANSB,int *M,int *N,int *K,double
        *ALPHA,double *A,int *IA,int *JA,int *DESCA,double *B,int *
        IB,int *JB,int *DESCB,double *BETA,double *C, int *IC,int *JC,
        int *DESCC );
```

Purpose: PvGEMM performs one of the matrix-matrix operations

sub(C) := alpha × op(sub(A)) × op(sub(B)) + beta × sub(C),

where sub(C) denotes C(IC:IC+M-1,JC:JC+N-1),

$$op(sub(A)) = \begin{cases} A(IA:IA+M-1, JA:JA+K-1) & \text{if } TRANSA = 'N' \\ A(IA:IA+K-1, JA:JA+M-1)' & \text{if } TRANSA = 'T \\ A(IA:IA+K-1, JA:JA+M-1)' & \text{if } TRANSA = '' \end{cases}$$

$$op(sub(B)) = \begin{cases} B(IB:IB+K-1, JB:JB+N-1) & \text{if } TRANSB = 'N' \\ B(IB:IB+N-1, JB:JB+K-1)' & \text{if } TRANSB = 'T \\ B(IB:IB+N-1, JB:JB+K-1)' & \text{if } TRANSB = 'C' \end{cases}$$

alpha and *beta* are scalars, and *sub(A)*, *sub(B)* and *sub(C)* are distributed matrices, with *op(sub(A))* an *M*-by-*K* distributed matrix, *op(sub(B))* a *K*-by-*N* distributed matrix and *sub(C)* an *M*-by-*N* distributed matrix.

Arguments

```
TRANSA (global input) CHARACTER
```

The form of op(A) to be used in the matrix multiplication as follows:

TRANSA = 'N', op(A) = A,

TRANSA = 'T', op(A) = A^T,

TRANSA = 'C', op(A) = A^T.

TRANSB (global input) CHARACTER

The form of op(B) to be used in the matrix multiplication as follows:
TRANSB = 'N', op(B) = B,
TRANSB = 'T', op(B) = B^T,
TRANSB = 'C', op(B) = B^T.

M (global input) INTEGER

The number of rows of the distributed matrices op(sub(A)) and sub(C). M≥ 0.

N (global input) INTEGER

The number of columns of the distributed matrices op(sub(B)) and sub(C). N ≥0.

K (global input) INTEGER

The number of columns of the distributed matrix op(sub(A)) and the number of rows of the distributed matrix op(B). K≥0.

ALPHA (global input) REAL/COMPLEX

On entry, ALPHA specifies the scalar alpha.

A (local input) array of dimension (LLD_A, KLa)

where KLa is LOCq(JA+K-1) when TRANSA = 'N', and is LOCq(JA+M-1) otherwise. Before entry, this array must contain the local pieces of the distributed matrix sub(A).

IA (global input) INTEGER

The global row index of the submatrix of the distributed matrix A to operate on.

JA (global input) INTEGER

The global column index of the submatrix of the distributed matrix A to operate on.

DESCA (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix A.

B (local input) array of dimension (LLD_B, KLb)

where KLb is LOC_q(JB+N-1) when TRANSB = 'N', and is LOC_q(JB+K-1) otherwise. Before entry this array must contain the local pieces of the distributed matrix sub(B).
IB (global input) INTEGER
The global row index of the submatrix of the distributed matrix B to operate on.

JB (global input) INTEGER

The global column index of the submatrix of the distributed matrix B to operate on.

DESCB (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix B.

BETA (global input) REAL/COMPLEX

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then sub(Y) need not be set on input.

C (local input/local output) array of dimension (LLD_C, LOC_q(JC+N-1))

*Before entry, this array must contain the local pieces of the distributed matrix sub(C). On exit, the distributed matrix sub(C) is overwritten by the M-by-N distributed matrix alpha*op(sub(A))*op(sub(B)) + beta*sub(C).*

IC (global input) INTEGER

The global row index of the submatrix of the distributed matrix C to operate on.

JC (global input) INTEGER

The global column index of the submatrix of the distributed matrix C to operate on.

DESCC (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix C.

***Example 2.9.1**. The using of function pdgemm_*

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "mpi.h"
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
using namespace std;
#define AA(i,j) AA[(i)*m+(j)]
#define BB(i,j) BB[(i)*k+(j)]
#define CC(i,j) CC[(i)*m+(j)]
extern "C"
{
void Cblacs_pinfo( int* mypnum, int* nprocs);
void Cblacs_get( int context, int request, int* value);
int  Cblacs_gridinit( int* context, char * order, int np_row, int np_col);
void Cblacs_gridinfo( int context, int*  np_row, int* np_col, int*  my_row, int*
                    my_col);
void Cblacs_gridexit( int context);
void Cblacs_barrier(int, const char*);
void Cblacs_exit( int error_code);
void Cblacs_pcoord(int, int, int*, int*);
int  numroc_( int *n, int *nb, int *iproc, int *isrcproc, int *nprocs);
int indxl2g_(int*, int*, int*, int*, int*);
void descinit_(int *desc, int *m, int *n, int *mb, int *nb, int *irsrc, int *icsrc,
              int *ictxt, int *lld, int *info);
void pdgemm_(char *TRANSA,char *TRANSB,int *M,int *N,int *K,double *ALPHA, double
             *A,int *IA,int *JA,int *DESCA, double * B, int * IB, int * JB, int
             *DESCB,double *BETA,double *C, int *IC, int *JC, int *DESCC );
} // extern "C"
int main(int argc, char **argv)
{
int i, j;
int iam,nprocs,nprow,npcol,myrow,mycol;
int descA[9],descB[9],descC[9];
int m,n,k,mb,nb,rsrc,csrc,ictxt,llda,lldb,lldc,info;
int lm,ln,lk;
int iloc,jloc;
int ZERO=0,ONE=1;
double alpha, beta;
m=5; n=5; k=5;
mb=2; nb=2;
double *AA = (double*)malloc(m*k*sizeof(double));// matricea "globala" pentru
             inmultirea AA*BB=CC
double *BB = (double*) malloc(k*n*sizeof(double));//matricea "globala" pentru
             inmultirea AA*BB=CC
double *CC = (double*) malloc(m*n*sizeof(double));//matricea "globala" pentru
             inmultirea AA*BB=CC
// initializarea marticelor globale
for(i=0;i<m;i++ )
     for(j=0;j<k;j++)
            AA[i*m+j]=(i+j);
for(i=0;i<k;i++ )
     for(j=0;j<n;j++)
            BB[i*k+j]=j+i;
nprow=2; npcol=2; // Astfel, programul se executa pe 4 procese
Cblacs_pinfo(&iam,&nprocs);
Cblacs_get(-1, 0, &ictxt);
```

```
Cblacs_gridinit( &ictxt,"Row",nprow,npcol);
Cblacs_gridinfo(ictxt,&nprow,&npcol,&myrow,&mycol);
rsrc=0; csrc=0;
  if (iam==0)
  {
  printf("=========== REZULT OF THE PROGRAM %s \n",argv[0]);
  cout << "Global matrix AA:\n";
              for (i = 0; i < m; ++i) {
                  for (j = 0; j < n; ++j) {
                      cout << setw(3) << *(AA + m*i + j) << " ";
                  }
                  cout << "\n";
              }
              cout << endl;
  cout << "Global matrix BB:\n";
              for (i = 0; i < k; ++i) {
                  for (j = 0; j < n; ++j) {
                      cout << setw(3) << *(BB + k*i + j) << " ";
                  }
                  cout << "\n";
              }
              cout << endl;
  }
Cblacs_barrier(ictxt, "All");
int mA=numroc_( &m, &mb, &myrow, &rsrc, &nprow );
int kA = numroc_( &k, &nb, &mycol, &rsrc, &npcol );
int kB = numroc_( &k, &mb, &myrow, &rsrc, &nprow );
int nB = numroc_( &n, &nb, &mycol, &rsrc, &npcol );
int mC = numroc_( &m, &mb, &myrow, &rsrc, &nprow );
int nC = numroc_( &n, &nb, &mycol, &rsrc, &nprow );
descinit_(descA, &m,  &k,  &mb,  &nb,  &rsrc, &csrc, &ictxt, &mA,  &info);
descinit_(descB, &k,  &n,  &mb,  &nb,  &rsrc, &csrc, &ictxt, &kB,  &info);
descinit_(descC, &m,  &n,  &mb,  &nb,  &rsrc, &csrc, &ictxt, &mC, &info);
double *A=(double*) malloc(mA*kA*sizeof(double)); //matricea locala
double *B = (double*) malloc(kB*nB*sizeof(double)); //matricea locala
double *C = (double*) malloc(mC*nC*sizeof(double)); //matricea locala
// se complecteaza cu valori matricele locale folosind algoritmul 2D-ciclic
for(iloc=0;iloc<mA;iloc++)
   for(jloc=0;jloc<kA;jloc++){
   int fortidl = iloc + 1;
   int fortjdl = jloc + 1;
   i = indxl2g_(&fortidl, &mb, &myrow, &ZERO, &nprow)-1;
   j = indxl2g_(&fortjdl, &nb, &mycol, &ZERO, &npcol)-1;
   A[jloc*mA+iloc]=AA(i,j);
                             }
for(iloc=0;iloc<kB;iloc++)
   for(jloc=0;jloc<nB;jloc++){
   int fortidl = iloc + 1;
   int fortjdl = jloc + 1;
   i = indxl2g_(&fortidl, &mb, &myrow, &ZERO, &nprow)-1;
   j = indxl2g_(&fortjdl, &nb, &mycol, &ZERO, &npcol)-1;
   B[jloc*kB+iloc]=BB(i,j);
                             }
alpha = 1.0; beta = 0.0;
pdgemm_("No Transpose","No Transpose",&m,&n,&k,&alpha,A,&ONE,&ONE, descA,
       B,&ONE,&ONE,descB,&beta,C,&ONE,&ONE,descC);
Cblacs_barrier(ictxt, "All");
// Print out the matrix product C
for (int id = 0; id < nprocs; ++id)
{
if (id == iam) {
```

```
cout << "C_loc on node " << iam << endl;
for (i = 0; i < mC; ++i)
 {
  for (j = 0; j < nC; ++j)
  cout << setw(3) << *(C+mC*j+i) << " ";
  cout << endl;
 }
Cblacs_barrier(ictxt, "All");
}
}
// Se construieste matricea globala CC
for(iloc=0;iloc<mC;iloc++)
   for(jloc=0;jloc<nC;jloc++){
   int fortidl = iloc + 1;
   int fortjdl = jloc + 1;
   i = indxl2g_(&fortidl, &mb, &myrow, &ZERO, &nprow)-1;
   j = indxl2g_(&fortjdl, &nb, &mycol, &ZERO, &npcol)-1;
   CC(i,j)=C[jloc*mC+iloc];
                             }
Cblacs_barrier(ictxt, "All");
//*** Tipar rezultate finale
for (int id = 0; id < nprocs; ++id)
{
  if (id == iam) {
 cout << "Global matrix CC on node "<< iam << endl;
              for (i = 0; i < m; ++i) {
                  for (j = 0; j < n; ++j)
                      cout << setw(3) << *(CC + m*i + j) << " ";
                  cout << "\n";
              }
              cout << endl;
 }
Cblacs_barrier(ictxt, "All");
}
/* Free memory */
    free(A);
    free(B);
    free(C);
    /* Release process grid */
    Cblacs_gridexit(ictxt);
} /* main */
```

The results are the following

```
[MI_gr_TPS1@hpc]$./mpiCC_ScL -o Example2.9.1.exe Example2.9.1.cpp
[MI_gr_TPS1@hpc]$/opt/openmpi/bin/mpirun -n 4 -host compute-0-0,compute-0-1
                Example2.9.1.exe
[MI_gr_TPS1@hpc]$./mpiCC_ScL -o pmm.exe pmm.cpp
[MI_gr_TPS1@hpc]$/opt/openmpi/bin/mpirun -n 4 -host compute-0-0,compute-0-1 pmm.exe

Global matrix AA:
  0   1   2   3   4
  1   2   3   4   5
  2   3   4   5   6
  3   4   5   6   7
  4   5   6   7   8
Global matrix BB:
  0   1   2   3   4
  1   2   3   4   5
  2   3   4   5   6
  3   4   5   6   7
  4   5   6   7   8
```

```
C_loc on node 0
 30  40  70
 40  55 100
 70 100 190
C_loc on node 1
 50  60
 70  85
130 160
C_loc on node 2
 50  70 130
 60  85 160
C_loc on node 3
 90 110
110 135
Global matrix CC on node 0
 30  40   0   0  70
 40  55   0   0 100
  0   0   0   0   0
  0   0   0   0   0
 70 100   0   0 190
Global matrix CC on node 1
  0   0  50  60   0
  0   0  70  85   0
  0   0   0   0   0
  0   0   0   0   0
  0   0 130 160   0
Global matrix CC on node 2
  0   0   0   0   0
  0   0   0   0   0
 50  70   0   0 130
 60  85   0   0 160
  0   0   0   0   0
Global matrix CC on node 3
  0   0   0   0   0
  0   0   0   0   0
  0   0  90 110   0
  0   0 110 135   0
  0   0   0   0   0
```

To verify the results we use the sequential algorithm based on function dgemm. The program is presented below:

```
#include <string>
#include <iostream>
#include <stdio.h>
#include <math.h>
#include <iomanip>
#include <fstream>
#include <sstream>
using namespace std;

extern "C" {
 void dgemm_(const char &transa, const char &transb,
              const int &l, const int &n, const int &m, const double &alpha,
              const double *a, const int &lda, const double *b, const int &ldb,
              const double &beta, double *c, const int &ldc);
}
int main() {
int i,j,m=5,k=5,n=5;
  //Matrizen werden als eindimensionale Arrays gespeichert
  double *AA = new double[m*k];
  double *BB = new double[k*n];
  double *CC = new double[m*n];
  //Matrizen fuelln
for(i=0;i<m;i++ )
     for(j=0;j<k;j++)
            AA[i*k+j]=i+j; //(10*i+j);
for(i=0;i<k;i++ )
     for(j=0;j<n;j++)
            BB[i*n+j]=i+j;//AA[i*5+j]+5;

for( i=0 ; i<m*n ; i++)
    CC[i] = 0.0;
  cout << "Global matrix AA:\n";
              for (i = 0; i < m; ++i) {
                  for (j = 0; j < k; ++j) {
                      cout << setw(3) << *(AA + k*i + j) << " ";
                  }
                  cout << "\n";
              }
              cout << endl;
  cout << "Global matrix BB:\n";
              for (i = 0; i < k; ++i) {
                  for (j = 0; j < n; ++j) {
                      cout << setw(3) << *(BB + n*i + j) << " ";
                  }
                  cout << "\n";
              }
              cout << endl;
  double alpha = 1.0;
  double beta = 0.0;
  //dgemm berechnet C=alpha*A*B+beta*C
  dgemm_('N','N',m,n,k,alpha,AA,m,BB,k,beta,CC,m);

  cout << "Global matrix CC:\n " << endl;
  for(i=0 ; i<m ; i++) {
    for(j=0 ; j<n ; j++)
cout << setw(3) << *(CC + n*i + j) << " ";
    cout << endl;
  }
  cout << endl;

  delete[] AA;
  delete[] BB;
  delete[] CC;

  return 0;
}
```

The results are the following:

```
[MI_gr_TPS1@hpc]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-0-4
dgemm1.exe
Global matrix AA:
  0   1   2   3   4
  1   2   3   4   5
  2   3   4   5   6
  3   4   5   6   7
  4   5   6   7   8

Global matrix BB:
  0   1   2   3   4
  1   2   3   4   5
  2   3   4   5   6
  3   4   5   6   7
  4   5   6   7   8

Global matrix CC:

 30  40  50  60  70
 40  55  70  85 100
 50  70  90 110 130
 60  85 110 135 160
 70 100 130 160 190
```

In the program Example2.9.1.cpp the parallelization on data level is "non-rational" in terms of usage of operative memory, because for all processes the operative memory is reserved for global matrices! Using the function `pdgeadd` we can develop a program, in which the operative memory is used already rationally. The program is presened below.

Example 2.9.2.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "mpi.h"
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
using namespace std;
#define AA(i,j) AA[(i)*n+(j)]
#define BB(i,j) BB[(i)*k+(j)]
#define CC(i,j) CC[(i)*n+(j)]
#define A(i,j) A[(i)*n+(j)]
#define B(i,j) B[(i)*n+(j)]
#define C(i,j) C[(i)*n+(j)]
static int MAX( int a, int b ){
        if (a>b) return(a); else return(b);
                        }
extern "C"
{

void Cblacs_pinfo( int* mypnum, int* nprocs);
void Cblacs_get( int context, int request, int* value);
int  Cblacs_gridinit( int* context, char * order, int np_row, int np_col);
void Cblacs_gridinfo( int context, int*  np_row, int* np_col, int*  my_row, int*
my_col);
void Cblacs_gridexit( int context);
void Cblacs_barrier(int, const char*);
void Cblacs_exit( int error_code);
void Cblacs_pcoord(int, int, int*, int*);
int  numroc_( int *n, int *nb, int *iproc, int *isrcproc, int *nprocs);
int indxl2g_(int*, int*, int*, int*, int*);
void descinit_(int *desc, int *m, int *n, int *mb, int *nb, int *irsrc, int *icsrc,
int *ictxt, int *lld, int *info);
void pdgemm_( char *TRANSA,char *TRANSB,int *M,int *N,int *K,double *ALPHA,double
*A,int *IA,int *JA,int *DESCA,
             double * B, int * IB, int * JB, int * DESCB,double * BETA,double * C,
int * IC, int * JC, int * DESCC );
void pdgeadd_(char *TRANS,int *M, int *N,double * ALPHA,double *A,int *IA,int *JA,int
*DESCA,double *BETA,double *C,
              int *IC,int *JC,int *DESCC);

} // extern "C"

int main(int argc, char **argv)
{
int i, j;
int iam,nprocs,nprow,npcol,myrow,mycol;
// descriptori pentri matrici locale
int descA[9],descB[9],descC[9];
// descriptori pentri matrici globale
int descAA[9],descBB[9],descCC[9];
int m,n,k,mb,nb,rsrc,csrc,ictxt,llda,lldb,lldc,info;
int lm,ln,lk;
int iloc,jloc;
int ZERO=0,ONE=1;
double zero=0.0E+0, one=1.0E+0;
int i_one = 1,  i_zero = 0;
int lld_AA, lld_BB,lld_CC;
double alpha, beta;
double *AA, *BB,*CC,*A,*B,*C, *work,*tau;
m=5; n=5; k=5;
mb=2; nb=2;
nprow=2; npcol=2; // Astfel, programul se executa pe 4 procese
Cblacs_pinfo(&iam,&nprocs);
Cblacs_get(-1, 0, &ictxt);
Cblacs_gridinit( &ictxt,"Row",nprow,npcol);
Cblacs_gridinfo(ictxt,&nprow,&npcol,&myrow,&mycol);
rsrc=0; csrc=0;
if ( iam==0 )
{
AA = (double*) malloc(m*k*sizeof(double));// matricea "globala" (de dim. m*k) pentru
inmultirea AA*BB=CC
BB = (double*) malloc(k*n*sizeof(double));//matricea "globala" (de dim. k*n) pentru
inmultirea AA*BB=CC
CC = (double*) malloc(m*n*sizeof(double));//matricea "globala" (de dim. m*n) pentru
inmultirea AA*BB=CC
// initializarea marticelor globale
for(i=0;i<m;i++ )
     for(j=0;j<k;j++)
            AA[i*k+j]=(10*i+j);//(i+j);
for(i=0;i<k;i++ )
     for(j=0;j<n;j++)
            BB[i*n+j]=AA[i*5+j]+5; //(i+j);
}
else{
AA = NULL;
BB = NULL;
//other processes don't contain parts of A
}
  if (iam==0)
  {
  printf("=========== REZULT OF THE PROGRAM %s \n",argv[0]);
  cout << "Global matrix AA:\n";
              for (i = 0; i < m; ++i) {
                  for (j = 0; j < k; ++j) {
                      cout << setw(3) << *(AA + k*i + j) << " ";
                  }
                  cout << "\n";
              }
              cout << endl;
  cout << "Global matrix BB:\n";
              for (i = 0; i < k; ++i) {
                  for (j = 0; j < n; ++j) {
                      cout << setw(3) << *(BB + n*i + j) << " ";
                  }
                  cout << "\n";
              }
              cout << endl;
  }

Cblacs_barrier(ictxt, "All");
int mA = numroc_( &m, &mb, &myrow, &rsrc, &nprow );
int kA = numroc_( &k, &nb, &mycol, &rsrc, &npcol );
int kB = numroc_( &k, &mb, &myrow, &rsrc, &nprow );
int nB = numroc_( &n, &nb, &mycol, &rsrc, &npcol );
int mC = numroc_( &m, &mb, &myrow, &rsrc, &nprow );
int nC = numroc_( &n, &nb, &mycol, &rsrc, &nprow );
lld_AA = MAX( numroc_( &m, &k, &myrow, &rsrc, &nprow ), 1 );
lld_BB = MAX( numroc_( &k, &n, &myrow, &rsrc, &nprow ), 1 );
lld_CC = MAX( numroc_( &m, &n, &myrow, &rsrc, &nprow ), 1 );
// Initialize discriptors (local matrix A is considered as distributed with blocking
parameters
// m, n, i.e. there is only one block - whole matrix A - which is located on process
(0,0) )
descinit_(descAA, &m, &k, &m, &k, &rsrc, &csrc, &ictxt, &lld_AA, &info );
descinit_(descBB, &k, &n, &k, &n, &rsrc, &csrc, &ictxt, &lld_BB, &info );
descinit_(descCC, &m, &n, &m, &n, &rsrc, &csrc, &ictxt, &lld_CC, &info );
descinit_(descA, &m,  &k,  &mb,  &nb,  &rsrc, &csrc, &ictxt, &mA,  &info);
descinit_(descB, &k,  &n,  &mb,  &nb,  &rsrc, &csrc, &ictxt, &kB,  &info);
descinit_(descC, &m,  &n,  &mb,  &nb,  &rsrc, &csrc, &ictxt, &mC, &info);
A = (double*) malloc(mA*kA*sizeof(double)); //matricea locala
B = (double*) malloc(kB*nB*sizeof(double)); //matricea locala
C = (double*) malloc(mC*nC*sizeof(double)); //matricea locala
// Call pdgeadd_ to distribute matrix (i.e. copy AA into A and  BB into B )
pdgeadd_( "N", &m, &k, &one, AA, &i_one, &i_one, descAA, &zero, A, &i_one, &i_one,
descA);
pdgeadd_( "N", &k, &n, &one, BB, &i_one, &i_one, descBB, &zero, B, &i_one, &i_one,
descB);
// Tipar matricele locale  A si  B
for (int id = 0; id < nprocs; ++id)
{
Cblacs_barrier(ictxt, "All");
if (id == iam) {
printf("Local A(%d*%d) on node %d (%d,%d) \n",kA, mA,iam,myrow,mycol);
//cout << "A on node " << iam << endl;
//for (i = 0; i < mA; ++i)
 for (j = 0; j < kA; ++j)
 {
  //for (j = 0; j < kA; ++j)
  for (i = 0; i < mA; ++i)
  cout << setw(3) << *(A+mA*j+i) << " ";
  cout << endl;
 }
printf("Local B(%d*%d) on node %d (%d,%d) \n", nB,kB,iam,myrow,mycol);
//cout << "B on node " << iam << endl;
//for (i = 0; i < kB; ++i)
for (j = 0; j < nB; ++j)
```

```
 {
//for (j = 0; j < nB; ++j)
for (i = 0; i < kB; ++i)
cout << setw(3) << *(B+kB*j+i) << " ";
cout << endl;
 }
cout << endl;
            }
Cblacs_barrier(ictxt, "All");
}
alpha = 1.0; beta = 0.0;
pdgemm_("No Transpose","No
Transpose",&m,&n,&k,&alpha,A,&ONE,&ONE,descA,B,&ONE,&ONE,descB,&beta,C,&ONE,&ONE,desc
C);
Cblacs_barrier(ictxt, "All");
// Print out the matrix product C
for (int id = 0; id < nprocs; ++id)
{
Cblacs_barrier(ictxt, "All");
if (id == iam) {
printf("Local C(%d*%d) on node %d (%d,%d) \n", mC,nC,iam,myrow,mycol);
//cout << "C_loc on node " << iam << endl;
for (i = 0; i < mC; ++i)
 {
  for (j = 0; j < nC; ++j)
  cout << setw(3) << *(C+mC*j+i) << " ";
  cout << endl;
 }
Cblacs_barrier(ictxt, "All");
}
}
// Se construieste matricea CC

pdgeadd_( "N", &m, &n, &one, C, &i_one, &i_one, descC, &zero, CC, &i_one, &i_one,
descCC);

//*** Tipar rezultate finale

if (iam==0)
  {
   cout << "Global matrix CC =AA*BB:\n";
              //for (i = 0; i < m; ++i)
            for (j = 0; j < n; ++j) {
                  //for (j = 0; j < n; ++j)
                  for (i = 0; i < m; ++i) {
                      cout << setw(5) << *(CC + n*i + j) << " ";
                  }
                  cout << "\n";
              }
              cout << endl;
}

/* Free memory */
    free(A);
    free(B);
    free(C);
if( myrow==0 && mycol==0 ){
free( AA );
free( BB);
free( CC);
}
```

```
    /* Release process grid */
    Cblacs_gridexit(ictxt);
} /* main */
```

The results are the following:

```
[MI_gr_TPS1@hpc]$./mpiCC_ScL -o Example2.9.2.exe Example2.9.2.cpp
[MI_gr_TPS1@hpc]$ /opt/openmpi/bin/mpirun -n 4 -host compute-0-0,compute-0-4
Example2.9.2.exe
Global matrix AA:
  0   1   2   3   4
 10  11  12  13  14
 20  21  22  23  24
 30  31  32  33  34
 40  41  42  43  44

Global matrix BB:
  5   6   7   8   9
 15  16  17  18  19
 25  26  27  28  29
 35  36  37  38  39
 45  46  47  48  49

Local A(3*3) on node 0 (0,0)
  0   1   4
 10  11  14
 40  41  44
Local B(3*3) on node 0 (0,0)
  5   6   9
 15  16  19
 45  46  49

Local A(2*3) on node 1 (0,1)
 20  21  24
 30  31  34
Local B(2*3) on node 1 (0,1)
 25  26  29
 35  36  39

Local A(3*2) on node 2 (1,0)
  2   3
 12  13
 42  43
Local B(3*2) on node 2 (1,0)
  7   8
 17  18
 47  48

Local A(2*2) on node 3 (1,1)
 22  23
 32  33
Local B(2*2) on node 3 (1,1)
 27  28
 37  38

Local C(3*3) on node 0 (0,0)
800 1800 4800
```

```
835 1885 5035
940 2140 5740
Local C(3*2) on node 1 (0,1)
2800 3800
2935 3985
3340 4540
Local C(2*3) on node 2 (1,0)
870 1970 5270
905 2055 5505
Local C(2*2) on node 3 (1,1)
3070 4170
3205 4355
Global matrix CC =AA*BB:
  800  1800  2800  3800  4800
  835  1885  2935  3985  5035
  870  1970  3070  4170  5270
  905  2055  3205  4355  5505
  940  2140  3340  4540  5740
```

The accuracy of the results is confirmed by the sequential program:

```
[MI_gr_TPS1@hpc]$ ./mpiCC_ScL -o dgemm1.exe dgemm1.cpp
[MI_gr_TPS1@hpc]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-0-4
dgemm1.exe
Global matrix AA:
  0   1   2   3   4
 10  11  12  13  14
 20  21  22  23  24
 30  31  32  33  34
 40  41  42  43  44

Global matrix BB:
  5   6   7   8   9
 15  16  17  18  19
 25  26  27  28  29
 35  36  37  38  39
 45  46  47  48  49

Global matrix CC:

800 1800 2800 3800 4800
835 1885 2935 3985 5035
870 1970 3070 4170 5270
905 2055 3205 4355 5505
940 2140 3340 4540 5740
```

```
PvSYMM(SIDE,UPLO,M,N,ALPHA,A,IA,JA,DESCA,B,IB,JB,DESCB,BETA,
       C,IC,JC,DESCC)
void pdsymm_(F_CHAR_T SIDE,F_CHAR_T UPLO,int *M,int *N,
       double *ALPHA,double *A,int *IA,int *JA,int *DESCA,
       double *B,int *IB,int *JB,int *DESCB,double *BETA,
       double *C,int *IC,int *JC,int *DESCC )
```

Purpose: PvSYMM performs one of the distributed matrix-matrix operations

sub(C) := alpha$\times$sub(A) $\times$ sub(B) + beta$\times$sub(C), or

sub(C) := alpha$\times$sub(B) $\times$sub(A) + beta$\times$sub(C),

where sub(C) denotes C(IC:IC+M-1,JC:JC+N-1),

$$\text{sub}(A) \text{ denotes } \begin{cases} A(IA:IA+M-1, JA:JA+M-1) & \text{if } SIDE = 'L', \\ A(IA:IA+N-1, JA:JA+N-1) & \text{if } SIDE = 'R' \end{cases}$$

sub(B) denotes B(IB:IB+M-1,JB:JB+N-1).

Alpha and beta are scalars, sub(A) is a symmetric distributed matrix and sub(B) and sub(C) are M-by-N distributed matrices.

Arguments

SIDE (global input) CHARACTER

On entry, SIDE specifies whether the symmetric distributed matrix sub(A) appears on the left or right in the operation as follows:
*SIDE = 'L' sub(C) := alpha*sub(A)*sub(B) + beta*sub(C),*
*SIDE = 'R' sub(C) := alpha*sub(B)*sub(A) + beta*sub(C),*

UPLO (global input) CHARACTER

On entry, UPLO specifies whether the upper or lower triangular part of the symmetric distributed matrix sub(A) is to be referenced.

M (global input) INTEGER

The number of rows to be operated on i.e., the number of rows of the distributed submatrix sub(C). M≥0.

N (global input) INTEGER

The number of columns to be operated on i.e the number of columns of the distributed submatrix sub(C). N≥0.

ALPHA (global input) REAL/COMPLEX

On entry, ALPHA specifies the scalar alpha.

A (local input) array of dimension (LLD_A, LOC_q(JA+NA-1))

Before entry this array contains the local pieces of the symmetric distributed matrix sub(A), such that when UPLO = 'U', the NA-by-NA upper triangular part of the distributed matrix sub(A) must contain the upper triangular part of the symmetric distributed matrix and the strictly lower triangular part of sub(A) is not referenced, and when UPLO = 'L', the NA-by-NA lower triangular part of the distributed matrix sub(A) must contain the lower triangular part of the symmetric distributed matrix and the strictly lower triangular part of sub(A) is not referenced.

IA (global input) INTEGER

The global row index of the submatrix of the distributed matrix A to operate on.

JA (global input) INTEGER

The global column index of the submatrix of the distributed matrix A to operate on.

DESCA (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix A.

B (local input) array of dimension (LLD_B, LOC_q(JB+N-1))

Before entry, this array contains the local pieces of the distributed matrix sub(B).

IB (global input) INTEGER

The global row index of the submatrix of the distributed matrix B to operate on.

JB (global input) INTEGER

The global column index of the submatrix of the distributed matrix B to operate on.

DESCB (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix B.

BETA (global input) REAL/COMPLEX

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then sub(C) need not be set on input.

C (local input/local output) array of dimension (LLD_C, LOCq(JC+N-1))

Before entry, this array must contain the local pieces of the distributed matrix *sub(C). On exit, the distributed matrix sub(C) is overwritten by the M-by-N updated distributed matrix.*

IC (global input) INTEGER

The global row index of the submatrix of the distributed matrix C to operate on.

JC (global input) INTEGER

The global column index of the submatrix of the distributed matrix C to operate on.

DESCC (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix C.

```
PvTRAN(M,N,ALPHA,A,IA,JA,DESCA,BETA,C,IC,JC,DESCC)
void pdtran_(int *M, int *N, double *ALPHA, double *A, int *IA,
            int *JA, int *DESCA, double *BETA, double *C, int
            *IC, int *JC, int *DESCC)
```

Purpose: PvTRAN transposes a distributed matrix sub(C) = beta x sub(C) + alpha x op(sub(A)) where sub(C) denotes C(IC:IC+M-1,JC:JC+N-1), sub(A) denotes A(IA:IA+N-1,JA:JA+M-1), op(A) denotes AT. Beta is a scalar, sub(C) is an M-by-N distributed matrix, sub(A) is an N-by-M distributed matrix.

Arguments

M (global input) INTEGER

The number of rows to be operated on i.e., the number of rows of the distributed submatrix sub(C). M≥0.

N (global input) INTEGER

The number of columns to be operated on i.e the number of columns of the distributed submatrix sub(C). N≥0.

ALPHA (global input) REAL

On entry, ALPHA specifies the scalar alpha.

A (local input) REAL array of dimension (LLD_A, LOCq(JA+M-1))

This array contains the local pieces of the distributed matrix sub(A).

IA (global input) INTEGER

The global row index of the submatrix of the distributed matrix A to operate on.

JA (global input) INTEGER

The global column index of the submatrix of the distributed matrix A to operate on.

DESCA (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix A.

BETA (global input) REAL

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then sub(C) need not be set on input.

C (local input/local output) array of dimension (LLD_C,LOCq(JC+N-1))

This array contains the local pieces of the distributed matrix sub(C). On exit, the distributed matrix sub(C) is over- written by the updated matrix.

IC (global input) INTEGER

The global row index of the submatrix of the distributed matrix C to operate on.

JC (global input) INTEGER

The global column index of the submatrix of the distributed matrix C to operate on.

DESCC (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix C.

```
PvTRSM(SIDE,UPLO,TRANSA,DIAG,M,N,ALPHA,A,IA,JA,DESCA,
       B,IB,JB,DESCB )
void pdtrsm_(F_CHAR_T SIDE, F_CHAR_T UPLO, F_CHAR_T TRANS,
            F_CHAR_T DIAG, int *M, int *N, double *ALPHA, double
            *A, int *IA, int *JA, int *DESCA, double *B, int
            *IB, int *JB, int *DESCB)
```

Purpose: PvTRSM solves one of the distributed matrix equations
op(sub(A)) $\times$ *X = alpha*$\times$*sub(B),* or *X* $\times$ *op(sub(A))=alpha* $\times$ *sub(B),* where

$$\text{sub}\ (A)\ \text{denotes}\ \begin{cases} A(\text{IA} : \text{IA} + \text{M} - 1, \text{JA} : \text{JA} + \text{M} - 1) & \text{if SIDE} = \text{'L'}, \\ A(\text{IA} : \text{IA} + \text{N} - 1, \text{JA} : \text{JA} + \text{N} - 1) & \text{if SIDE} = \text{'R'} \end{cases}$$

sub(B) denotes *B(IB:IB+M-1,JB:JB+N-1), alpha* is a scalar, *X* and *sub(B)* are an *M*-by-*N* distributed matrix, *sub(A)* is a unit, or non-unit, upper or lower triangular distributed matrix and *op(A)* is one of *op(A) = A* or *op(A) =* A^T. The distributed matrix *X* is overwritten on *sub(B).*

Arguments

`SIDE (global input) CHARACTER`

On entry, SIDE specifies whether op(A) appears on the left or right of X as follows:

SIDE = 'L', op(sub(A))*X = alpha*sub(B),
SIDE = 'R', X*op(sub(A)) = alpha*sub(B).

`UPLO (global input) CHARACTER`

On entry, UPLO specifies whether the distributed matrix sub(A) is an upper or lower triangular distributed matrix.

`TRANSA (global input) CHARACTER`

The form of op(A) to be used in the matrix multiplication as follows:
TRANSA = 'N', op(A) = A,
TRANSA = 'T', op(A) = A^T*,*
TRANSA = 'C', op(A) = A^T*.*

`DIAG (global input) CHARACTER`

On entry, DIAG specifies whether or not sub(A) is unit triangular as follows:
DIAG = 'U', sub(A) is assumed to be unit triangular,
DIAG = 'N', sub(A) is not assumed to be unit triangular.

`M (global input) INTEGER`

The number of rows to be operated on i.e., the number of rows of the distributed submatrix sub(B). $M \geq 0$.

`N (global input) INTEGER`

The number of columns to be operated on i.e., the number of columns of the distributed submatrix sub(B). $N \geq 0$.

`ALPHA (global input) REAL/COMPLEX`

On entry, ALPHA specifies the scalar alpha.

`A (local input) array of dimension (LLD_A, LOCq(JA+NA-1)`

Before entry with UPLO = 'U', the leading NA-by-NA upper triangular part of the distributed matrix sub(A) must contain the local pieces of the upper triangular distributed matrix and its strictly lower triangular part is not referenced. Before entry with UPLO = 'L', the leading NA-by-NA lower triangular part of the distributed matrix sub(A) must contain the lower triangular distributed matrix and its strictly upper triangular part is not referenced. Note that when DIAG = 'U', the diagonal elements of sub(A) are not referenced either, but are assumed to be unity.

`IA (global input) INTEGER`

The global row index of the submatrix of the distributed matrix A to operate on.

`JA (global input) INTEGER`

The global column index of the submatrix of the distributed matrix A to operate on.

`DESCA (global and local input) INTEGER array of dimension 8`

The array descriptor of the distributed matrix A.

B (local input) array of dimension (LLD_B, LOC_q(JB+N-1))

Before entry, this array contains the local pieces of the distributed matrix sub(B). On exit, sub(B) is overwritten by the solution distributed matrix.

IB (global input) INTEGER

The global row index of the submatrix of the distributed matrix B to operate on.

JB (global input) INTEGER

The global column index of the submatrix of the distributed matrix B to operate on.

DESCB (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix B.

```
PvSYMM(SIDE,UPLO,M,N,ALPHA,A,IA,JA,DESCA,B,IB,JB,DESCB,BETA,C,IC
       ,JC,DESCC)
void pdsymm_(F_CHAR_T SIDE, F_CHAR_T UPLO, int *M, int *N,
       double *ALPHA, double *A, int *IA, int *JA, int *DESCA,
       double *B, int *IB, int *JB, int *DESCB, double *BETA,
       double *C, int *IC, int *JC, int *DESCC)
```

Purpose: Performs a scalar-matrix-matrix product (one matrix operand is symmetric) and adds the result to a scalar-matrix product for distribute matrices. The operation is defined as

sub(C):=ALPHA*sub(A)*sub(B)+BETA*sub(C) or

sub(C):=ALPHA*sub(B)*sub(A)+ BETA*sub(C),

where ALPHA and BETA are scalars, sub(A) is a symmetric distributed matrix, sub(A)=A(IA:IA+M-1,JA:JA+M-1), if SIDE ='L', and sub(A)=A(IA:IA+N-1,JA:JA+N-1), if SIDE='R', sub(B) and sub(C) are M-by-N distributed matrices, sub(B)=B(IB:IB+M-1,JB:JB+N-1),sub(C)=C(IC:IC+M-1, JC:JC+N-1).

Input Parameters

SIDE (GLOBAL) CHARACTER*1.

Specifies whether the symmetric distributed matrix SUB(A) appears on the left or right in the operation:

*if side = 'L' or 'L', then SUB(C):=ALPHA*SUB(A)*SUB(B)+BETA*SUB(C);*

*if SIDE = 'R' or 'r', then SUB(C):=ALPHA*SUB(B)*SUB(A)+BETA*SUB(C).*

UPLO (global) CHARACTER*1.

Specifies whether the upper or lower triangular part of the symmetric distributed matrix SUB(A) is used:

if UPLO = 'U' or 'U', then the upper triangular part is used;

if uplo = 'L' or 'l', then the lower triangular part is used.

M (global) INTEGER.

Specifies the number of rows of the distribute submatrix SUB(C), M≥0.

N(global) INTEGER.

Specifies the number of columns of the distribute submatrix SUB(C), M≥0.

ALPHA (global) REAL for PSSYMM, DOUBLE PRECISION for PDSYMM, COMPLEX for PCSYMM,DOUBLE COMPLEX for PZSYMM.

Specifies the scalar ALPHA.

A (local) REAL for PSSYMM, DOUBLE PRECISION for PDSYMM, COMPLEX for PCSYMM, DOUBLE COMPLEX for PZSYMM. Array, DIMENSION (LLD_A, LOC_q(JA+NA-1)).

Before entry this array must contain the local pieces of the symmetric distributed matrix SUB(A), such that when UPLO = 'U' or 'U', the NA-by-NA upper triangular part of the distributed matrix SUB(A) must contain the upper triangular part of the symmetric distributed matrix and the strictly lower triangular part of

SUB(A) is not referenced, and when UPLO = 'L' or 'L', the NA-by-NA lower triangular part of the distributed matrix SUB(A) must contain the lower triangular part of the symmetric distributed matrix and the strictly upper triangular part of SUB(A) is not referenced.

IA, JA (global) INTEGER.

The row and column indices in the distributed matrix A indicating the first row and the first column of the submatrix SUB(A), respectively.

DESCA (global and local) INTEGER array of dimension 8.

The array descriptor of the distributed matrix A.

B (local) REAL for PSSYMM, DOUBLE PRECISION for PDSYMM, COMPLEX for PCSYMM, DOUBLE COMPLEX for PZSYMM.

Array, DIMENSION (LLD_B, LOC_q(JB+N-1)). Before entry this array must contain the local pieces of the distributed matrix SUB(B).

IB, JB (global) INTEGER.

The row and column indices in the distributed matrix B indicating the first row and the first column of the submatrix SUB(B), respectively.

DESCB (global and local) INTEGER array of dimension 8.

The array descriptor of the distributed matrix B.

BETA (global) REAL for PSSYMM, DOUBLE PRECISION for PDSYMM, COMPLEX for PCSYMM, DOUBLE COMPLEX for PZSYMM.

Specifies the scalar BETA. When BETA is set to zero, then SUB(C) need not be set on input.

C (local) REAL for PSSYMM, DOUBLE PRECISION for PDSYMM, COMPLEX for PCSYMM, DOUBLE COMPLEX for PZSYMM.

Array, DIMENSION (LLD_C, LOCQ(JC+N-1)). Before entry this array must contain the local pieces of the distributed matrix SUB(C).

IC, JC (global) INTEGER.

The row and column indices in the distributed matrix C indicating the first row and the first column of the submatrix SUB(C), respectively.

DESCC (global and local) INTEGER array of dimension 8.

The array descriptor of the distributed matrix C

Output Parameters

C

Overwritten by the m-by-n updated matrix.

```
PvLACPY(UPLO,M,N,A,IA,JA,DESCA,B,IB,JB,DESCB)
void pdlacpy_(char *uplo,int *m,int *n,double *a,int *ia,int
           *ja,int *desca,double *b,int *ib,int *jb,int
           *descb);
```

Purpose: The routine copies all or part of a distributed matrix A to another distributed matrix B. No communication is performed, PvLACPY performs a local copy sub(A)=sub(B), where sub(A) denotes A(IA:IA+M-1,JA:JA+N-1) and sub(B) denotes B(IB:IB+M-1,JB:JB+N-1).

Input Parameters

UPLO (global) CHARACTER.

Specifies the part of the distributed matrix sub(A) to be copied:

= 'U': Upper triangular part; the strictly lower triangular part of sub(A) is not referenced;

= 'L': Lower triangular part; the strictly upper triangular part of sub(A) is not referenced.

Otherwise: all of the matrix sub(A) is copied.

M (global) INTEGER.

The number of rows to be operated on, that is, the number of rows of the distributed submatrix `sub(A) (M ≥0)`.

N (global) INTEGER.

The number of columns to be operated on, that is, the number of columns of the distributed submatrix `sub(A) (N≥0)`.

A (local) REAL for PSLACPY, DOUBLE PRECISION for PDLACPY, COMPLEX for PCLACPY, COMPLEX*16 for PZLACPY.

Pointer into the local memory to an array of DIMENSION(LLD_A, $LOC_c(JA+N-1)$*). On entry, this array contains the local pieces of the distributed matrix* `sub(A)`.

IA, JA (global) INTEGER.

The row and column indices in the global array a indicating the first row and the first column of the submatrix `sub(A)`*, respectively.*

DESCA (global and local) INTEGER array, DIMENSION (DLEN_).

The array descriptor for the distributed matrix `A`.

IB, JB (global) INTEGER.

The row and column indices in the global array `B` *indicating the first row and the first column of* `sub(B)` *respectively.*

DESCB (global and local) INTEGER array, DIMENSION (DLEN_).

The array descriptor for the distributed matrix `A`.

Output Parameters

B (local) REAL for PSLACPY, DOUBLE PRECISION for PDLACPY, COMPLEX for PCLACPY, COMPLEX*16 for PZLACPY.

Pointer into the local memory to an array of DIMENSION `(LLD_B, LOCC(JB+N-1))`*. This array contains on exit the local pieces of the distributed matrix* `sub(B)` *set as follows:*

if `UPLO='U', B(IB+I-1,JB+J-1)=A(IA+I-1,JA+J-1),1≤I≤J,1≤J≤N;`

if `UPLO='L', B(IB+I-1,JB+J-1)=A(IA+I-1,JA+J-1),J≤I≤M, 1≤J≤n;`

otherwise, `B(IB+I-1,JB+J-1)=A(IA+I-1,JA+J-1), 1≤I≤M, 1≤J≤N.`

The PBLAS Level 3 routines perform distributed matrix-matrix operations. Table "PBLAS Level 3 Routine Groups and Their Data Types" lists the PBLAS Level 3 routine groups and the data types associated with them.

PBLAS Level 3 Routine Groups and Their Data Types

Routine Group	**Data Types**	**Description**
p?geadd	s, d, c, z	Distributed matrix-matrix sum of general matrices
p?tradd	s, d, c, z	Distributed matrix-matrix sum of triangular matrices
p?gemm	s, d, c, z	Distributed matrix-matrix product of general matrices
p?hemm	c, z	Distributed matrix-matrix product, one matrix is Hermitian
p?herk	c, z	Rank-k update of a distributed Hermitian matrix
p?her2k	c, z	Rank-2k update of a distributed Hermitian matrix
p?symm	s, d, c, z	Matrix-matrix product of distributed symmetric matrices
p?syrk	s, d, c, z	Rank-k update of a distributed symmetric matrix

Routine Group	**Data Types**	**Description**
p?syr2k	s, d, c, z	Rank-2k update of a distributed symmetric matrix
p?tran	s, d	Transposition of a real distributed matrix
p?tranc	c, z	Transposition of a complex distributed matrix (conjugated)
p?tranu	c, z	Transposition of a complex distributed matrix
p?trmm	s, d, c, z	Distributed matrix-matrix product, one matrix is triangular
p?trsm	s, d, c, z	Solution of a distributed matrix equation, one matrix is triangular