Parallel Matrix Multiplication

Parallel Matrix Multiplication

- Computing C=A*B
- Using basic algorithms
  - N^3
- Variables
  - Data layout
  - Topology of machine
  - Scheduling communications

Multiple Algorithms

- Divide the computation and data on the rows of the product matrix C
  - Each process holds n/p rows of A, entire B, and calculate n/p rows of C
  - Communications?
  - Scalable?

SUMMA Algorithm

- SUMMA = Scalable Universal Matrix Multiply
- Simple and easy to generalize
- Presentation from van de Geijn and Watts
  - 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

Standard Algorithm

- standard matrix multiplication is computed using a sequence of inner product computations
- Assuming all Ci,j are initialized to 0, the outer-product is

for i = 0 to n-1 do

for j = 0 to n-1 do

{

$C_{i,j} = 0$

for k = 0 to n-1 do

$C_{i,j} = C_{i,j} + A_{i,k} \times B_{k,j}$

}

Outer-Product Algorithm

- Assuming all Ci,j are initialized to 0, the outer-product is

```
for k = 0 to n-1 do
    for i =  0 to n-1 do
        for j = 0 to n-1 do
            C[i,j] = C[i,j] + A[i,k] × B[k,j]
```

- This outer-product leads to a simple and elegant parallelization on a torus of processors.
- At each step k, all $C_{i,j}$ are updated

Matrix Multiplication on a Grid

- Assume $p = q^2$ processors.
- Assume the matrix is $n \times n$ and that q divides n.
- The matrices are stored on the square $q \times q$ grid processors
- If $m = n/q$, each process holds a $m \times m$ block of each matrix.
- Technically, processor $P_{i,j}$ for $0 \leq i,j < q$ holds matrix blocks $A_{i,j}$, $B_{i,j}$, and $C_{i.j}$.
- This is illustrated on the next slide.

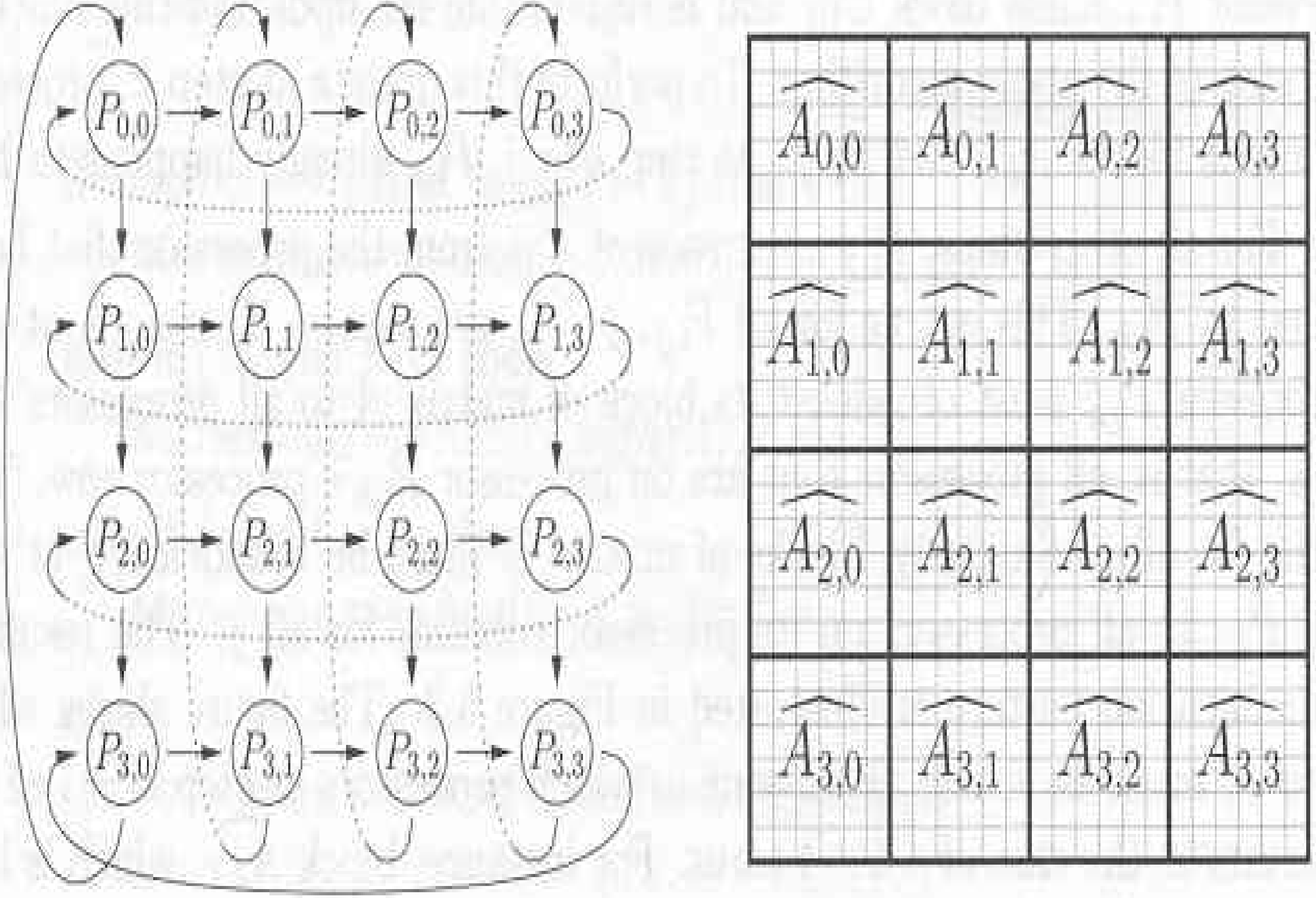
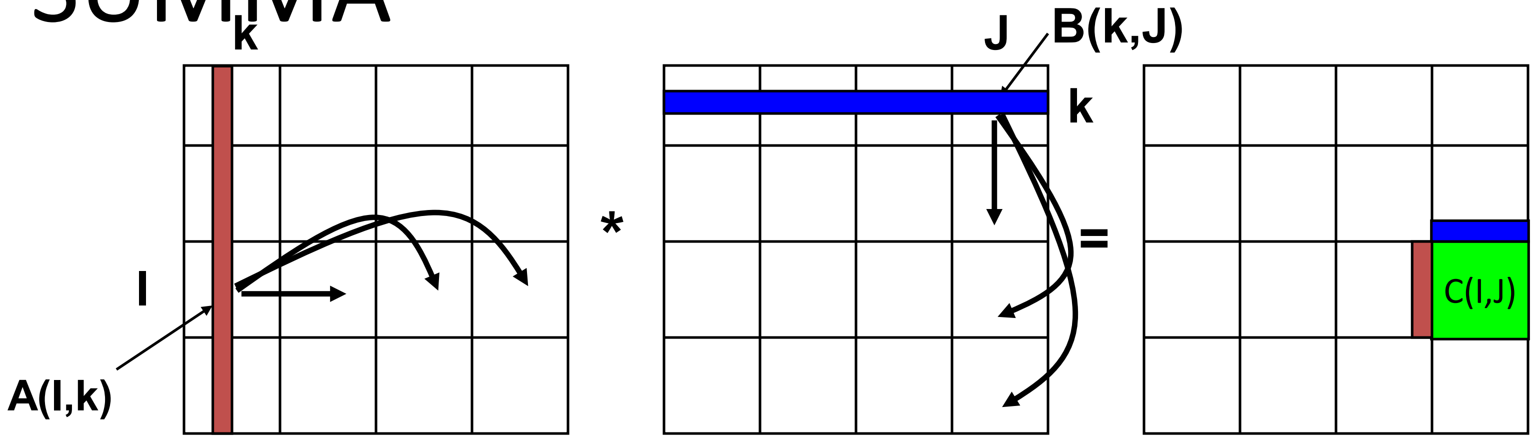


FIGURE 5.2: 2-D block distribution of an $n \times n$ matrix ($n = 24$) on a unidirectional grid/torus of p processors ($p = 4^2 = 16$).

SUMMA



For k=0 to q-1

for all I = 1 to p_r … in parallel

owner of A(I,k) broadcasts it to whole processor row

for all J = 1 to p_c … in parallel

owner of B(k,J) broadcasts it to whole processor column

Receive A(I,k) into Acol

Receive B(k,J) into Brow

C(myrow , mycol) = C(myrow , mycol) + Acol * Brow

Outer-Product Algorithm

- This algorithm can be summarized in terms of matrix blocks and matrix multiplications as

$$
\begin{array}{l}
\textbf{for } k = 0 \textbf{ to } q-1 \textbf{ do} \\
\quad \textbf{for } i = 0 \textbf{ to } q-1 \textbf{ do} \\
\quad\quad \textbf{for } j = 0 \textbf{ to } q-1 \textbf{ do} \\
\quad\quad\quad \widehat{C_{i,j}} \leftarrow \widehat{C_{i,j}} + \widehat{A_{i,k}} \times \widehat{B_{k,j}}
\end{array}
$$

- Next we consider executing this algorithm on a torus of $p = q^2$ processors.
- Processor $P_{i,j}$ holds block $C_{i,j}$ and updates it each step.
- To perform Step k, $P_{i,j}$ needs blocks $A_{i,k}$ & $B_{k,j}$.
- At Step k=j, $P_{i,j}$ already holds block $A_{i,j}$.
- For all other steps, $P_{i,j}$ must obtain $A_{i,k}$ from $P_{i,k}$.

Outer-Product Algorithm

- This is true for all processors $P_{i,j}$ with $j \neq k$.
- Note this means that at step k, processor $P_{i,k}$ must broadcast its block of matrix A to all processors $P_{i,j}$ on its row.
  - This is true for all rows i, as well.
- Similarly, blocks of matrix B must be broadcast at step k by $P_{k,j}$ to all processors on column– and for all j.
- The resulting communication pattern is shown on the next slide.

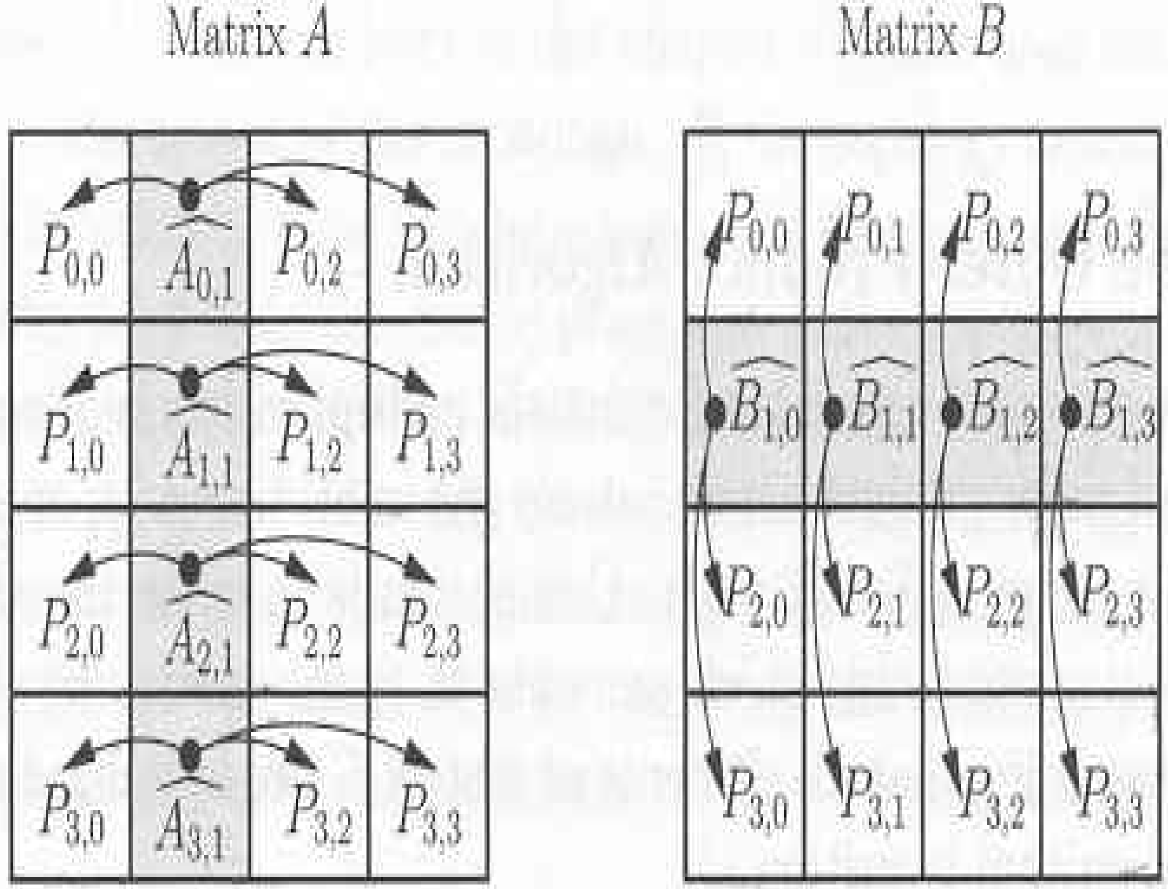


FIGURE 5.3: Communications of blocks of matrices A and B at step $k = 1$ of the outer-product matrix multiplication algorithm on a 4×4 torus of processors.

Algorithm 1 matmul function implementation if row and col communicators are used. Your program should have the exact structure. Pay close attention to how MPI_Bcast is called. You will need to implement a matmulAdd function.

Input: myrank, proc_grid_sz, block_sz, myA, myB, myC
Output: none

double **buffA, **buffB
$buffA = alloc\_2d\_double(block\_sz, block\_sz)$
$buffB = alloc\_2d\_double(block\_sz, block\_sz)$
create grid comm and get coordinates {Follow cartesian example}
create row comm
create col comm
for $k \leftarrow 0$ **to** $proc\_grid\_sz - 1$ **do**
    if $coordinates[1] = k$ **then**
        copy items of myA to buffA
    end if
    MPI_Bcast(*buffA, block_sz*block_sz, MPI_DOUBLE, k, row_comm){*buffA specifies the starting memory location of the matrix buffA.}

```
    if coordinates[0] = k then
        copy items of myB to buffB
    end if
    MPI_Bcast(*buffB, block_sz*block_sz, MPI_DOUBLE, k, col_comm)
    if coordinates[0] = k&&coordinates[1] = k then
        matmulAdd(myC, myA, myB, block_sz)
    else if coordinates[0] = k then
        matmulAdd(myC, buffA, myB, block_sz)
    else if coordinates[1] = k then
        matmulAdd(myC, myA, buffB, block_sz)
    else
        matmulAdd(myC, buffA, buffB, block_sz)
    end if
end for
```

Outer Product Algorithm Steps

- Statement 1 declares the square blocks of the three matrices stored by each processor.
  - The matrix C is assumed to be initialized to zero
  - Arrays A & B contain sub-matrices in PEs in Fig 5.2
- Statements 2&3 declare two helper buffers used by PEs
- The q steps of program occur in lines 7-25
- In statements 8-11, all q processors in column k broadcast (in parallel) their block of A to the processors in each of their rows.
- Statements 12-15 implement similar broadcasts of blocks of matrix B along processor columns.

Outer Product Algorithm Steps (cont)

- Comments:
  - When preceding broadcasts are complete, each PE holds all the needed blocks.
  - Each processor will multiply a block of A by a block of B and adds the result to the block of C, for which it is responsible.
  - The algorithm uses the notation MatrixMultiplyAdd() for PE matrix block operations of $C_{i,j} \leftarrow C_{i,j} + A_{i,k}B_{k,j}$.
- In lines 16-17, if the PE is on both row k & column k, then it can just multiply the two blocks of A and B that it holds.
- Lines 18-19: If the PE is on row k but not on column k, then it will multiply the block of A that it receives with the block of B that it holds.

Outer Product Algorithm Steps (cont)

- Lines 20-21: Similarly, if a PE is on column k but not row k, then it multiplies the block of A it holds with the block of B it just received.
- Lines 22-23 (General Case): If a PE is neither on row k or column k, then it will multiply the block of A it receives with the block of B that it receives.

Generalization of Matrix Multiply:

- By allotting rectangular blocks of Matrix A and B to processors, the preceding algorithm can be adapted to work for non-square matrix products.