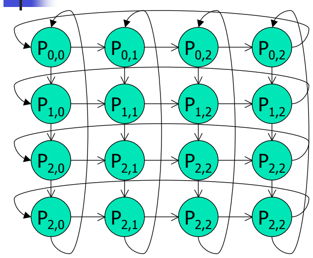
Principles of High Performance Computing (ICS 632)

Algorithms on a Grid of Processors (II)

2-D Matrix Distribution



- We denote by $a_{i,j}$ an element of the matrix
- We denote by $A_{i,j}$ (or A_{ij}) the block of the matrix allocated to $P_{i,j}$

C_{00}	C_{01}	C_{02}	C_{03}
C_{10}	C_{11}	C_{12}	C_{13}
C_{20}	C_{21}	C_{22}	C_{23}
C_{30}	C_{31}	C_{32}	C_{33}

A_{00}	A_{01}	A_{02}	A_{03}
A_{10}	A_{11}	A_{12}	A_{13}
A_{20}	A_{21}	A_{22}	A_{23}
A_{30}	A_{31}	A_{32}	A_{33}

B_{00}	B_{01}	B_{02}	B_{03}
B_{10}	B_{11}	B_{12}	B_{13}
B_{20}	B_{21}	B_{22}	B_{23}
B_{30}	B_{31}	B_{32}	B_{33}

The Cannon Algorithm

- This is a very old algorithm
  - From the time of systolic arrays
  - Adapted to a 2-D grid
- The algorithm starts with a redistribution of matrices A and B
  - Called "preskewing"
- Then the matrices are multiplied
- Then the algorithms are re-redistributed to match the initial distribution
  - Called "postskewing"

Cannon's Preskewing

- Matrix A: each block row of matrix A is shifted so that each processor in the first processor column holds a diagonal block of the matrix

A_{00}	A_{01}	A_{02}	A_{03}
A_{10}	A_{11}	A_{12}	A_{13}
A_{20}	A_{21}	A_{22}	A_{23}
A_{30}	A_{31}	A_{32}	A_{33}

A_{00}	A_{01}	A_{02}	A_{03}
A_{11}	A_{12}	A_{13}	A_{14}
A_{22}	A_{23}	A_{20}	A_{21}
A_{33}	A_{30}	A_{31}	A_{32}

Cannon's Preskewing

- Matrix B: each block column of matrix B is shifted so that each processor in the first processor row holds a diagonal block of the matrix

B_{00}	B_{01}	B_{02}	B_{03}
B_{10}	B_{11}	B_{12}	B_{13}
B_{20}	B_{21}	B_{22}	B_{23}
B_{30}	B_{31}	B_{32}	B_{33}

B_{00}	B_{11}	B_{22}	B_{33}
B_{10}	B_{21}	B_{32}	B_{03}
B_{20}	B_{31}	B_{02}	B_{13}
B_{30}	B_{01}	B_{12}	B_{23}

Cannon's Computation

- The algorithm proceeds in q steps
- At each step each processor performs the multiplication of its block of A and and adds the result to its block of C
- Then blocks of A are shifted to the left and blocks of B are shifted upward
  - Blocks of C never move
- Let's see it on a picture

Cannon's Steps

C_{00}	C_{01}	C_{02}	C_{03}	A_{00}	A_{01}	A_{02}	A_{03}	B_{00}	B_{11}	B_{22}	B_{33}	
C_{10}	C_{11}	C_{12}	C_{13}	A_{11}	A_{12}	A_{13}	A_{10}	B_{10}	B_{21}	B_{32}	B_{03}	local computation on proc (0,0)
C_{20}	C_{21}	C_{22}	C_{23}	A_{22}	A_{23}	A_{20}	A_{21}	B_{20}	B_{31}	B_{02}	B_{13}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{33}	A_{30}	A_{31}	A_{32}	B_{30}	B_{01}	B_{12}	B_{23}	

C_{00}	C_{01}	C_{02}	C_{03}	A_{01}	A_{02}	A_{03}	A_{00}	B_{10}	B_{21}	B_{32}	B_{03}	
C_{10}	C_{11}	C_{12}	C_{13}	A_{12}	A_{13}	A_{10}	A_{11}	B_{20}	B_{31}	B_{02}	B_{13}	Shifts
C_{20}	C_{21}	C_{22}	C_{23}	A_{23}	A_{20}	A_{21}	A_{22}	B_{30}	B_{01}	B_{12}	B_{23}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{30}	A_{31}	A_{32}	A_{33}	B_{00}	B_{11}	B_{22}	B_{33}	

C_{00}	C_{01}	C_{02}	C_{03}	A_{01}	A_{02}	A_{03}	A_{00}	B_{10}	B_{21}	B_{32}	B_{03}	
C_{10}	C_{11}	C_{12}	C_{13}	A_{12}	A_{13}	A_{10}	A_{11}	B_{20}	B_{31}	B_{02}	B_{13}	local computation on proc (0,0)
C_{20}	C_{21}	C_{22}	C_{23}	A_{23}	A_{20}	A_{21}	A_{22}	B_{30}	B_{01}	B_{12}	B_{23}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{30}	A_{31}	A_{32}	A_{33}	B_{00}	B_{11}	B_{22}	B_{33}	

The Algorithm

```
Participate in preskewing of A
Partitipate in preskweing of B
For k = 1 to q
   Local C = C + A*B
   Vertical shift of B
   Horizontal shift of A
Participate in postskewing of A
Partitipate in postskewing of B
```

Performance Analysis

- Let's do a simple performance analysis with a 4-port model
  - The 1-port model is typically more complicated
- Symbols
  - n: size of the matrix
  - qxq: size of the processor grid
  - m = n / q
  - b: communication start-up cost
  - Ta: time to do a basic computation (+= . * .)
  - Tc: time to communicate a matrix element
- T(m,q) = Tpreskew + Tcompute + Tpostskew

Pre/Post-skewing times

- Let's consider the horizontal shift
- Each row must be shifter so that the diagonal block ends up on the first column
- On a mono-directional ring:
  - The last row needs to be shifted (q-1) times
  - All rows can be shifted in parallel
  - Total time needed: (q-1) (b + m^2 Tc)
- On a bi-directional ring, a row can be shifted left or right, depending on which way is shortest!
  - A row is shifted at most floor(q/2) times
  - All rows can be shifter in parallel
  - Total time needed: floor(q/2) (b + m^2 Tc)
- Because of the 4-port assumption, preskewing of A and B can occur in parallel (horizontal and vertical shifts do not interfere)
- Therefore: Tpreskew = Tpostskew = floor(q/2) (b+m^2Tc)

Time for each step

- At each step, each processor computes an mxm matrix multiplication
  - Compute time: m^3 Ta
- At each step, each processor sends/receives a mxm block in its processor row and its processor column
  - Both can occur simultaneously with a 4-port model
  - Takes time b + m^2Tc
- Therefore, the total time for the q steps is: Tcompute = q max (b + m^2Tc, m^3Ta)

Cannon Performance Model

- T(m,n) =2* floor(q/2) (b + m^2Tc) + q max(m^3Ta, b + m^2Tc)
- This performance model is easily adapted
  - If one assumes mono-directional links, then the "floor(q/2)" above becomes "(q-1)"
  - If one assumes 1-port, there is a factor 2 added in front of communication terms
  - If one assumes no overlap of communication and computation at a processor, the "max" above becomes a sum

The Fox Algorithm

- This algorithm was originally developed to run on a hypercube topology
  - But in fact it uses a grid, embedded in the hypercube
- This algorithm requires no pre- or post-skewing
- It relies on horizontal broadcasts of the diagonals of matrix A and on vertical shifts of matrix B
- Sometimes called the “multiply-broadcast-roll” algorithm
- Let’s see it on a picture
  - Although it’s a bit awkward to draw because of the broadcasts

Execution Steps...

C_{00}	C_{01}	C_{02}	C_{03}	A_{00}	A_{01}	A_{02}	A_{03}	B_{00}	B_{01}	B_{02}	B_{03}	initial state
C_{10}	C_{11}	C_{12}	C_{13}	A_{10}	A_{11}	A_{12}	A_{13}	B_{10}	B_{11}	B_{12}	B_{13}	
C_{20}	C_{21}	C_{22}	C_{23}	A_{20}	A_{21}	A_{22}	A_{23}	B_{20}	B_{21}	B_{22}	B_{23}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{30}	A_{31}	A_{32}	A_{33}	B_{30}	B_{31}	B_{32}	B_{33}	

C_{00}	C_{01}	C_{02}	C_{03}	A_{00}	A_{00}	A_{00}	A_{00}	B_{00}	B_{01}	B_{02}	B_{03}	Broadcast of A’s 1st diag. (stored in a Separate buffer)
C_{10}	C_{11}	C_{12}	C_{13}	A_{11}	A_{11}	A_{11}	A_{11}	B_{10}	B_{11}	B_{12}	B_{13}	
C_{20}	C_{21}	C_{22}	C_{23}	A_{22}	A_{22}	A_{22}	A_{22}	B_{20}	B_{21}	B_{22}	B_{23}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{33}	A_{33}	A_{33}	A_{33}	B_{30}	B_{31}	B_{32}	B_{33}	

C_{00}	C_{01}	C_{02}	C_{03}	A_{00}	A_{00}	A_{00}	A_{00}	B_{00}	B_{01}	B_{02}	B_{03}	Local computation
C_{10}	C_{11}	C_{12}	C_{13}	A_{11}	A_{11}	A_{11}	A_{11}	B_{10}	B_{11}	B_{12}	B_{13}	
C_{20}	C_{21}	C_{22}	C_{23}	A_{22}	A_{22}	A_{22}	A_{22}	B_{20}	B_{21}	B_{22}	B_{23}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{33}	A_{33}	A_{33}	A_{33}	B_{30}	B_{31}	B_{32}	B_{33}	

Execution Steps...

C_{00}	C_{01}	C_{02}	C_{03}	A_{00}	A_{01}	A_{02}	A_{03}	B_{10}	B_{11}	B_{12}	B_{13}	Shift of B
C_{10}	C_{11}	C_{12}	C_{13}	A_{10}	A_{11}	A_{12}	A_{13}	B_{20}	B_{21}	B_{22}	B_{23}	
C_{20}	C_{21}	C_{22}	C_{23}	A_{20}	A_{21}	A_{22}	A_{23}	B_{30}	B_{31}	B_{32}	B_{33}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{30}	A_{31}	A_{32}	A_{33}	B_{00}	B_{01}	B_{02}	B_{03}	

C_{00}	C_{01}	C_{02}	C_{03}	A_{01}	A_{01}	A_{01}	A_{01}	B_{10}	B_{11}	B_{12}	B_{13}	Broadcast of A’s 2nd diag. (stored in a Separate buffer)
C_{10}	C_{11}	C_{12}	C_{13}	A_{12}	A_{12}	A_{12}	A_{12}	B_{20}	B_{21}	B_{22}	B_{23}	
C_{20}	C_{21}	C_{22}	C_{23}	A_{23}	A_{23}	A_{23}	A_{23}	B_{30}	B_{31}	B_{32}	B_{33}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{30}	A_{30}	A_{30}	A_{30}	B_{00}	B_{01}	B_{02}	B_{03}	

C_{00}	C_{01}	C_{02}	C_{03}	A_{01}	A_{01}	A_{01}	A_{01}	B_{10}	B_{11}	B_{12}	B_{13}	Local computation
C_{10}	C_{11}	C_{12}	C_{13}	A_{12}	A_{12}	A_{12}	A_{12}	B_{20}	B_{21}	B_{22}	B_{23}	
C_{20}	C_{21}	C_{22}	C_{23}	A_{23}	A_{23}	A_{23}	A_{23}	B_{30}	B_{31}	B_{32}	B_{33}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{30}	A_{30}	A_{30}	A_{30}	B_{00}	B_{01}	B_{02}	B_{03}	

Fox’s Algorithm

```
// No initial data movement
for k = 1 to q  in parallel
  Broadcast A’s kth diagonal
  Local C = C + A*B
  Vertical shift of B
// No final data movement
```

- Again note that there is an additional array to store incoming diagonal block
- This is the array we use in the A*B multiplication

Performance Analysis

- You’ll have to do it in a homework assignment
  - Write pseudo-code of the algorithm in more details
  - Write the performance analysis

Snyder’s Algorithm (1992)

- More complex than Cannon’s or Fox’s
- First transposes matrix B
- Uses reduction operations (sums) on the rows of matrix C
- Shifts matrix B

Execution Steps...

C_{00}	C_{01}	C_{02}	C_{03}	A_{00}	A_{01}	A_{02}	A_{03}	B_{00}	B_{01}	B_{02}	B_{03}	initial state
C_{10}	C_{11}	C_{12}	C_{13}	A_{10}	A_{11}	A_{12}	A_{13}	B_{10}	B_{11}	B_{12}	B_{13}	
C_{20}	C_{21}	C_{22}	C_{23}	A_{20}	A_{21}	A_{22}	A_{23}	B_{20}	B_{21}	B_{22}	B_{23}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{30}	A_{31}	A_{32}	A_{33}	B_{30}	B_{31}	B_{32}	B_{33}	

C_{00}	C_{01}	C_{02}	C_{03}	A_{00}	A_{01}	A_{02}	A_{03}	B_{00}	B_{10}	B_{20}	B_{30}	Transpose B
C_{10}	C_{11}	C_{12}	C_{13}	A_{10}	A_{11}	A_{12}	A_{13}	B_{01}	B_{11}	B_{21}	B_{31}	
C_{20}	C_{21}	C_{22}	C_{23}	A_{20}	A_{21}	A_{22}	A_{23}	B_{02}	B_{12}	B_{22}	B_{32}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{30}	A_{31}	A_{32}	A_{33}	B_{03}	B_{13}	B_{23}	B_{33}	

C_{00}	**C_{01}**	**C_{02}**	**C_{03}**	**A_{00}**	**A_{01}**	**A_{02}**	**A_{03}**	**B_{00}**	**B_{10}**	**B_{20}**	**B_{30}**	Local computation
C_{10}	C_{11}	C_{12}	C_{13}	A_{10}	A_{11}	A_{12}	A_{13}	B_{01}	B_{11}	B_{21}	B_{31}	
C_{20}	C_{21}	C_{22}	C_{23}	A_{20}	A_{21}	A_{22}	A_{23}	B_{02}	B_{12}	B_{22}	B_{32}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{30}	A_{31}	A_{32}	A_{33}	B_{03}	B_{13}	B_{23}	B_{33}	

Execution Steps...

C_{00}	C_{01}	C_{02}	C_{03}	A_{00}	A_{01}	A_{02}	A_{03}	B_{01}	B_{11}	B_{21}	B_{31}	Shift B
C_{10}	C_{11}	C_{12}	C_{13}	A_{10}	A_{11}	A_{12}	A_{13}	B_{02}	B_{12}	B_{22}	B_{32}	
C_{20}	C_{21}	C_{22}	C_{23}	A_{20}	A_{21}	A_{22}	A_{23}	B_{03}	B_{13}	B_{23}	B_{32}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{30}	A_{31}	A_{32}	A_{33}	B_{00}	B_{10}	B_{20}	B_{30}	

C_{00}	C_{01}	C_{02}	C_{03}	A_{00}	A_{01}	A_{02}	A_{03}	B_{01}	B_{11}	B_{21}	B_{31}	Global sum on the rows of C
C_{10}	**C_{11}**	C_{12}	C_{13}	A_{10}	A_{11}	A_{12}	A_{13}	B_{02}	B_{12}	B_{22}	B_{32}	
C_{20}	C_{21}	**C_{22}**	C_{23}	A_{20}	A_{21}	A_{22}	A_{23}	B_{03}	B_{13}	B_{23}	B_{32}	
C_{30}	C_{31}	C_{32}	**C_{33}**	A_{30}	A_{31}	A_{32}	A_{33}	B_{00}	B_{10}	B_{20}	B_{30}	

C_{00}	**C_{01}**	**C_{02}**	**C_{03}**	**A_{00}**	**A_{01}**	**A_{02}**	**A_{03}**	**B_{01}**	**B_{11}**	**B_{21}**	**B_{31}**	Local computation
C_{10}	C_{11}	C_{12}	C_{13}	A_{10}	A_{11}	A_{12}	A_{13}	B_{02}	B_{12}	B_{22}	B_{32}	
C_{20}	C_{21}	C_{22}	C_{23}	A_{20}	A_{21}	A_{22}	A_{23}	B_{03}	B_{13}	B_{23}	B_{32}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{30}	A_{31}	A_{32}	A_{33}	B_{00}	B_{10}	B_{20}	B_{30}	

Execution Steps...

C_{00}	C_{01}	C_{02}	C_{03}	A_{00}	A_{01}	A_{02}	A_{03}	B_{02}	B_{12}	B_{22}	B_{32}	Shift B
C_{10}	C_{11}	C_{12}	C_{13}	A_{10}	A_{11}	A_{12}	A_{13}	B_{03}	B_{13}	B_{23}	B_{33}	
C_{20}	C_{21}	C_{22}	C_{23}	A_{20}	A_{21}	A_{22}	A_{23}	B_{00}	B_{10}	B_{20}	B_{30}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{30}	A_{31}	A_{32}	A_{33}	B_{01}	B_{11}	B_{21}	B_{31}	

C_{00}	**C_{01}**	C_{02}	C_{03}	A_{00}	A_{01}	A_{02}	A_{03}	B_{02}	B_{12}	B_{22}	B_{32}	Global sum on the rows of C
C_{10}	C_{11}	**C_{12}**	C_{13}	A_{10}	A_{11}	A_{12}	A_{13}	B_{03}	B_{13}	B_{23}	B_{33}	
C_{20}	C_{21}	C_{22}	**C_{23}**	A_{20}	A_{21}	A_{22}	A_{23}	B_{00}	B_{10}	B_{20}	B_{30}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{30}	A_{31}	A_{32}	A_{33}	B_{01}	B_{11}	B_{21}	B_{31}	

C_{00}	**C_{01}**	**C_{02}**	**C_{03}**	**A_{00}**	**A_{01}**	**A_{02}**	**A_{03}**	**B_{02}**	**B_{12}**	**B_{22}**	**B_{32}**	Local computation
C_{10}	C_{11}	C_{12}	C_{13}	A_{10}	A_{11}	A_{12}	A_{13}	B_{03}	B_{13}	B_{23}	B_{33}	
C_{20}	C_{21}	C_{22}	C_{23}	A_{20}	A_{21}	A_{22}	A_{23}	B_{00}	B_{10}	B_{20}	B_{30}	
C_{30}	C_{31}	C_{32}	C_{33}	A_{30}	A_{31}	A_{32}	A_{33}	B_{01}	B_{11}	B_{21}	B_{31}	

The Algorithm

var A,B,C: array[0..m-1][0..m-1] of real
var bufferC: array[0..m-1][0..m-1] of real
Transpose B
MatrixMultiplyAdd(bufferC, A, B, m)
Vertical shifts of B
For k = 1 to q-1
    Global sum of bufferC on proc rows into $C_{i,(i+k-1)\%q}$
    MatrixMultiplyAdd(bufferC, A, B, m)
    Vertical shift of B
Global sum of bufferC on proc rows into $C_{i,(i+k-1)\%q}$
Transpose B

Performance Analysis

- The performance analysis isn't fundamentally different than what we've done so far
- But it's a bit cumbersome
- In a homework you'll do the algorithm for transposing a matrix and come up with the performance analysis of this algorithm
  - Just corresponds to the pre- and post-skewing for the Snyder Algorithm

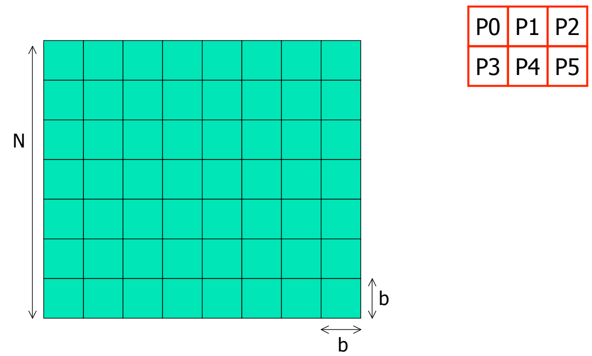
Which Data Distribution?

- So far we've seen:
  - Block Distributions
  - 1-D Distributions
  - 2-D Distributions
  - Cyclic Distributions
- One may wonder what a good choice is for a data distribution?
- Many people argue that a good "Swiss Army knife" is the "2-D block cyclic distribution

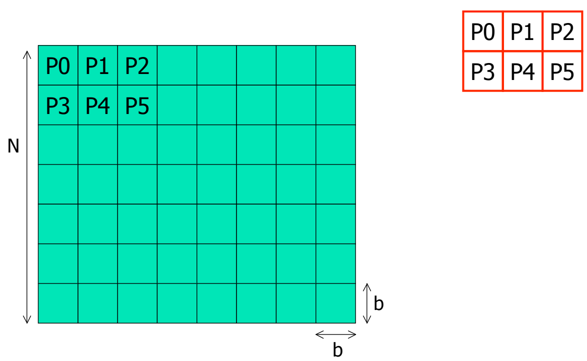
The 2-D block cyclic distribution

- Goal: try to have all the advantages of both the horizontal and the vertical 1-D block cyclic distribution
  - Works whichever way the computation "progresses"
    - left-to-right, top-to-bottom, wavefront, etc.
- Consider a number of processors p = r * c
  - arranged in a rxc matrix
- Consider a 2-D matrix of size NxN
- Consider a block size b (which divides N)

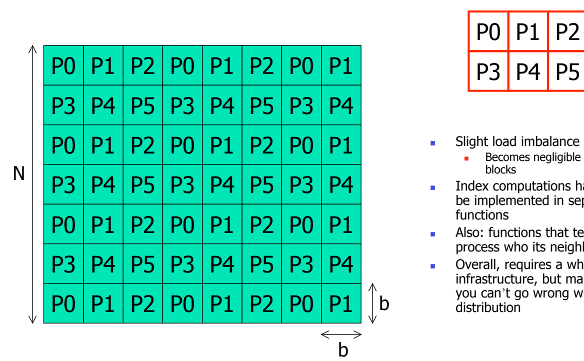
The 2-D block cyclic distribution



The 2-D block cyclic distribution



The 2-D block cyclic distribution



- Slight load imbalance
  - Becomes negligible with many blocks
- Index computations had better be implemented in separate functions
- Also: functions that tell a process who its neighbors are
- Overall, requires a whole infrastructure, but many think you can't go wrong with this distribution

Conclusion

- All the algorithms we have seen in the semester can be implemented on a 2-D block cyclic distribution
- The code ends up much more complicated
- But one may expect several benefits "for free"
- The ScaLAPAK library recommends to use the 2-D block cyclic distribution
  - Although its routines support all other distributions