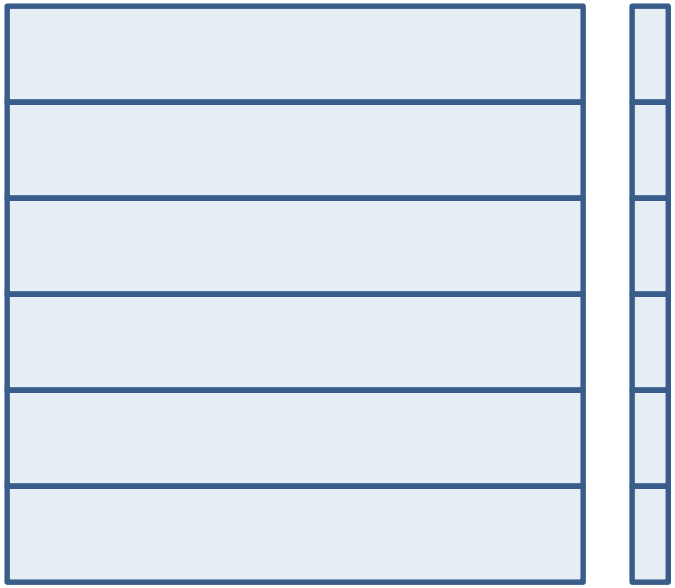
Partitioning and reducing communication for matrix computations

- How to partition the data or work of an algorithm for parallel computing
- One goal is to find a partitioning that reduces communication
- May be the determinant of parallel performance of an algorithm or code
- Communication volume: the sum (in bytes or words) of all communication (across the network) by all processors

Dense matrix-vector multiply

1D partitioning



2D partitioning

For n-by-n matrix and p processors (p=rs in 2D case), what is the communication volume in each case? The data partitioning must be the same before and after the multiply.

Communicaiton Volume for matrix-vector multiply

- 1D partitioning: $V = np$
- 2D partitioning: $V = n(r + s)$
- Thus 2D partitioning has smaller communication volume

Cannon's algorithm for dense matrix multiplication

Initial data layout

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1, 1 2, 5 3, 9 4,13	1, 2 2, 6 3,10 4,14	1, 3 2, 7 3,11 4,15	1, 4 2, 8 3,12 4,16
5, 1 6, 5 7, 9 8,13	5, 2 6, 6 7,10 8,14	5, 3 6, 7 7,11 8,15	5, 4 6, 8 7,12 8,16
9, 1 10, 5 11, 9 12,13	9, 2 10, 6 11,10 12,14	9, 3 10, 7 11,11 12,15	9, 4 10, 8 11,12 12,16
13, 1 14, 5 15, 9 16,13	13, 2 14, 6 15,10 16,14	13, 3 14, 7 15,11 16,15	13, 4 14, 8 15,12 16,16

- Example: AB=C for 16 processors: partition each matrix into 4x4 blocks
- Main idea: processors compute local part of C. What parts of A and B are needed?
- Max storage per node is 1 block of A, 1 block of B, and 1 block of C (not including receive buffers)

Cannon's algorithm for dense matrix multiplication

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1, 1 2, 5 3, 9 4,13	1, 2 2, 6 3,10 4,14	1, 3 2, 7 3,11 4,15	1, 4 2, 8 3,12 4,16
5, 1 6, 5 7, 9 8,13	5, 2 6, 6 7,10 8,14	5, 3 6, 7 7,11 8,15	5, 4 6, 8 7,12 8,16
9, 1 10, 5 11, 9 12,13	9, 2 10, 6 11,10 12,14	9, 3 10, 7 11,11 12,15	9, 4 10, 8 11,12 12,16
13, 1 14, 5 15, 9 16,13	13, 2 14, 6 15,10 16,14	13, 3 14, 7 15,11 16,15	13, 4 14, 8 15,12 16,16

- Example: AB=C for 16 processors: partition each matrix into 4x4 blocks
- For i=1:4
  - At each step, A and B data are first shifted (A by rows; B by cols)
  - Then, processors perform local multiply and accumulate (dgemm)

Initial data layout on 2D mesh of nodes.

Numbers are data blocks. Figure shows where the blocks are initially stored.

This initial layout is *not* suitable for computing terms of the matrix product.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4
6	7	8	5
11	12	9	10
16	13	14	15

1, 1 2, 5 3, 9 4,13	1, 2 2, 6 3,10 4,14	1, 3 2, 7 3,11 4,15	1, 4 2, 8 3,12 4,16
5, 1 6, 5 7, 9 8,13	5, 2 6, 6 7,10 8,14	5, 3 6, 7 7,11 8,15	5, 4 6, 8 7,12 8,16
9, 1 10, 5 11, 9 12,13	9, 2 10, 6 11,10 12,14	9, 3 10, 7 11,11 12,15	9, 4 10, 8 11,12 12,16
13, 1 14, 5 15, 9 16,13	13, 2 14, 6 15,10 16,14	13, 3 14, 7 15,11 16,15	13, 4 14, 8 15,12 16,16

Layout for first multiplication (rows in matrix A have shifted and columns in matrix B have been shifted).

Red products are computed.

Rows and columns are shifted for the subsequent multiplies.

1	6	11	16
5	10	15	4
9	14	3	8
13	2	7	12

1	2	3	4
6	7	8	5
11	12	9	10
16	13	14	15

1, 1 2, 5 3, 9 4,13	1, 2 2, 6 3,10 4,14	1, 3 2, 7 3,11 4,15	1, 4 2, 8 3,12 4,16
5, 1 6, 5 7, 9 8,13	5, 2 6, 6 7,10 8,14	5, 3 6, 7 7,11 8,15	5, 4 6, 8 7,12 8,16
9, 1 10, 5 11, 9 12,13	9, 2 10, 6 11,10 12,14	9, 3 10, 7 11,11 12,15	9, 4 10, 8 11,12 12,16
13, 1 14, 5 15, 9 16,13	13, 2 14, 6 15,10 16,14	13, 3 14, 7 15,11 16,15	13, 4 14, 8 15,12 16,16

Layout for second multiplication (after shifting rows 1 left and columns 1 up).

Green products are computed.

Repeat for all block rows and columns.

5	10	15	4
9	14	3	8
13	2	7	12
1	6	11	16

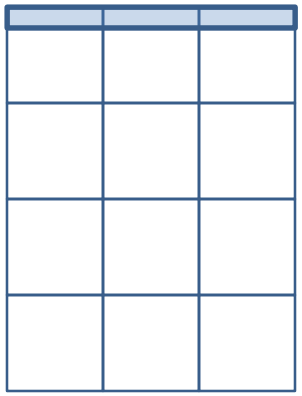
2	3	4	1
7	8	5	6
12	9	10	11
13	14	15	16

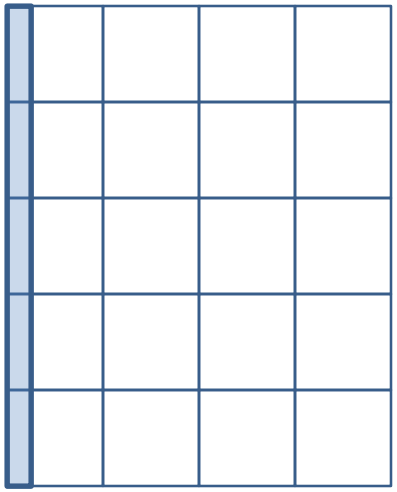
1, 1 2, 5 3, 9 4,13	1, 2 2, 6 3,10 4,14	1, 3 2, 7 3,11 4,15	1, 4 2, 8 3,12 4,16
5, 1 6, 5 7, 9 8,13	5, 2 6, 6 7,10 8,14	5, 3 6, 7 7,11 8,15	5, 4 6, 8 7,12 8,16
9, 1 10, 5 11, 9 12,13	9, 2 10, 6 11,10 12,14	9, 3 10, 7 11,11 12,15	9, 4 10, 8 11,12 12,16
13, 1 14, 5 15, 9 16,13	13, 2 14, 6 15,10 16,14	13, 3 14, 7 15,11 16,15	13, 4 14, 8 15,12 16,16

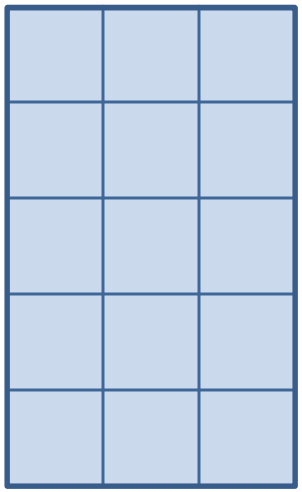
Drawbacks of Cannon's Algorithm

- How to use if number of processors is not square?
- For m-by-n matrix and p-by-p mesh, how to use if m or n is not divisible by p, i.e., blocks of A (or B) are not the same size?

SUMMA algorithm for dense matrix multiplication

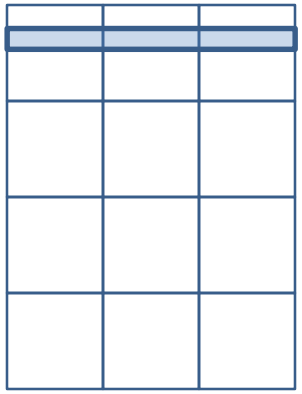


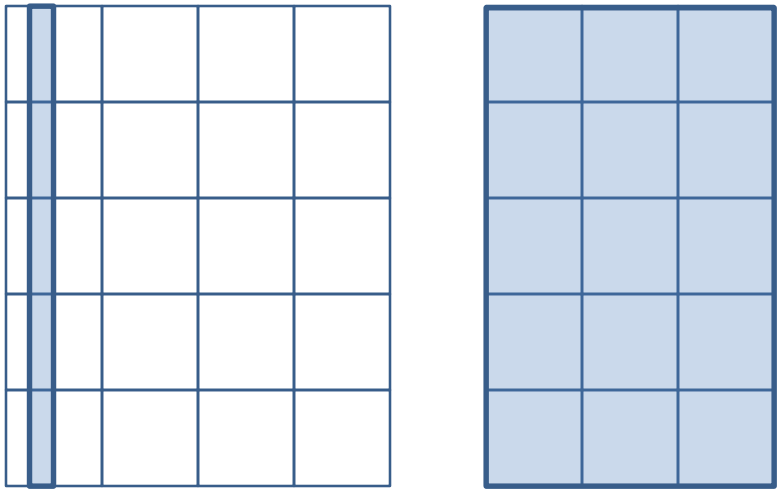




- Algorithm used in PDGEMM function in Parallel BLAS (PBLAS)
- $C = \sum_k A(:,k)B(k,:)$

SUMMA algorithm example for 5x3 processor grid





$$C = \sum_k A(:,k)B(k,:)$$

For k = 1:nk

- Owners of col k of A send to procs in proc row
- Owners of row k of B send to procs in row col
- Compute outer product and accumulate to Cij

Endfor

3D matrix multiplication algorithm

- Partition the work rather than the data
- Data is replicated in order to reduce communication
- For 8 processes, the matrix is replicated by a factor of 2 over the entire machine
- For 27 processes, the replication factor is 3
- In general, for q x q x q processes, the replication factor is q
- Algorithm works also for non-cubic numbers of processes
- Communication time is asymptotically better than that for SUMMA algorithm