Parallel Matrix Operations using MPI

CS 5334/4390 Spring 2015

Shirley Moore, Instructor

April 15, 2015

Matix Algorithms: Introduction

- Due to their regular structure, parallel computations involving matrices and vectors readily lend themselves to data-decomposition.
- Most algorithms use one- and two-dimensional block, cyclic, and block-cyclic partitionings.

Matrix-Vector Multiplication

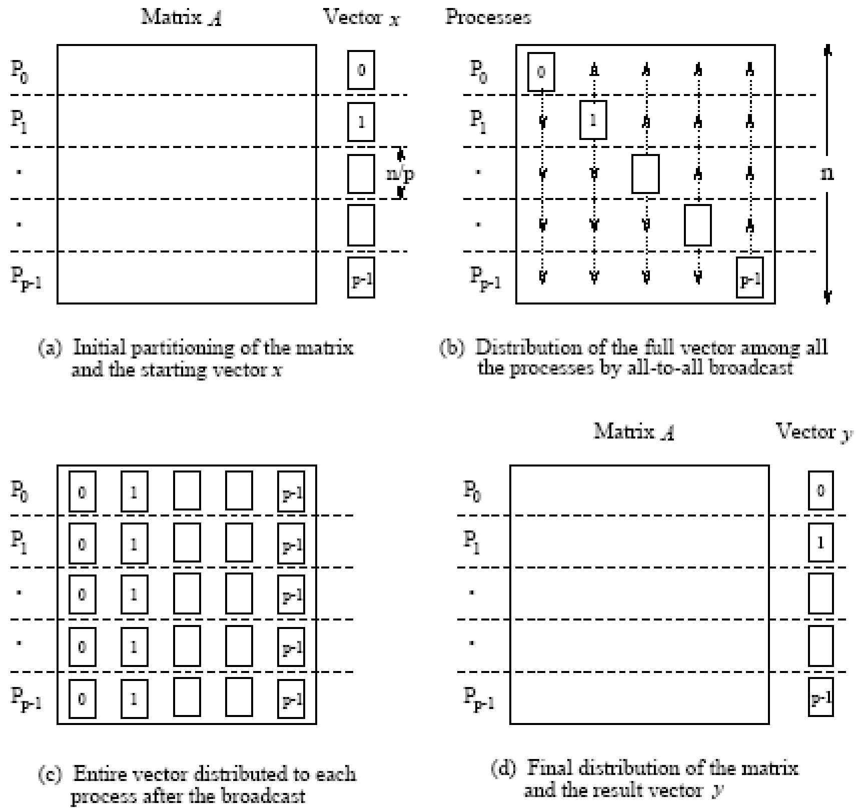
- We aim to multiply a dense n x n matrix A with an n x 1 vector x to yield the n x 1 result vector y.
- The serial algorithm requires n^2 multiplications and additions.

$$W = n^2.$$

Matrix-Vector Multiplication: Rowwise 1-D Partitioning

- The n x n matrix is partitioned among p processors, with each processor storing n/p complete rows of the matrix.
- The n x 1 vector x is distributed such that each process owns n/p of its elements.

Matrix-Vector Multiplication: Rowwise 1-D Partitioning



(a) Initial partitioning of the matrix and the starting vector x

(b) Distribution of the full vector among all the processes by all-to-all broadcast

(c) Entire vector distributed to each process after the broadcast

(d) Final distribution of the matrix and the result vector y

Multiplication of an n x n matrix with an n x 1 vector using rowwise block 1-D partitioning. For the one-row-per-process case, $p = n$.

Matrix-Vector Multiplication: Rowwise 1-D Partitioning

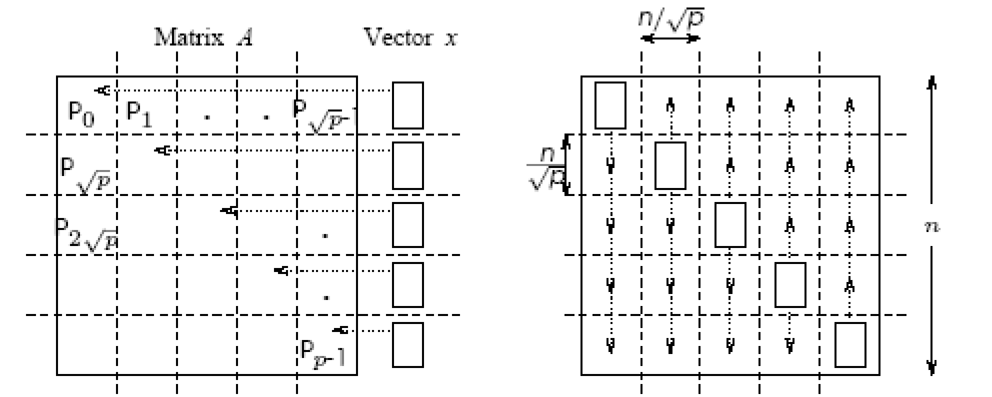
- Consider the case when $p < n$ and we use block 1D partitioning.
- Each process initially stores n/p complete rows of the matrix and a portion of the vector of size n/p.
- The all-to-all broadcast takes place among p processes and involves messages of size n/p.
- This is followed by n/p local dot products.
- Thus, the parallel run time of this procedure on a hypercube or fat tree is

$$T_p = \frac{n^2}{p} + t_s \log p + t_w \frac{n}{p}(p-1) \approx \frac{n^2}{p} + t_s \log p + t_w n$$

Matrix-Vector Multiplication: 2-D Partitioning

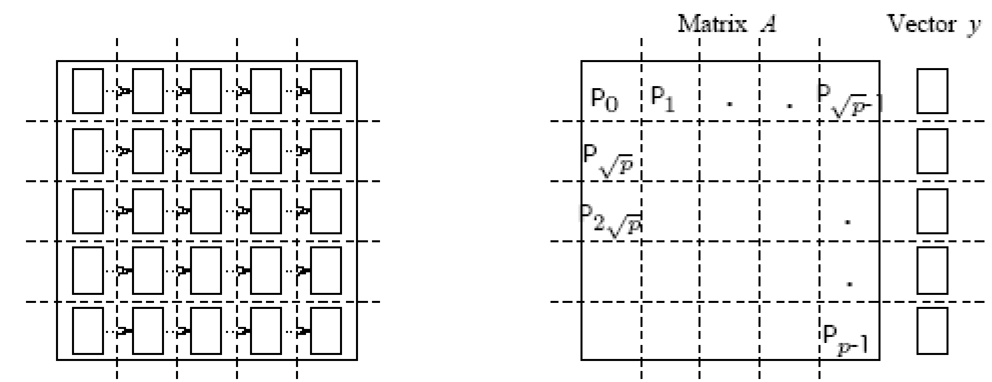
- The n x n matrix is partitioned among p processors such that each processor owns n^2/p elements.
- The n x 1 vector x is distributed only in the last column of processors.

Matrix-Vector Multiplication: 2-D Partitioning



(a) Initial data distribution and communication steps to align the vector along the diagonal

(b) One-to-all broadcast of portions of the vector along process columns



(c) All-to-one reduction of partial results

(d) Final distribution of the result vector

Matrix-vector multiplication with block 2-D partitioning. For the one-element-per-process case, $p = n^2$ if the matrix size is n x n .

Matrix-Vector Multiplication: 2-D Partitioning

- We must first align the vector with the matrix appropriately.
- The first communication step for the 2-D partitioning aligns the vector x along the principal diagonal of the matrix.
- The second step copies the vector elements from each diagonal process to all the processes in the corresponding column using n simultaneous broadcasts among all processors in the column.
- Finally, the result vector is computed by performing an all-to-one reduction along the columns.

Matrix-Vector Multiplication: 2-D Partitioning

- When using fewer than n^2 processors, each process owns an $(n/\sqrt{p}) \times (n/\sqrt{p})$ block of the matrix.
- The vector is distributed in portions of $n/\sqrt{p}$ elements in the last process-column only.
- In this case, the message sizes for the alignment, broadcast, and reduction are all $n/\sqrt{p}$.
- The computation is a product of an $(n/\sqrt{p}) \times (n/\sqrt{p})$ submatrix with a vector of length $n/\sqrt{p}$.

Matrix-Vector Multiplication: 2-D Partitioning

- The first alignment step takes time

$$t_s + t_w n/\sqrt{p}$$

- The broadcast and reductions take time

$$(t_s + t_w n/\sqrt{p}) \log(\sqrt{p})$$

- Local matrix-vector products take time

$$t_c n^2/p$$

- Total time is

$$T_P \approx \frac{n^2}{p} + t_s \log p + t_w \frac{n}{\sqrt{p}} \log p$$

Matrix-Matrix Multiplication

- Consider the problem of multiplying two n x n dense, square matrices A and B to yield the product matrix $C = A$ x B.
- The serial complexity is $O(n^3)$.
- We do not consider better serial algorithms (Strassen's method), although, these can be used as serial kernels in the parallel algorithms.
- A useful concept in this case is called *block* operations. In this view, an n x n matrix A can be regarded as a q x q array of blocks $A_{i,j}$ ($0 \leq i, j < q$) such that each block is an (n/q) x (n/q) submatrix.
- In this view, we perform q^3 matrix multiplications, each involving (n/q) x (n/q) matrices.

Matrix-Matrix Multiplication

- Consider two n x n matrices A and B partitioned into p blocks $A_{i,j}$ and $B_{i,j}$ ($0 \leq i, j < \sqrt{p}$) of size $(n/\sqrt{p}) \times (n/\sqrt{p})$ each.
- Process $P_{i,j}$ initially stores $A_{i,j}$ and $B_{i,j}$ and computes block $C_{i,j}$ of the result matrix.
- Computing submatrix $C_{i,j}$ requires all submatrices $A_{i,k}$ and $B_{k,j}$ for $0 \leq k < \sqrt{p}$.
- All-to-all broadcast blocks of A along rows and B along columns.
- Perform local submatrix multiplication.

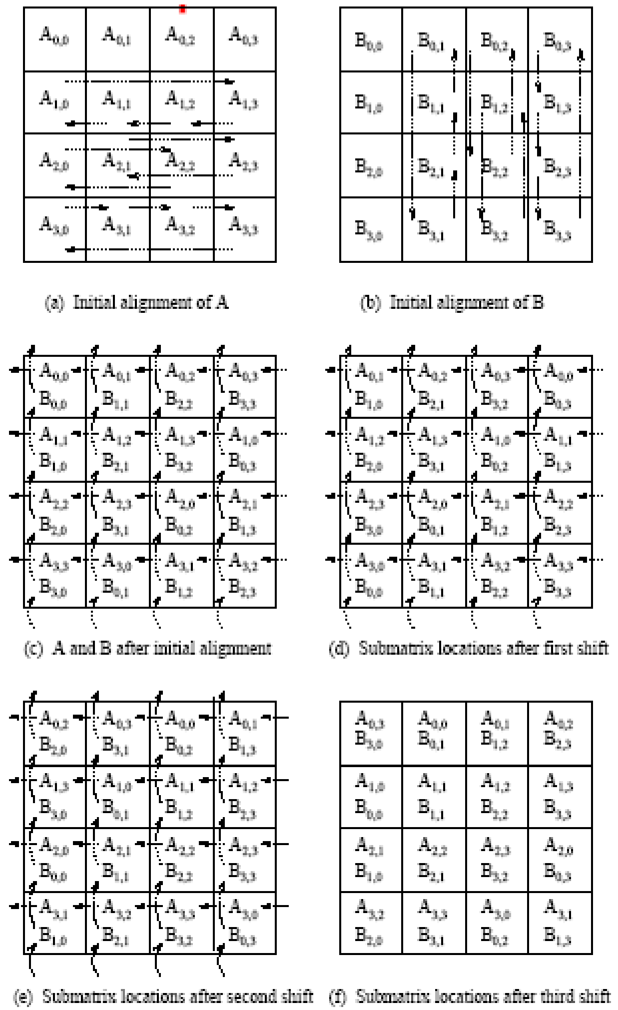
Matrix-Matrix Multiplication

- The two broadcasts take time
$$2(t_s \log(\sqrt{p}) + t_w(n^2/p)(\sqrt{p} - 1))$$
- The computation requires $\sqrt{p}$ multiplications of $(n/\sqrt{p}) \times (n/\sqrt{p})$ sized submatrices.
- The parallel run time is approximately
$$T_P = \frac{n^3}{p} + t_s \log p + 2t_w \frac{n^2}{\sqrt{p}}.$$
- Major drawback of the algorithm is that it is not memory optimal.

Matrix-Matrix Multiplication: Cannon's Algorithm

- In this algorithm, we schedule the computations of the $\sqrt{p}$ processes of the ith row such that, at any given time, each process is using a different block $A_{i,k}$.
- These blocks can be systematically rotated among the processes after every submatrix multiplication so that every process gets a fresh $A_{i,k}$ after each rotation.

Matrix-Matrix Multiplication: Cannon's Algorithm



Communication steps in Cannon's algorithm on 16 processes.

Matrix-Matrix Multiplication: Cannon's Algorithm

- Align the blocks of A and B in such a way that each process multiplies its local submatrices. This is done by shifting all submatrices $A_{i,j}$ to the left (with wraparound) by i steps and all submatrices $B_{i,j}$ up (with wraparound) by j steps.
- Perform local block multiplication.
- Each block of A moves one step left and each block of B moves one step up (again with wraparound).
- Perform next block multiplication, add to partial result, repeat until all $\sqrt{p}$ blocks have been multiplied.