Parallel Applications

Sathish Vadhiyar

Gaussian Elimination - Review

Version 1

for each column i

  zero it out below the diagonal by adding multiples of row i to later rows

for i= 1 to n-1

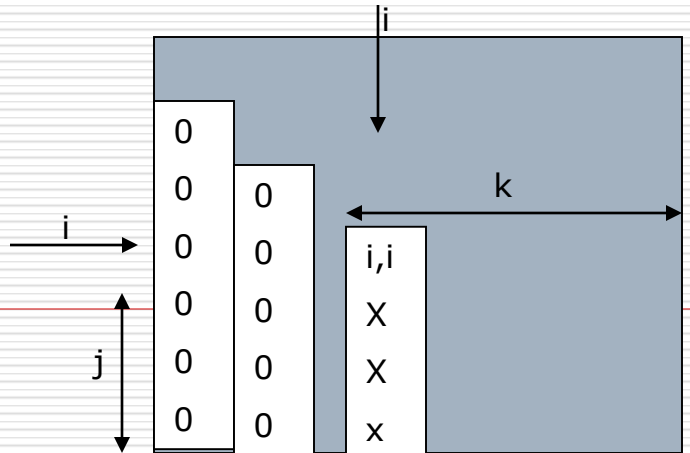
  for each row j below row i

  for j = i+1 to n

    add a multiple of row i to row j

    for k = i to n

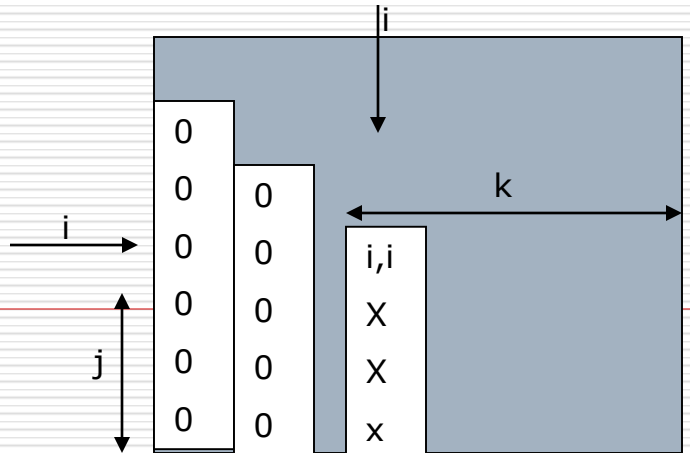
      A(j, k) = A(j, k) – A(j, i)/A(i, i) * A(i, k)



Gaussian Elimination - Review

Version 2 – Remove A(j, i)/A(i, i) from inner loop

```
for each column i
  zero it out below the diagonal by adding multiples of row i to
      later rows
for i= 1 to n-1
  for each row j below row i
  for j = i+1 to n
    m = A(j, i) / A(i, i)
    for k = i to n
      A(j, k) = A(j, k) – m* A(i, k)
```



Gaussian Elimination - Review

Version 3 – Don't compute what we already know

for each column i

  zero it out below the diagonal by adding multiples of row i to later rows

for i= 1 to n-1

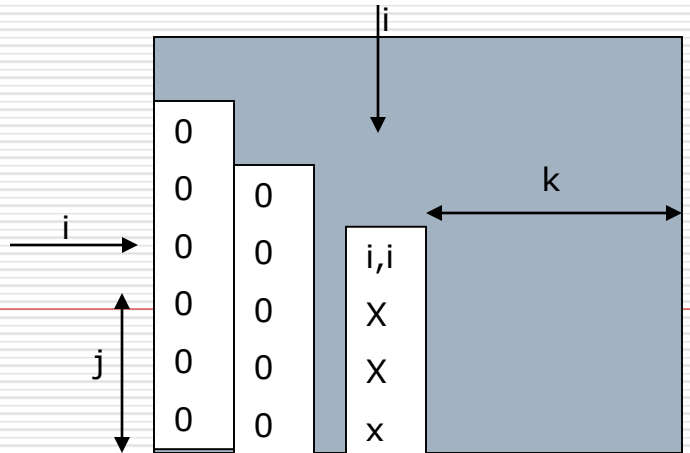
  for each row j below row i

  for j = i+1 to n

    m = A(j, i) / A(i, i)

    for k = i+1 to n

      A(j, k) = A(j, k) – m* A(i, k)



Gaussian Elimination - Review

Version 4 – Store multipliers m below diagonals

for each column i

  zero it out below the diagonal by adding multiples of row i to later rows

for i= 1 to n-1

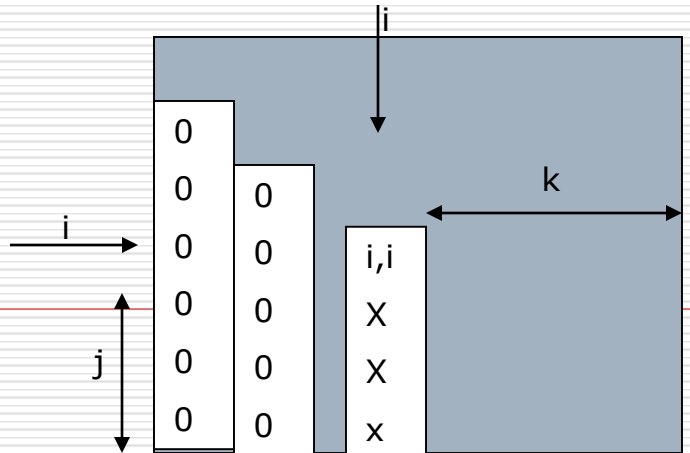
  for each row j below row i

  for j = i+1 to n

    A(j, i) = A(j, i) / A(i, i)

    for k = i+1 to n

      A(j, k) = A(j, k) – A(j, i)* A(i, k)



GE - Runtime

- Divisions

$$1 + 2 + 3 + \ldots (n-1) = n^2/2 \text{ (approx.)}$$

- Multiplications / subtractions

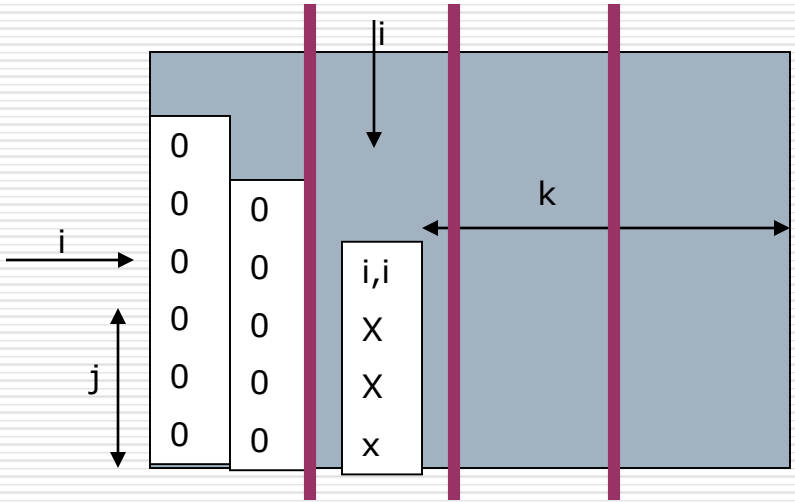
$$1^2 + 2^2 + 3^2 + 4^2 + 5^2 + \ldots (n-1)^2 = n^3/3 - n^2/2$$

- Total

$$2n^3/3$$

Parallel GE

- 1st step – 1-D block partitioning along blocks of n columns by p processors



1D block partitioning - Steps

1. Divisions

$n^2/2$

2. Broadcast

$x\log(p) + y\log(p-1) + z\log(p-3) + \ldots \log 1 < n^2\log p$

3. Multiplications and Subtractions

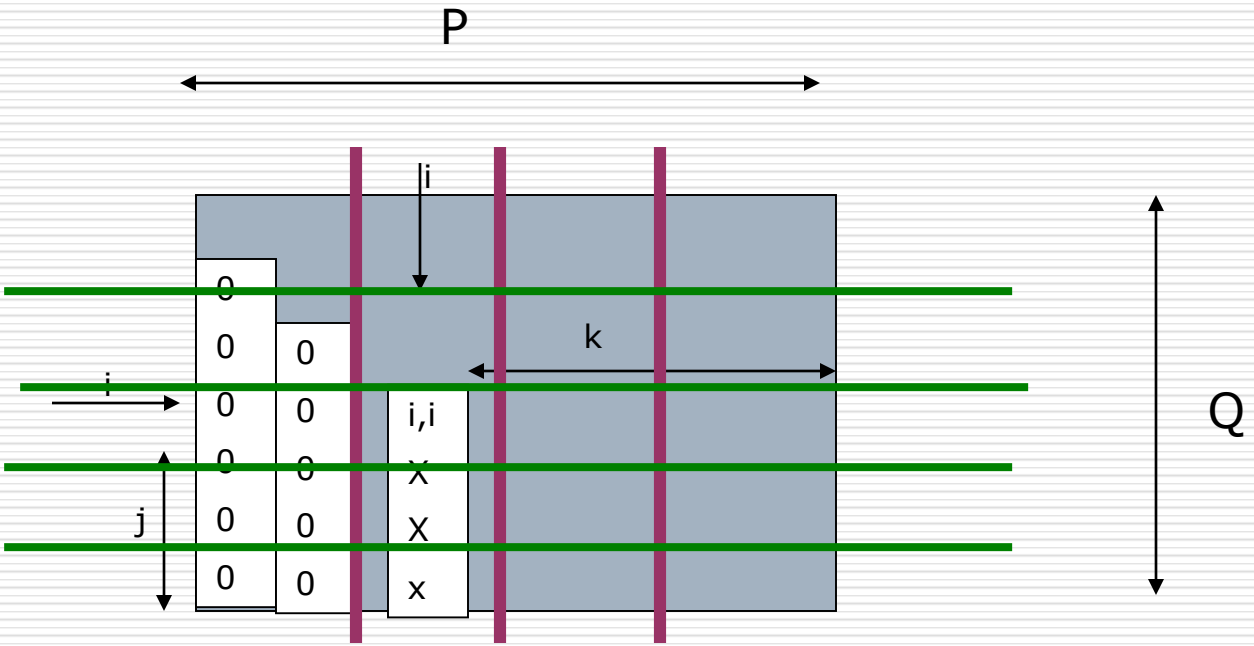
$(n-1)n/p + (n-2)n/p + \ldots. 1x1 = n^3/p$ (approx.)

Runtime:

$< n^2/2 + n^2\log p + n^3/p$

2-D block

- To speedup the divisions



2D block partitioning - Steps

1. Broadcast of (k,k)

   logQ

2. Divisions

   n^2/Q (approx.)

3. Broadcast of multipliers

   $x\log(P) + y\log(P-1) + z\log(P-2) + \ldots = n^2/Q \log P$

4. Multiplications and subtractions

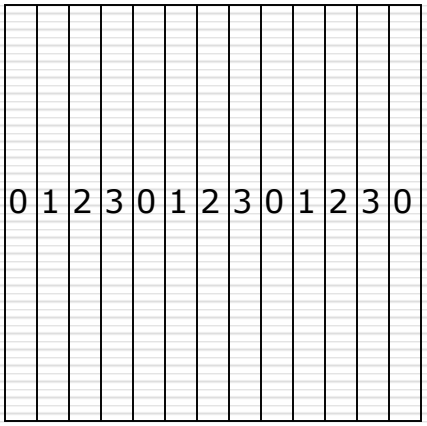
   n^3/PQ (approx.)

Problem with block partitioning for GE

- Once a block is finished, the corresponding processor remains idle for the rest of the execution
- Solution? -

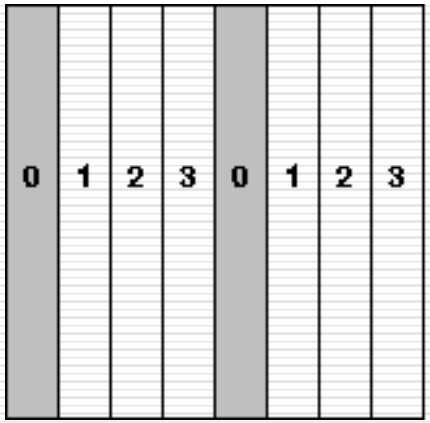
Onto cyclic

- The block partitioning algorithms waste processor cycles. No load balancing throughout the algorithm.
- Onto cyclic



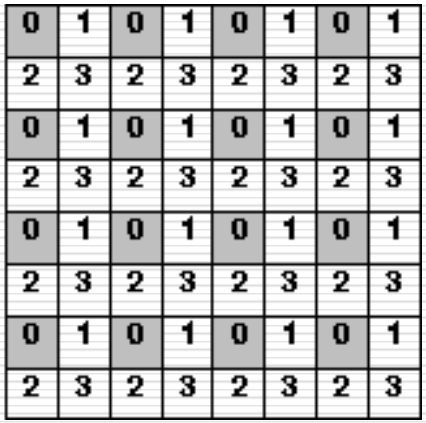
cyclic

Load balance



1-D block-cyclic

Load balance, block operations, but column factorization bottleneck



2-D block-cyclic

Has everything

Block cyclic

- Having blocks in a processor can lead to block-based operations (block matrix multiply etc.)
- Block based operations lead to high performance

AN ADVANCED SCIENTIFIC APPLICATION: MOLECULAR DYNAMICS

GE: Miscellaneous
GE with Partial Pivoting

- 1D block-column partitioning: which is better? Column or row pivoting

•Column pivoting does not involve any extra steps since pivot search and exchange are done locally on each processor. O(n-i-1)

•The exchange information is passed to the other processes by piggybacking with the multiplier information

- Row pivoting
- Involves distributed search and exchange – O(n/P)+O(logP)

- 2D block partitioning: Can restrict the pivot search to limited number of columns

Molecular Dynamics

- Application in many areas including biological systems (e.g. drug discovery), metallurgy (e.g. interaction of metal with liquids) etc.
- A domain consisting of number of particles (molecules)
- Each molecule, i is exerted a force, f_{ij} by another molecule, j
- Forces are of two kinds:
  - Non-bonded forces – computations of pairwise interactions.
  - Bonded forces – computations of interactions between molecules that are connected by bonds. Connectivities are fixed. Hence these forces depend on topology of the structure

Molecular Dynamics

- The sum of all the forces, $F_i = \sum_j f_{ij}$ makes the particles assume a new position and velocity
- Particles that are r distance apart do not influence each other
- Thus non-bonded forces are only computed between atoms that are within this cutoff distance
- Given initial velocities and positions of particles, their movements are followed for discrete time steps

MD Parallelization

- 3 methods
- 1. Atom decomposition
- 2. Space decomposition
- 3. Force decomposition

Atom Decomposition

- Each processor is assigned N/P atoms and updates their positions and velocities irrespective of where they move in the physical domain
- The computational work involved can be represented by the NxN matrix, F, where Fi,j is the non-bonded force on atom i due to atom j
- x and f are vectors that represent positions of and total force on each atom

Atom Decomposition

- For parallelization, F, x and f are distributed with 1-D block distribution across processors. i.e., every processor computes consecutive N/P rows
- Each processor will need the positions of many atoms owned by other processors; hence each processor stores a copy of all N atom positions, x
- Hence this algorithm is also called **replicated data** algorithm

RD Algorithm

- For each time step
- each processor computes forces on its atoms
- updates positions
- processors communicate their positions to all the other processors

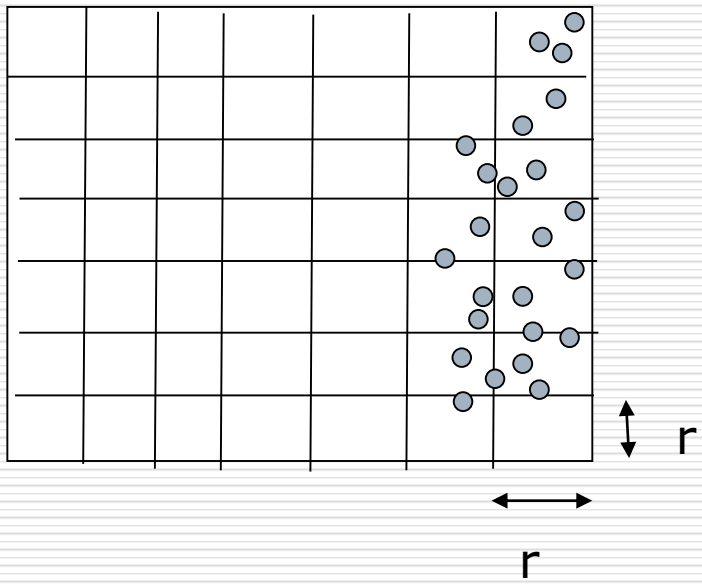
- Different atoms have different neighbor entitites; hence the F matrix has to be load balanced
- The main disadvantage is the all-to-all communication of x; also causes memory overhead since x is replicated

Method 2 – Space decomposition

- Using 2D decomposition
- In a typical Molecular Dynamics simulation problem, the amount of data that are communicated between processors are not known in advance
- The communication is slightly irregular

Space Decomposition - Solution

- The cutoff distance, r is used to reduce the time for summation from $O(n^2)$



Domain decomposed into cells of size rxr

Particles in one cell interact with particles in the neighbouring 8 cells and particles in the same cell

Space Decomposition - Solution

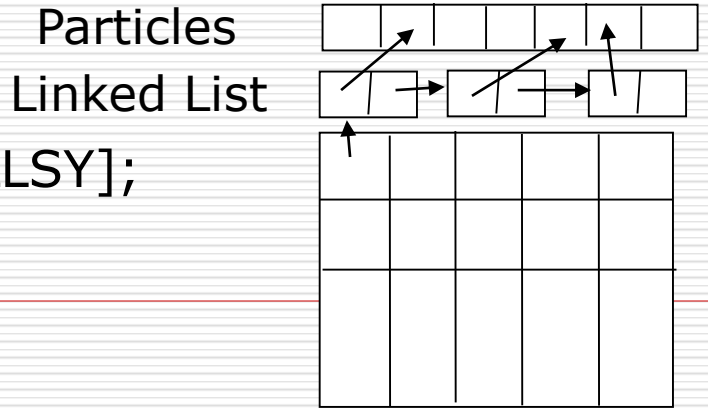
Data structures:

An array of all particles. Each element holds <position, velocity>

A 2D array of linked lists, one for each cell. Each element of a linked list contains pointers to particles.

```
struct particle{
  double position[2];
  double velocity[2];
} Particles[MAX_PARTICLES];

struct list{
  particle* part;
  struct list* next;
}*List[MAX_CELLSX][MAX_CELLSY];
```



Space Decomposition – Sequential Logic

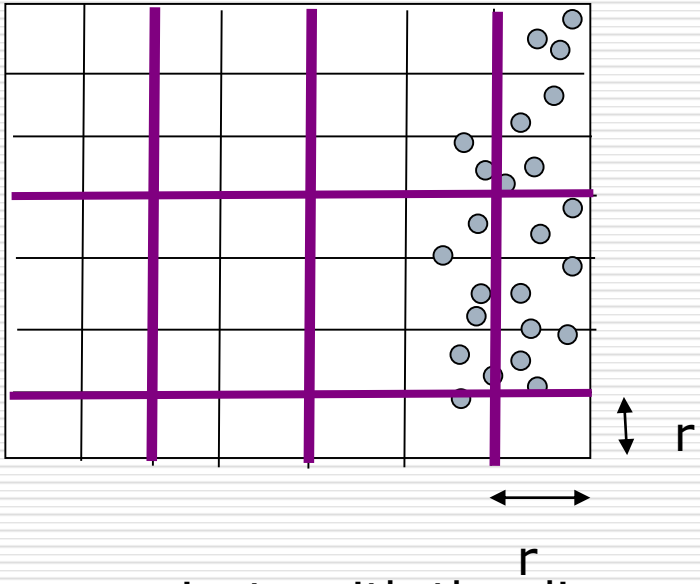
```
Initialize Particles and Lists;

for each time step
  for each particle i
    Let cell(k, l) hold i
    F[i] = 0;
     for each particle j in this cell and neighboring 8 cells, and
      are r distance from i{
        F[i]+= f[i, j];
    }
     update particle[i].{position, velocity} due to F[i];
     if new position in new cell (m,n) update Lists[k,l] and
      Lists[m,n]
```

MD – Space Decomposition

A 2D array of processors similar to Laplace

Each processor holds a set of cells



Differences:

- A processor can communicate with the diagonal neighbors
- Amount of data communicated varies over time steps
- Receiver does not know the amount of data

MDS – parallel solution

- Steps

1. Communication – Each processor communicates parameters of the particles on the boundary cells to its 8 neighboring cells

   Challenges – to communicate diagonal cells

2. Update – Each processor calculates new particle velocities and positions

3. Migration – Particles may migrate to cells in other processors

Other challenges:

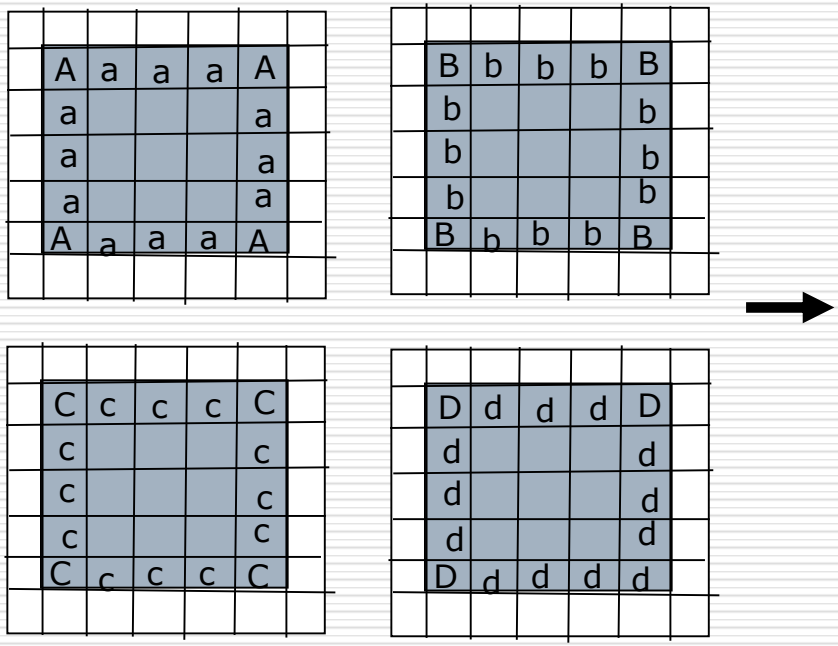
1. Appropriate packing of data.
2. Particles may have to go through several hops during migration

Assumptions:

1. For simplicity, let us assume that particles are transported to only neighboring cells during migration

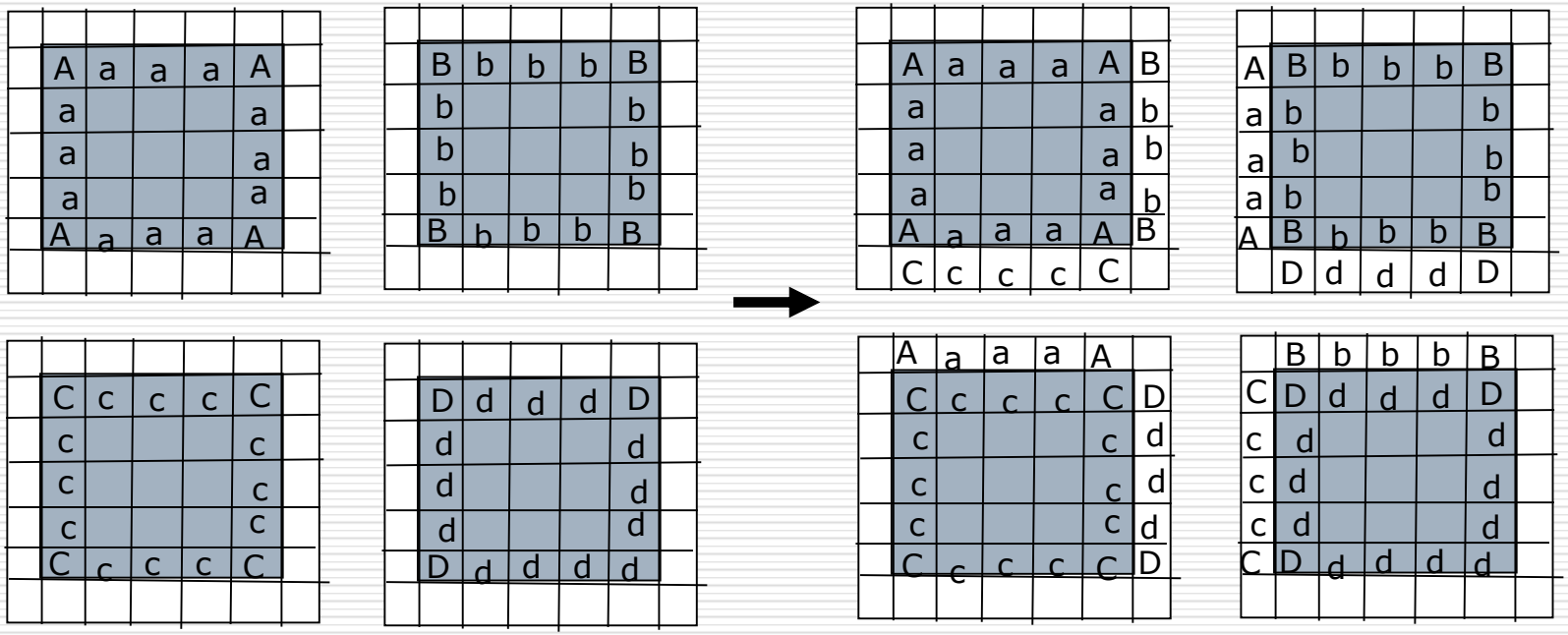
MDS – parallel solution – 1st step

- Communication of boundary data



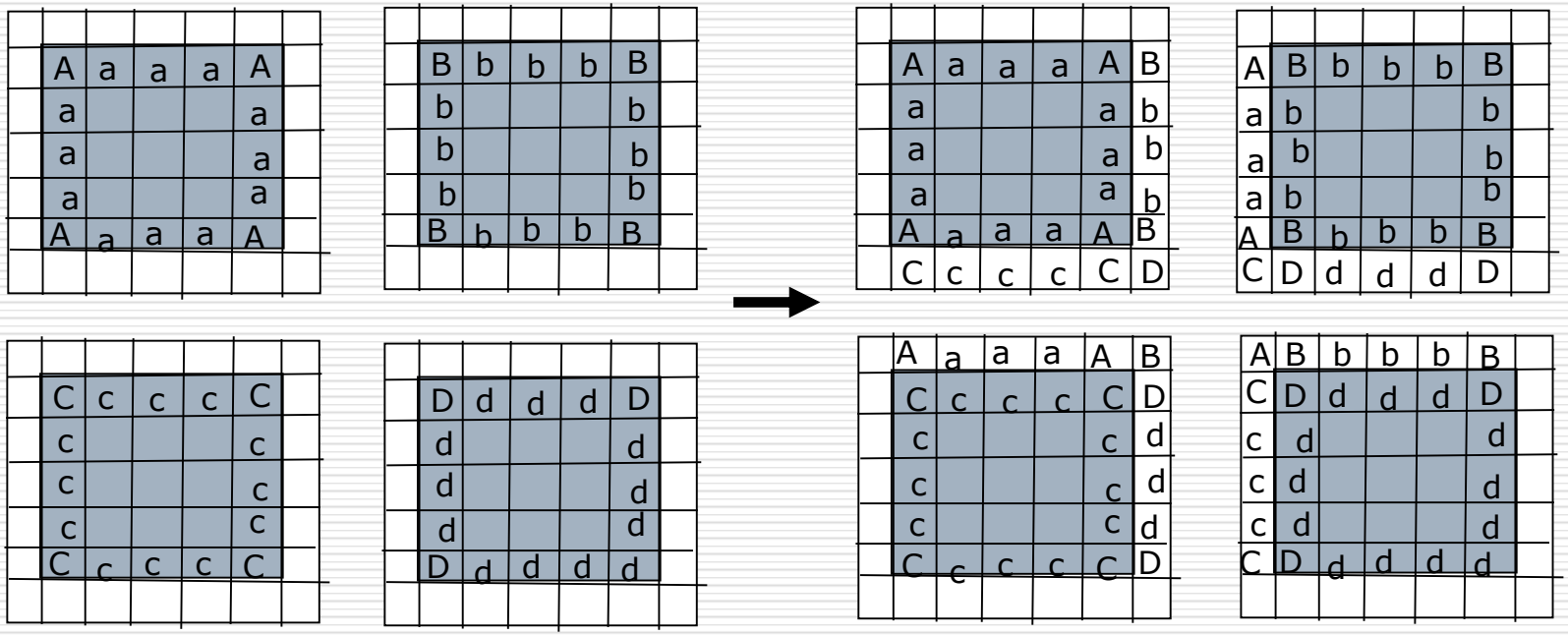
MDS – parallel solution – 1st step

- Communication of boundary data



MDS – parallel solution – 1st step

- Communication of boundary data



Can be achieved by ?

Shift left, shift right, shift up, shift down

MDS – parallel solution – 1st step

Left shift

```
nsend = 0;
for(i=0; i<local_cellsx; i++){
  for each particle p in cell (i, 1){
    pack position of p in sbuf
    nsend += 2
  }
}

MPI_Sendrecv(sbuf, nsend, …, left,..
                        rbuf, max_particles*2, …, right, &status);
MPI_Getcount(status, MPI_DOUBLE, &nrecv);
particles = nrecv/2;
for(i=0; i<particles; i++){
  read (x,y) from next 2 positions in rbuf;
  add (x,y) to particles[local_particle_count+i];
  determine cell k, l for the particle
  Add it to list (k, l);
}
```

MDS – parallel solution – 2nd step

Update:

- Similar to sequential program.
- A processor has all the required information for calculating F_i for all its particles
- Thus new position and velocity determined.
- If new position belongs to the same cell in the same processor, do nothing
- If new position belongs to the different cell in the same processor, update link lists for old and new cells.

MDS – parallel solution – 3rd step

- If new position belongs to the different cell in a different processor – **particle migration**

```
for each particle p
  update {position, velocity}
  determine new cell
  if new cell # old cell
    delete p from list of old cell
    if(different processor)
      pack p into appropriate communication buffer
      remove p from particle array
```

Shift left
Shift right
Shift up
Shift down

MDS – parallel solution – 3rd step

- This shifting is a bit different from the previous shifting
- A processor may just act as a transit point for a particle
- Hence particles have to be packed with care

Shift left:

```
MPI_Sendrecv(leftbuf, nsend_left, …, left
                        rbuf, max_size*4, .., right, &status);
MPI_Getcount(status, MPI_DOUBLE, &nrecv);
particles = nrecv/4;

for(i=0; i<particles; i++){
  read next 4 numbers in {x, y vx, vy}
  if(particle in this process)
    add particle to particle array
    determine cell
    add particle to list for the cell
  else
    put data in the appropriate comm. buffer for the next up or down
      shifts
}
```

Force Decomposition

- For computing the total force on an atom due to all the other atoms, the individual force contributions from the other atoms are independent and can be parallelized
- Fine-grained parallelism
- Especially suitable for shared-memory (OpenMP) parallelization

Hybrid Decomposition

- Divide the domain into cells (spatial decomposition)
- Create a parallel thread whose responsibility is to compute interacting forces between every pairs of cells (force decomposition)