



22nd Summer
School on
PARALLEL
COMPUTING

Parallel Algorithms-1

Parallel Domain Decomposition in Physical sciences

Andrew Emerson - [a.emerson@cineca.it](mailto:a.emerson@ Cineca.it)
SuperComputing Applications and Innovation Department





contents

- Introduction
- Particle decomposition
- Domain decomposition
- Case studies
 - Game of Life
 - Classical molecular dynamics and scaling limits
- Final comments



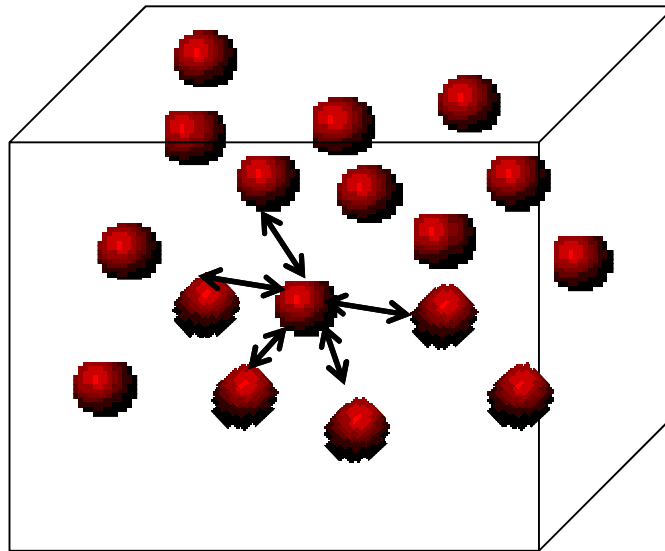
introduction

- *Domain decomposition* refers to a general set of methods for solving a boundary value problem by splitting it into smaller boundary value problems.
- In mathematics often used to solve systems of partial differential equations.
- In computational chemistry and physics used to parallelise simulations over large number of interacting particles.



example

- Consider example of a system of interacting particles



Assume pair-wise, short-range interactions between particles:

$$U = \sum_{i < j} U_{ij}$$



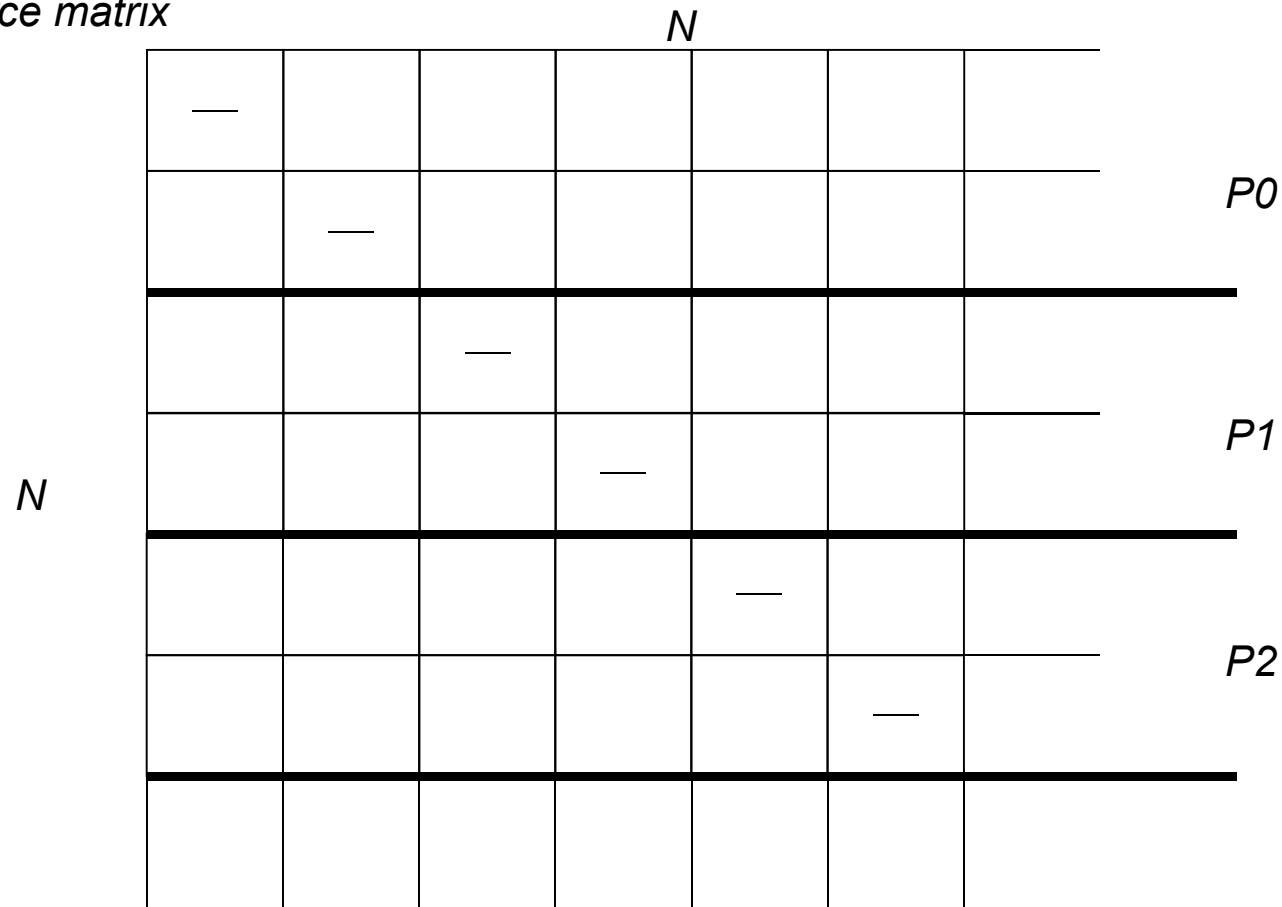
Serial code

```
Utot=0.0
do i=1,N-1
  F(i) = 0.0
  do j=i+1,N
    rij=r(i)-r(j)
    Utot=Utot+Uij
    F(i)=F(i)+force(i,j)
  enddo
enddo
```



Interaction Matrix

force matrix

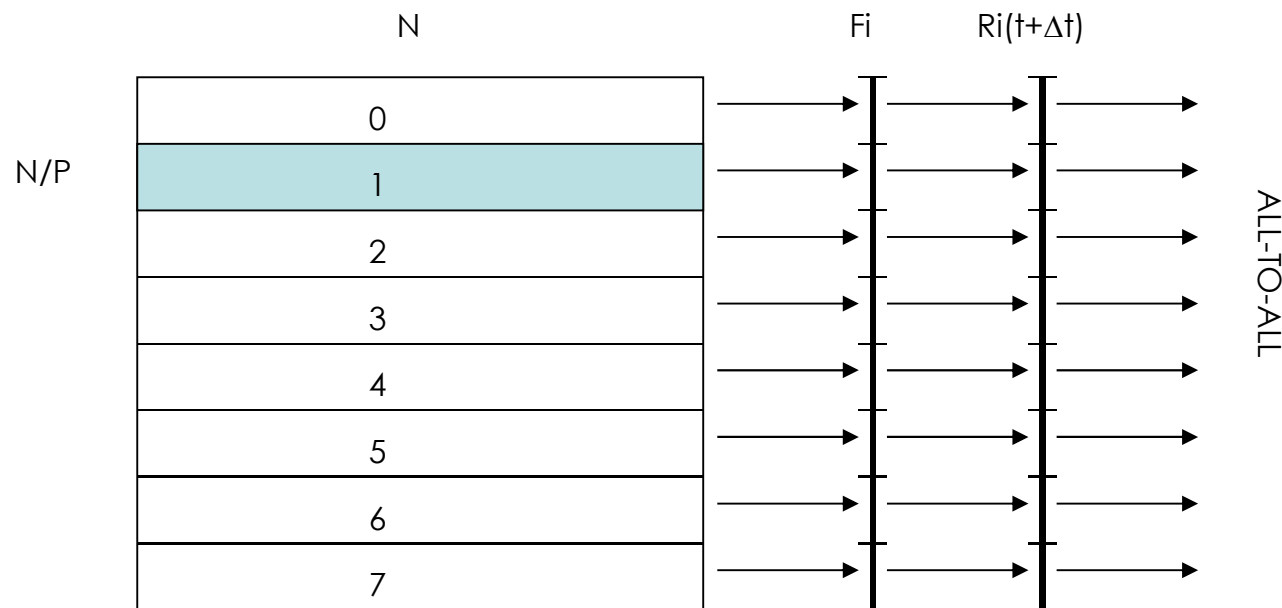


- Double loop is equivalent to an interaction (e.g. force) matrix

- Can parallelise by replicating the data on each processor



Particle decomposition

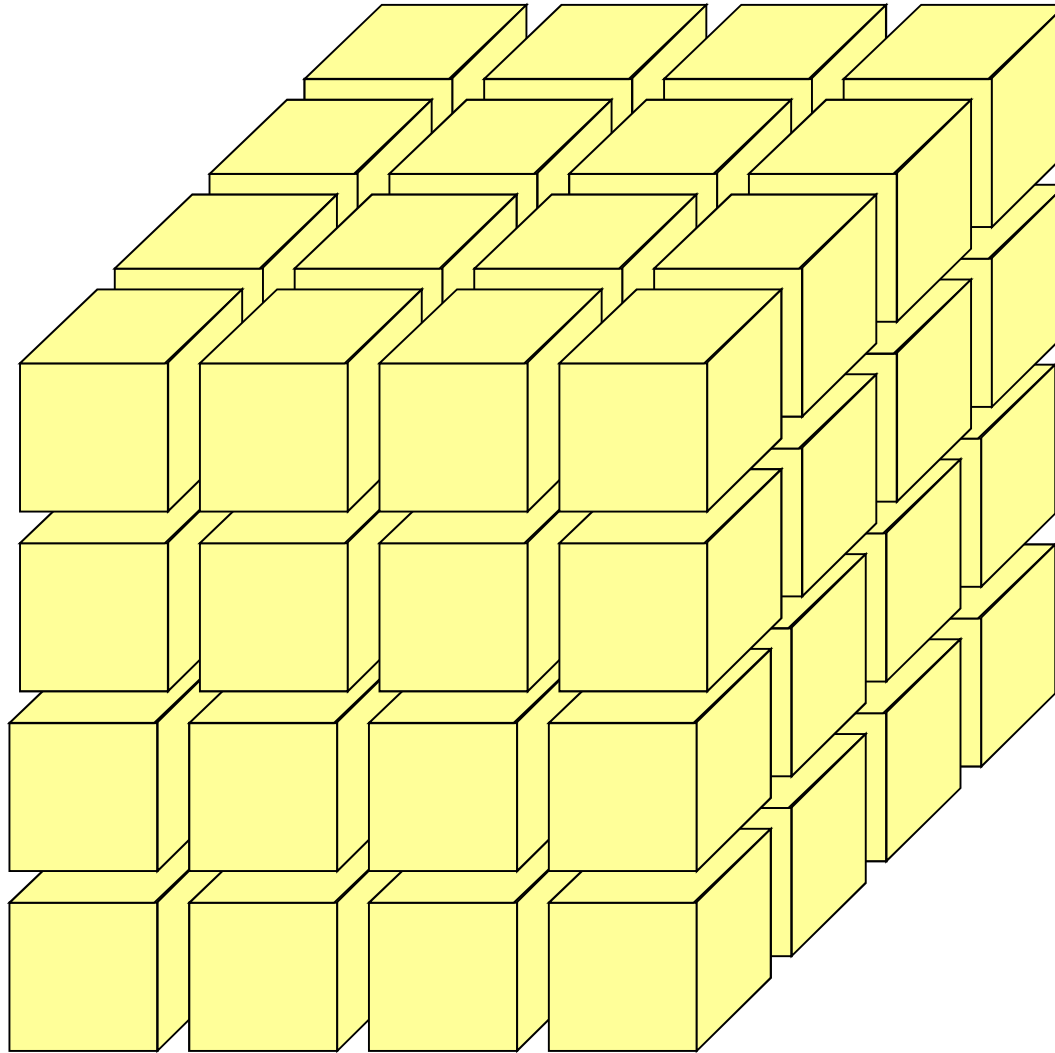


- *In particle decomposition there is a normally at least one all-to-all communication step to gather the processor information*
- *In Molecular Dynamics codes (above) at least three global communications are required.*



Particle decomposition (replicated data)

- Simple to code and scales reasonably well for small N .
- But communication scales as $O(N)$ so at large N the all-to-all communication knocks performance - may not scale over 8-16 processors.
- Replicated data means memory requirements are also high.
- Possible to improve performance by using a block subdivision of the interaction matrix, but scaling is still limited.
- To do better we can exploit the *locality* of the interactions and use domain decomposition.



Divide 3d space up into domains and assign particles to domains. Also assign a processor to each domain.

Domain size usually chosen such that particles in one domain only interact with those in nearest-neighbour domains.

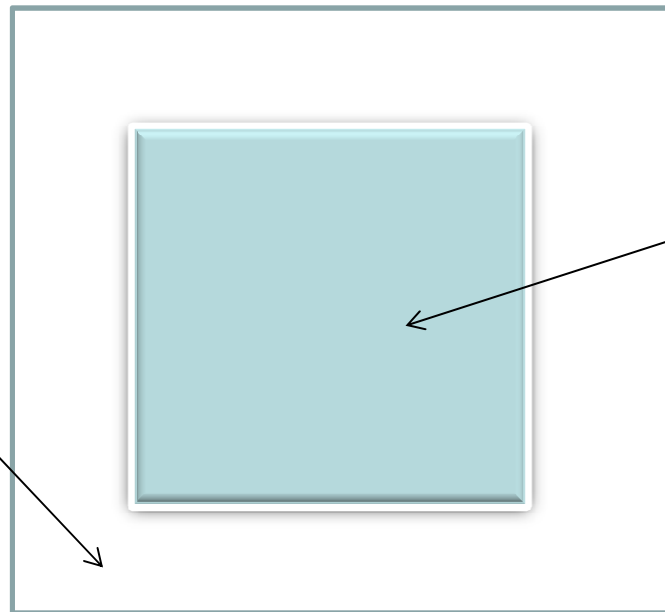
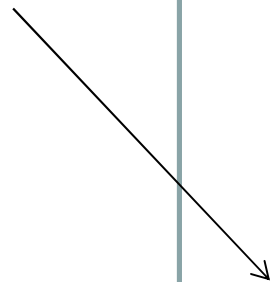


domain decomposition

Each domain will have 2 types of particles:

- 1. Those which can be entirely managed by the processor, i.e. all the interactions are within the domain*
- 2. Those for which the interactions extend outside the domain. Here data have to be sent/received to/from neighbouring domains.*

*Atoms which
need to be
shared with
neighbouring
domains.*



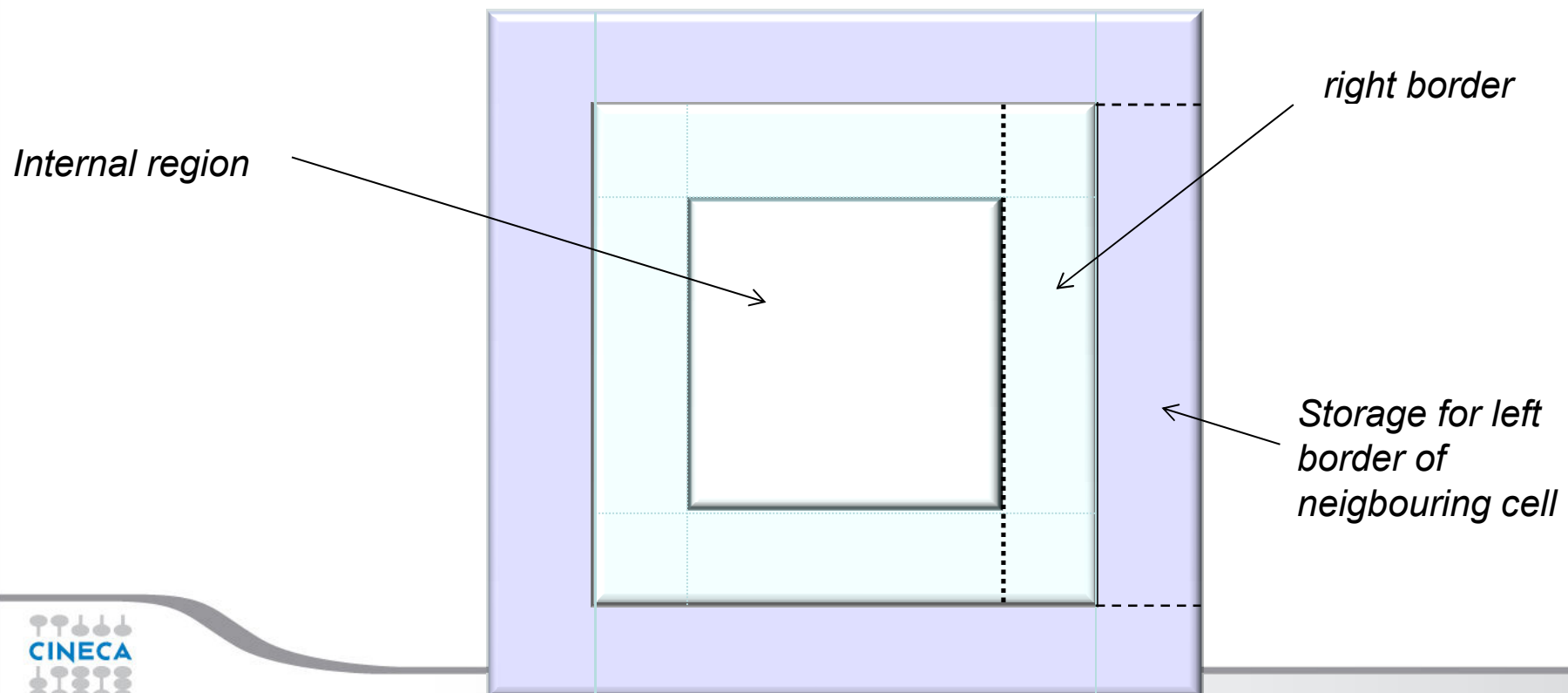
*internal part of
domain*





domain decomposition

- *The difficult part of DD is then to communicate data between a domain and its neighbours.*
- *Convenient for each processor to assign storage also for atoms in neighbouring regions within the cutoff (some times call “ghost” or “halo” regions).*



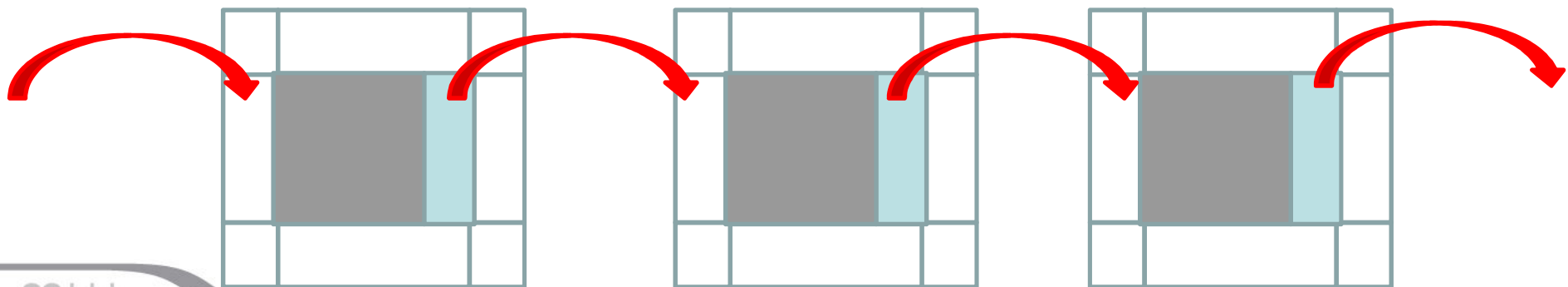


domain decomposition - neighbour communication

- neighbour coords in each dimension conveniently exchanged via `mpi_cart_shift` and `mpi_sendrecv` calls
- First pass, x-direction, left to right,

```
call mpi_cart_shift(mpi_box,1,1,proc_left,proc_right,ierror)
```

```
call mpi_sendrecv(right_side,nright,MPI_INTEGER,proc_right,0,  
halo_left,nleft,MPI_INTEGER,proc_left,0,mpi_box,status,ierror)
```

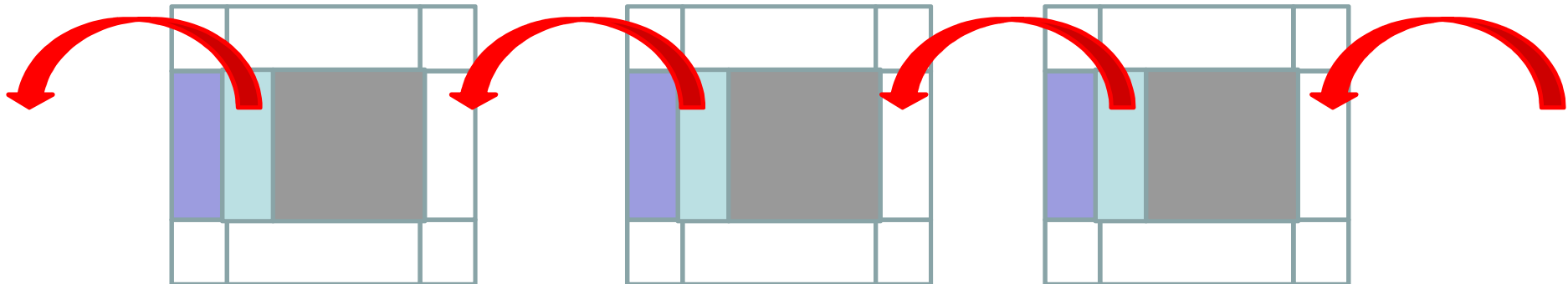




domain decomposition

Then right to left

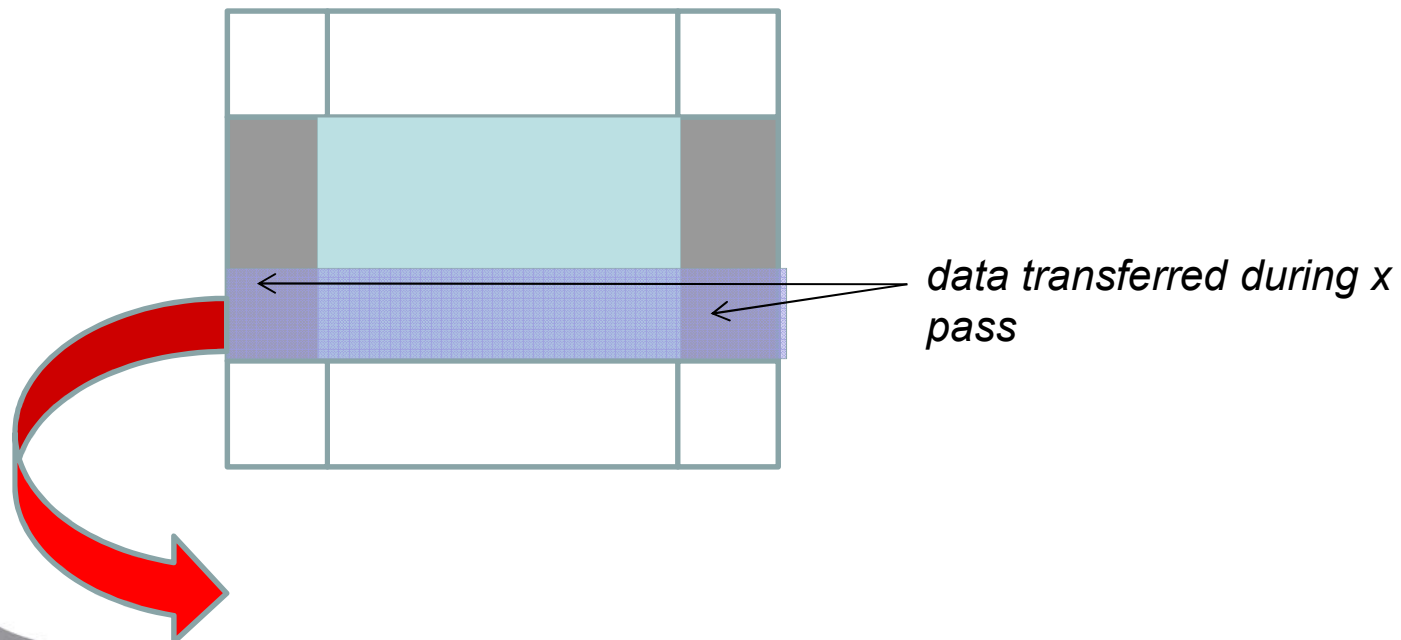
```
call mpi_sendrecv(left_side, nleft, MPI_REAL, proc_left, 0,  
halo_right, nright, MPI_REAL, proc_right, 0, mpi_box, status, ier  
ror)
```





domain decomposition

We can repeat in the y direction but to ensure we transfer the corners we need to include data transferred in the x pass





domain decomposition

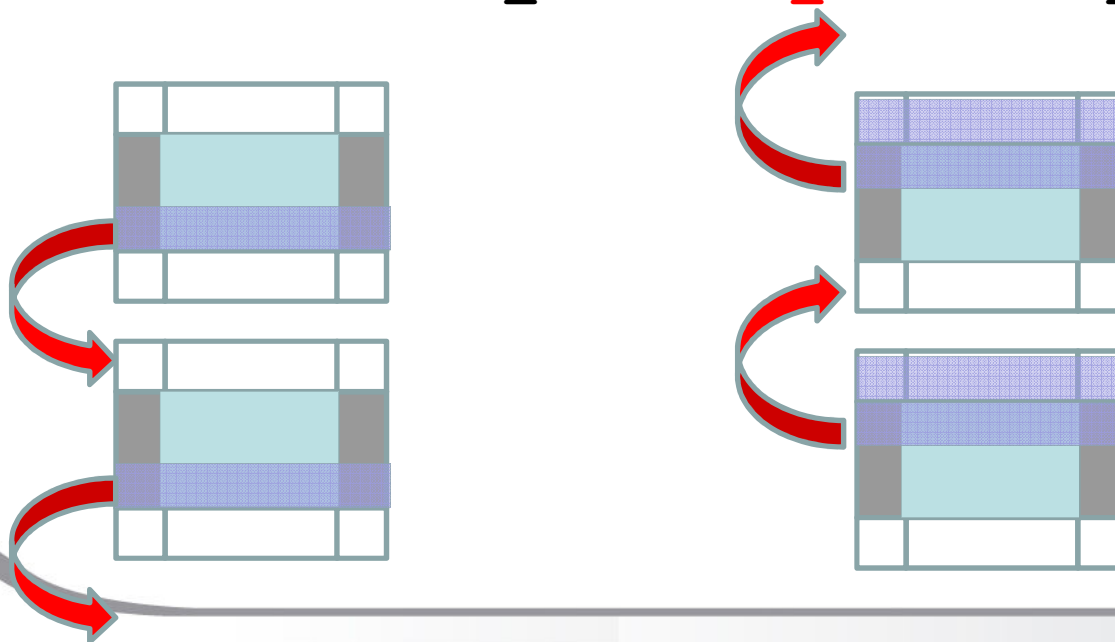
```
call mpi_cart_shift(mpi_box,0,1,proc_up,proc_down,ierror)
```

! top to bottom

```
call mpi_sendrecv(bottom_side,nlower,MPI_FLOAT,proc_down,0,  
halo_top,ntop,MPI_REAL,proc_up,0,mpi_box,status,ierror)
```

! bottom to top

```
call mpi_sendrecv(top_side,ntop,MPI_REAL,proc_up,0,  
halo_bottom,nbottom,MPI_REAL,proc_down,0,mpi_box,status,ierror)
```





domain decomposition

- *Similarly in the z direction, using data transferred in the previous y passes (which includes data transferred in x)*
- *Each processor now has enough information to calculate all the interactions in its domain.*



particle and domain decomposition

- *Compared to PD (or Replicated Data), DD*
 - *Exploits the intrinsic **locality** , minimizing communications (no All-to-All) and memory required per processor*
 - *scalable, for large systems*
 - *can exploit MPI cartesian topology*



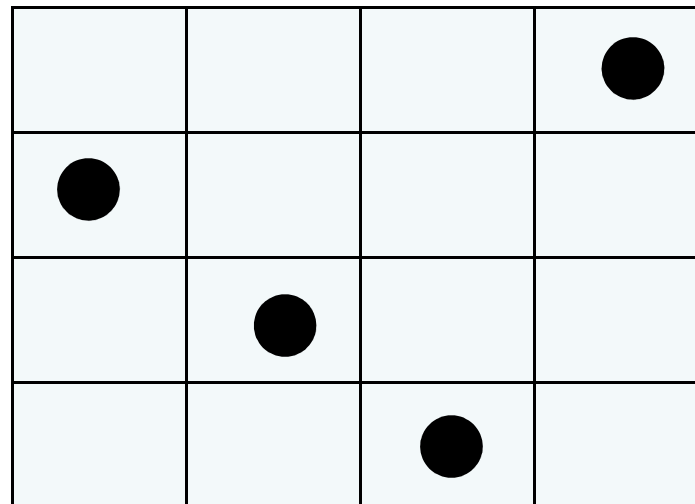
Case study 1 - Game of Life

- A simple 2D cellular automata originally conceived by J. Conway in 1970.
- Based on a few simple rules, able to exhibit complex evolution depending on starting configuration and run time.
- Locality of interactions (i.e. state of neighbouring cells) implies good candidate for parallelization by domain decomposition



Case study 1 - Game of Life

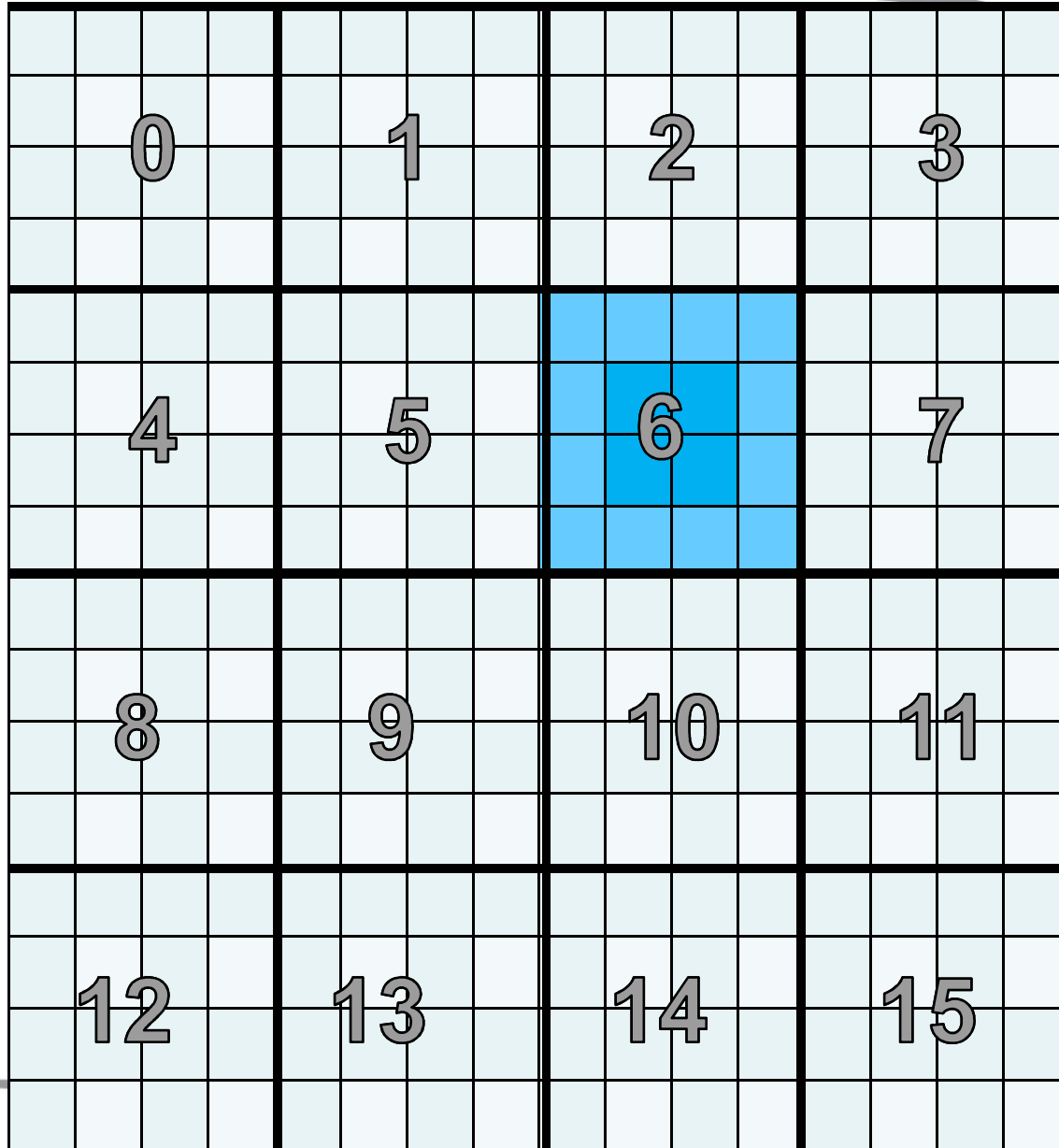
- The system consists of a 2D grid of cells. Cells evolve as follows: in the next generation a cell will
 1. Be **dead** if the cell has < 2 live neighbours (lonely)
 2. Stay **the same** if has exactly 2 neighbours (content)
 3. Be **born** if the cell has exactly 3 live neighbours
 4. **Die** if > 3 live neighbours (overcrowding)





Game of Life - strategy

- Maintain two boards, one for the current generation and one for the next generation.
- Create a master-slave model: the master (e.g. rank 0) will generate the original configuration, collect results from other procs and write output to file.
- Partition the 2D array amongst the processors.
- Generate a cartesian topology.
- Each processor allocates storage for its own cells + halo regions (for neighbours).
- Procs update their own cells, then communicate boundaries to neighbouring cells. Calculate remaining cells.
- Master gathers data from procs, updates current board, writes to file → *next generation*.



Game of Life decomposition



Game of Life - implementation hints

- cartesian topology

```
integer dlength(2),reorder
logical periods(0:1)
call mpi_init(ierr)
call mpi_comm_size(MPI_COMM_WORLD, size, ierr)
```

! Cartesian topology for grid

```
call mpi_dims_create(size,2,dlength,ierr)
periods(0)=.true.
periods(1)=.true.
reorder=1
call mpi_cart_create(MPI_COMM_WORLD,2,dlength,periods,
reorder,mpi_grid, ierr)
```



Game of Life - implementation hints

- MPI derived data types for transferring data to neighbours

```
integer mpi_block, coltype, rowtype
```

```
! define a row type
```

```
call mpi_type_vector  
(local,1,nrow,MPI_INTEGER,rowtype, ierror)  
call mpi_type_commit(rowtype,ierror)
```

```
! find up and down neighbours
```

```
call  
mpi_cart_shift(mpi_grid,0,1,proc_up,proc_down,ierror)
```

```
! send row to down proc, receive row data from up proc
```

```
call  
mpi_sendrecv(locarray(nrow,1),1,rowtype,proc_down,0, &  
edge_up,ncol,MPI_INTEGER,proc_up,0,mpi_grid,status,ier  
ror)
```

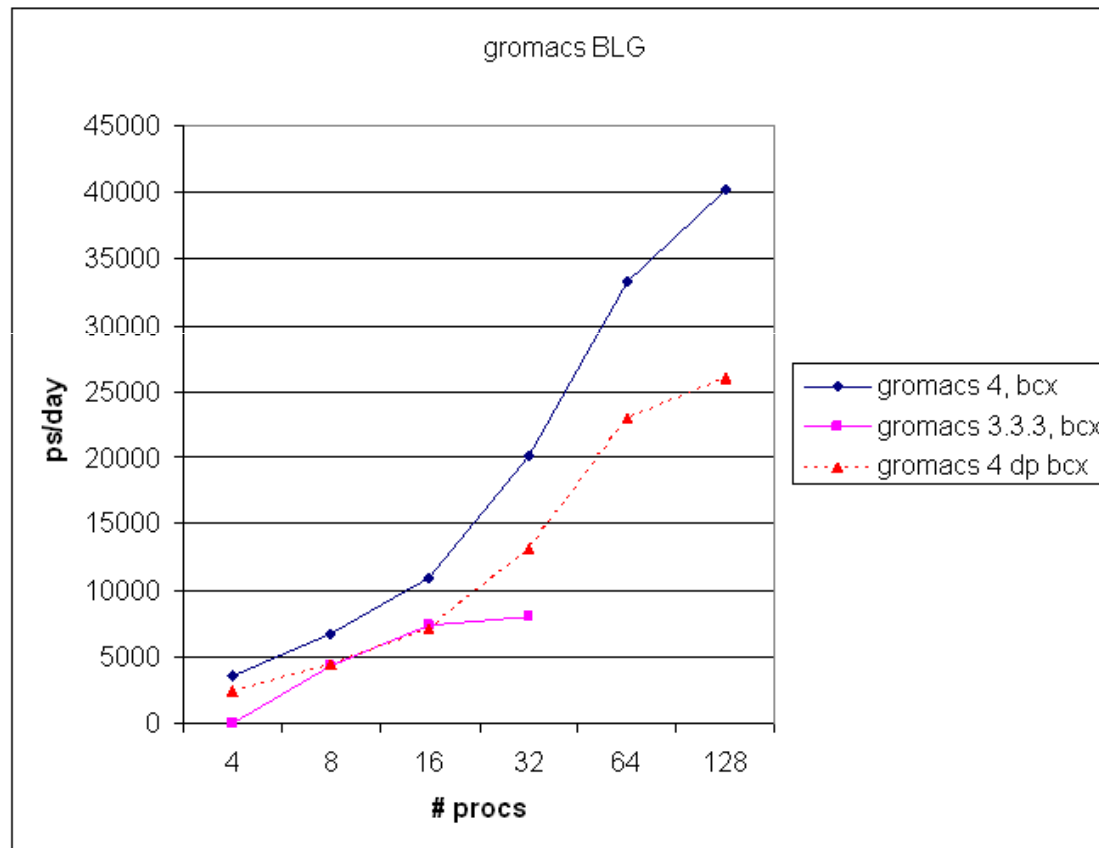



Case study 2 - Classical molecular dynamics

- Molecular dynamics (MD) programs model physical or chemical systems by simulating the movements of interacting atoms or molecules.
- For realistic models, many tens of thousands or even millions of interacting atoms may need to be simulated.
- All common MD programs (e.g. GROMACS, NAMD, DL_POLY, etc) rely on DD for parallelisation.



Particle and domain decomposition comparison



- Gromacs v3.3 used particle/force decomposition as a parallel scheme.

- DD was introduced into Gromacs 4.x



domain decomposition and molecular dynamics - limitations

- System size
 - The number of particles may be fixed by the physical system (e.g. the protein size). This will limit scaling since there must be at least one particle/processor (in practice many more)
- Inhomogeneity
 - Particles can move between domains → possible load imbalance. Modern programs have Dynamic Load Balancing to adjust no. of particles/domain.
- Long-range forces and FFT
 - In practice MD programs often stop scaling due to the FFT used for the calculation of long-range forces, but even without FFT scaling stops eventually.



Without FFT

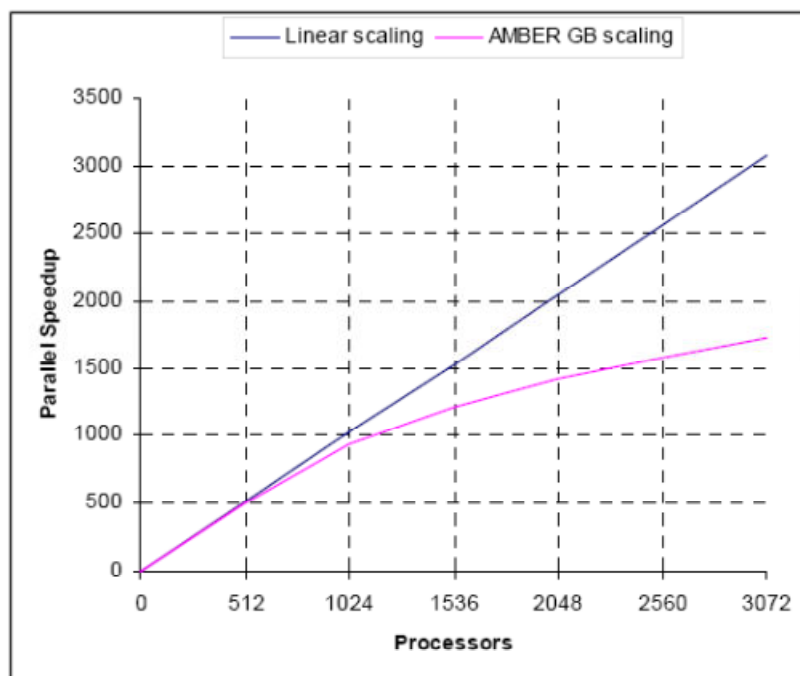


Figure 1. Parallel scaling of AMBER on Blue Gene. The experiment is with an implicit solve (GB) model of 120,000 atoms (Aon benchmark)

With FFT

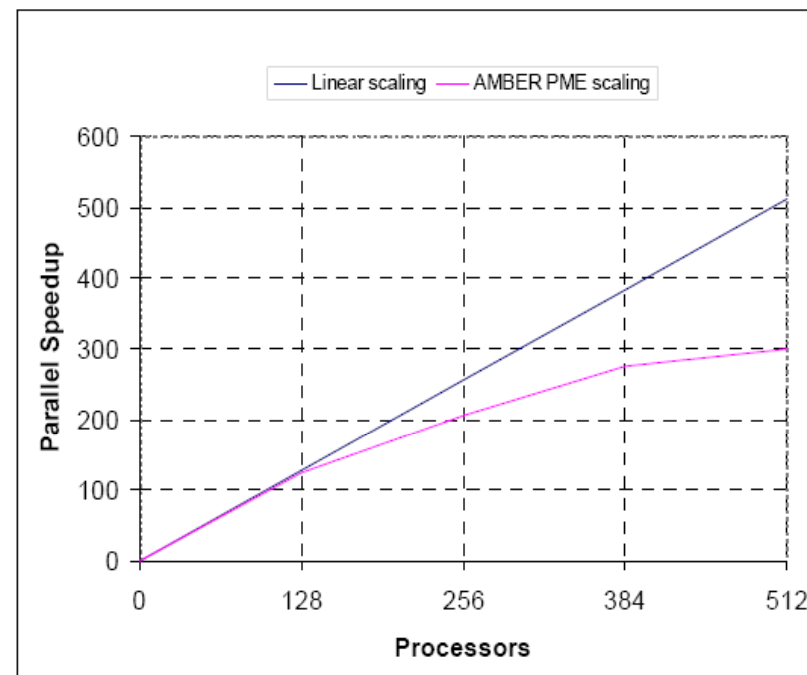


Figure 2. Parallel scaling of AMBER on Blue Gene. The experiment is with an explicit solvent (PME) model of 290,000 atoms (Rubisco).

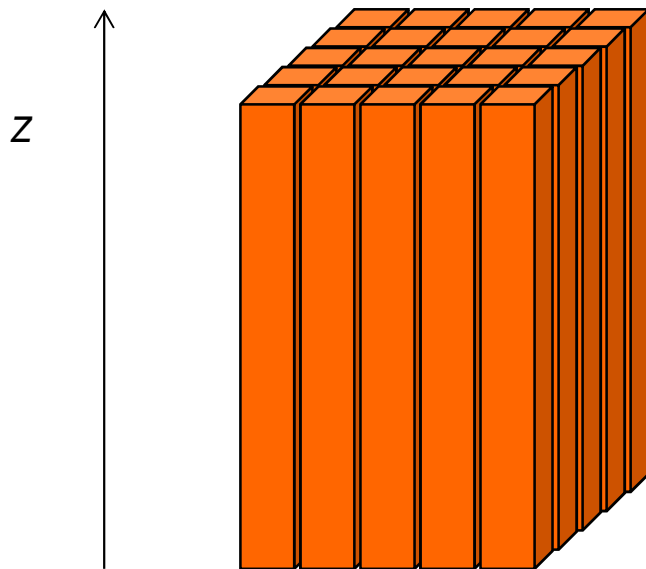
*Life Sciences Molecular Dynamics Applications on the IBM System Blue Gene Solution:
Performance Overview,*

http://www-03.ibm.com/systems/resources/systems_deepcomputing_pdf_lsmdabg.pdf



1D and 2D decompositions

- Sometimes convenient to use 2D or even 1D domain decompositions (send more data but to fewer processors).
- Example includes the FFT used solve the equations for long-range interactions in many simulations (e.g. electrostatic forces in MD).



3D FFT is very inefficient for many processors (or small N) because of all-to-all global communications (MPI_AlltoAll)

Instead use 2D decomposition of thin columns or “pencils”:

In this way the first 1D part of the 3D can be done within a single processor (e.g. along z) to avoid extra communication



final comments

- domain decomposition commonly used in computational chemistry, physics and astrophysics to distribute physical domain over processors by exploiting locality of interactions
- can be quite complex to program but MPI has many useful commands to simplify programming.