計算科学技術特論A

Parallelization of molecular dynamics

2019年 7月 11日

Jaewoon Jung
(RIKEN Center for Computational Science)

Overview of MD

Molecular Dynamics (MD)

1. Energy/forces are described by classical molecular mechanics force field.

2. Update state according to equations of motion

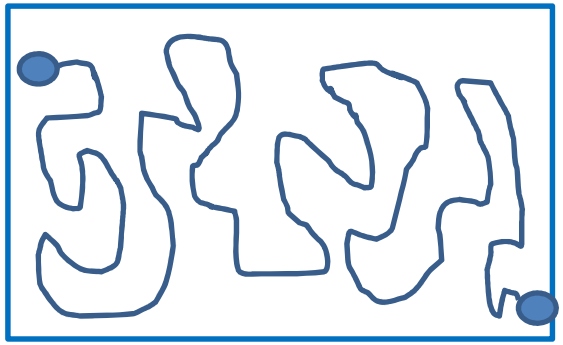
$$\dot{\mathbf{r}}_i = \frac{\mathbf{p}_i}{m_i}$$
$$\dot{\mathbf{p}}_i = \mathbf{F}_i$$

Equation of motion

$$\mathbf{p}_i\left(t + \frac{1}{2}\Delta t\right) = \mathbf{p}_i\left(t - \frac{1}{2}\Delta t\right) + \mathbf{F}_i(t)\Delta t$$

$$\mathbf{r}_i(t + \Delta t) = \mathbf{r}_i(t) + \frac{\tilde{\mathbf{p}}_i\left(t + \frac{1}{2}\Delta t\right)\Delta t}{m_i}$$

Integration



Long time MD trajectory => Ensemble generation

Long time MD trajectories are important to obtain thermodynamic quantities of target systems.

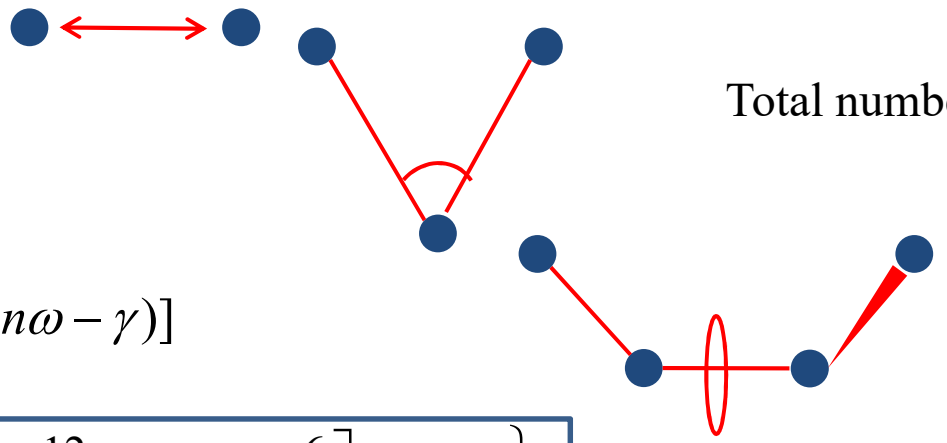
Potential energy in MD

$$E_{\text{total}} = \sum_{\text{bonds}} k_b (b - b_0)^2$$

$$+ \sum_{\text{angles}} k_a (\theta - \theta_0)^2$$

$$+ \sum_{\text{dihedrals}} V_n [1 + \cos(n\omega - \gamma)]$$

$$+ \sum_{j=1}^{N-1} \sum_{i=j+1}^{N} \left\{ \varepsilon_{ij} \left[\left(\frac{r_{0ij}}{r_{ij}} \right)^{12} - 2 \left(\frac{r_{0ij}}{r_{ij}} \right)^{6} \right] + \frac{q_i q_j}{r_{ij}} \right\}$$



O(*N*)

Total number of particles

O(*N*)

O(*N*)

O(N^2)

Main bottleneck in MD

$$\sum_{|i-j|<R} \left\{ \varepsilon_{ij} \left[\left(\frac{r_{0ij}}{r_{ij}} \right)^{12} - 2 \left(\frac{r_{0ij}}{r_{ij}} \right)^{6} \right] + \frac{q_i q_j \operatorname{erfc}(\alpha r_{ij})}{r_{ij}} \right\}$$

Real space, O(C*N*)

$$+ \sum_{\mathbf{k} \neq 0} \frac{\exp(-\mathbf{k}^2 / 4\alpha^2)}{\mathbf{k}^2} \operatorname{FFT}(Q(\mathbf{k}))$$

Reciprocal space, O(*N*log*N*)

Non-bonded interaction

1. Non-bond energy calculation is reduced by introducing cutoff

$$\sum_{j=1}^{N-1}\sum_{i=j+1}^{N}\left\{\varepsilon_{ij}\left[\left(\frac{r_{0ij}}{r_{ij}}\right)^{12}-2\left(\frac{r_{0ij}}{r_{ij}}\right)^{6}\right]+\frac{q_i q_j}{4\pi\varepsilon r_{ij}}\right\}$$

O(N²)

$$\sum_{|i-j|<R}^{N}\left\{\varepsilon_{ij}\left[\left(\frac{r_{0ij}}{r_{ij}}\right)^{12}-2\left(\frac{r_{0ij}}{r_{ij}}\right)^{6}\right]\right\}+\mathrm{U}_{elec}$$

O(N¹)

2. The electrostatic energy calculation beyond cutoff will be done in the reciprocal space with FFT

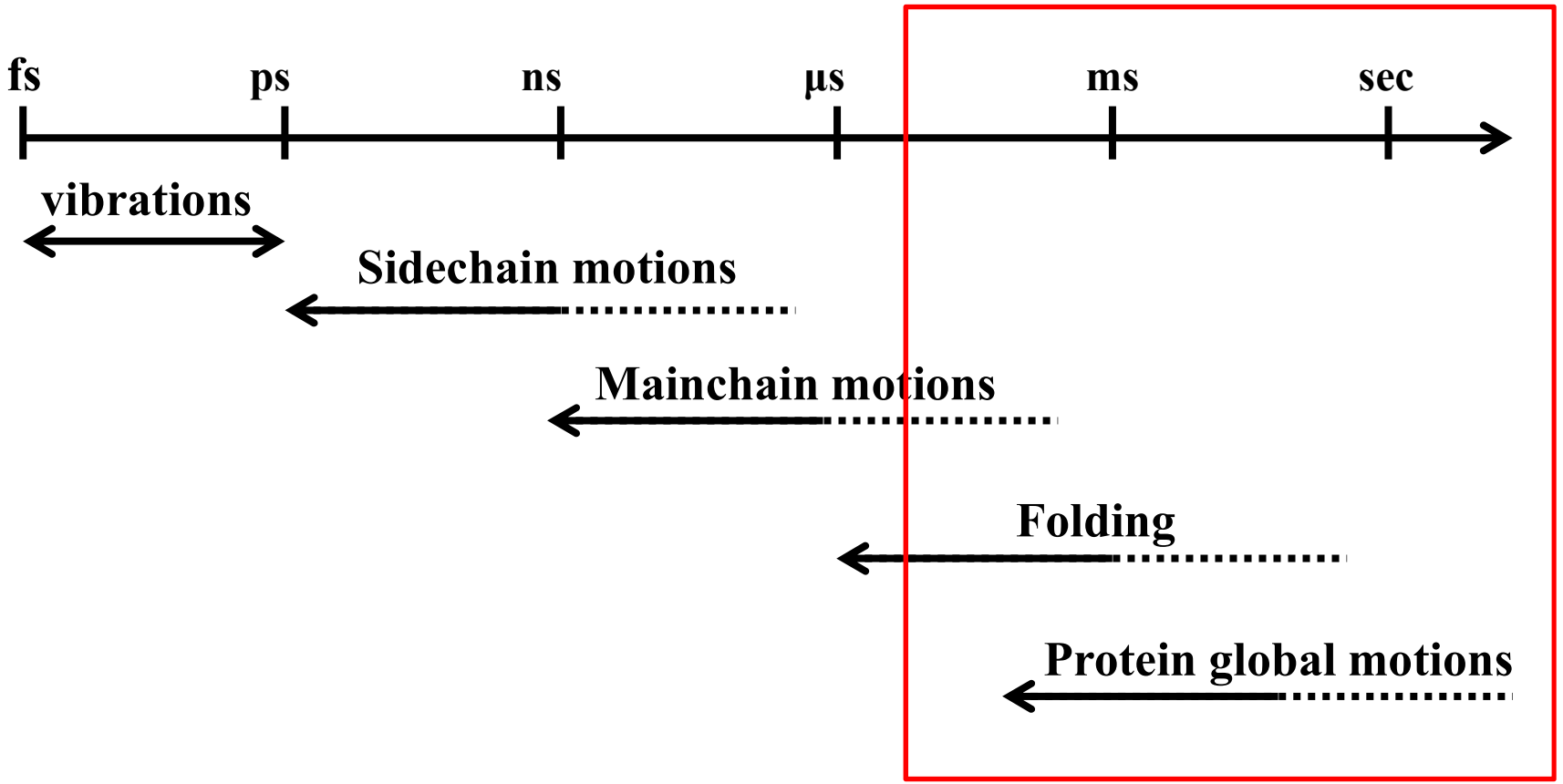
$$\mathrm{U}_{elec}=\underbrace{\sum_{|i-j|<R}\frac{q_i q_j}{4\pi\varepsilon_0}\frac{\mathrm{erfc}(\alpha_{ij})}{r_{ij}}}_{\text{Real part}}+\underbrace{\frac{2\pi}{V}\sum_{\mathbf{G}\neq 0}\frac{\exp(-|\mathbf{G}|^2/4\alpha^2)}{|\mathbf{G}|^2}\sum_{ij}\frac{q_i q_j}{4\pi\varepsilon_0}\cos(\mathbf{G}\cdot r_{ij})}_{\text{Reciprocal part}}-\underbrace{\sum_{i}\frac{q_i q_i}{4\pi\varepsilon_0}\frac{\alpha}{\sqrt{\pi}}}_{\text{Self energy}}$$

3. Further, it could be reduced by properly distributing over parallel processes, in particular good domain decomposition scheme.

Difficulty to perform long time MD simulation

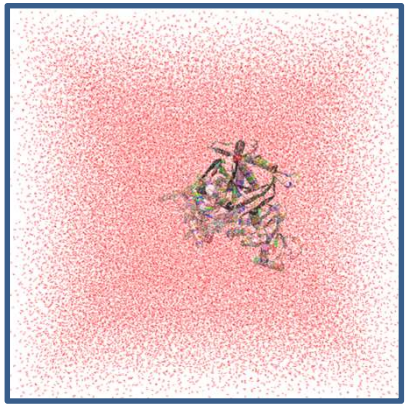
1. One time step length (Δt) is limited to 1-2 fs due to vibrations.

2. On the other hand, biologically meaningful events occur on the time scale of milliseconds or longer.



How to accelerate MD simulations?
=> Parallelization

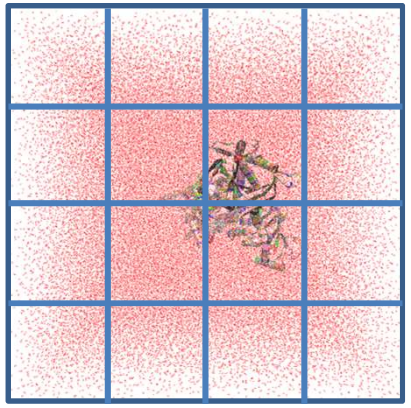
Serial



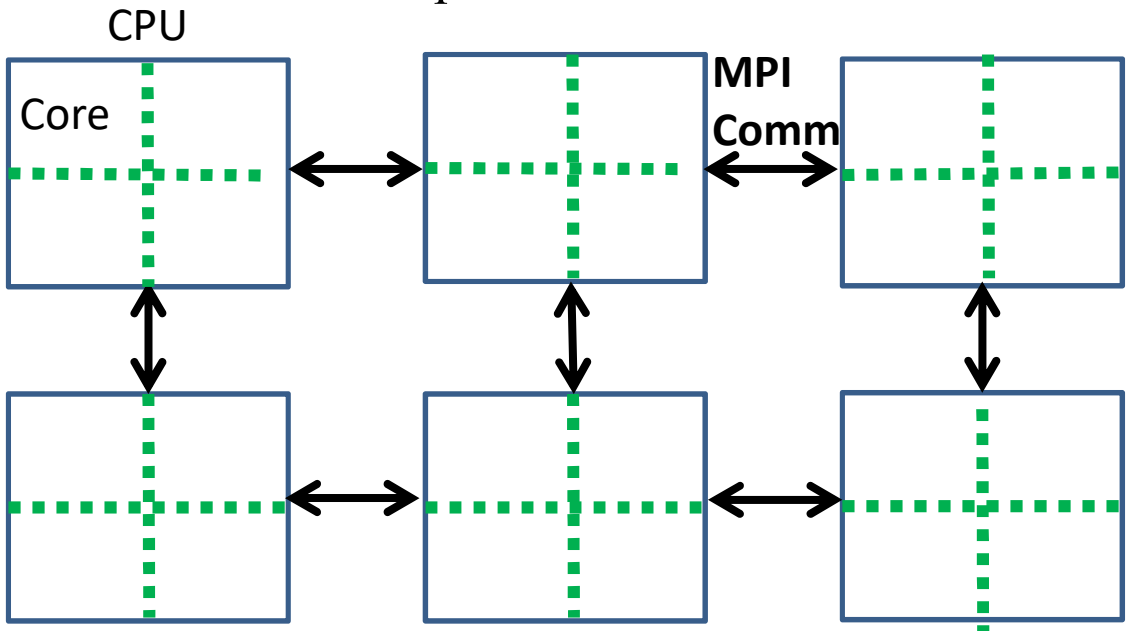
1cpu

X16?

Parallel



16 cpus

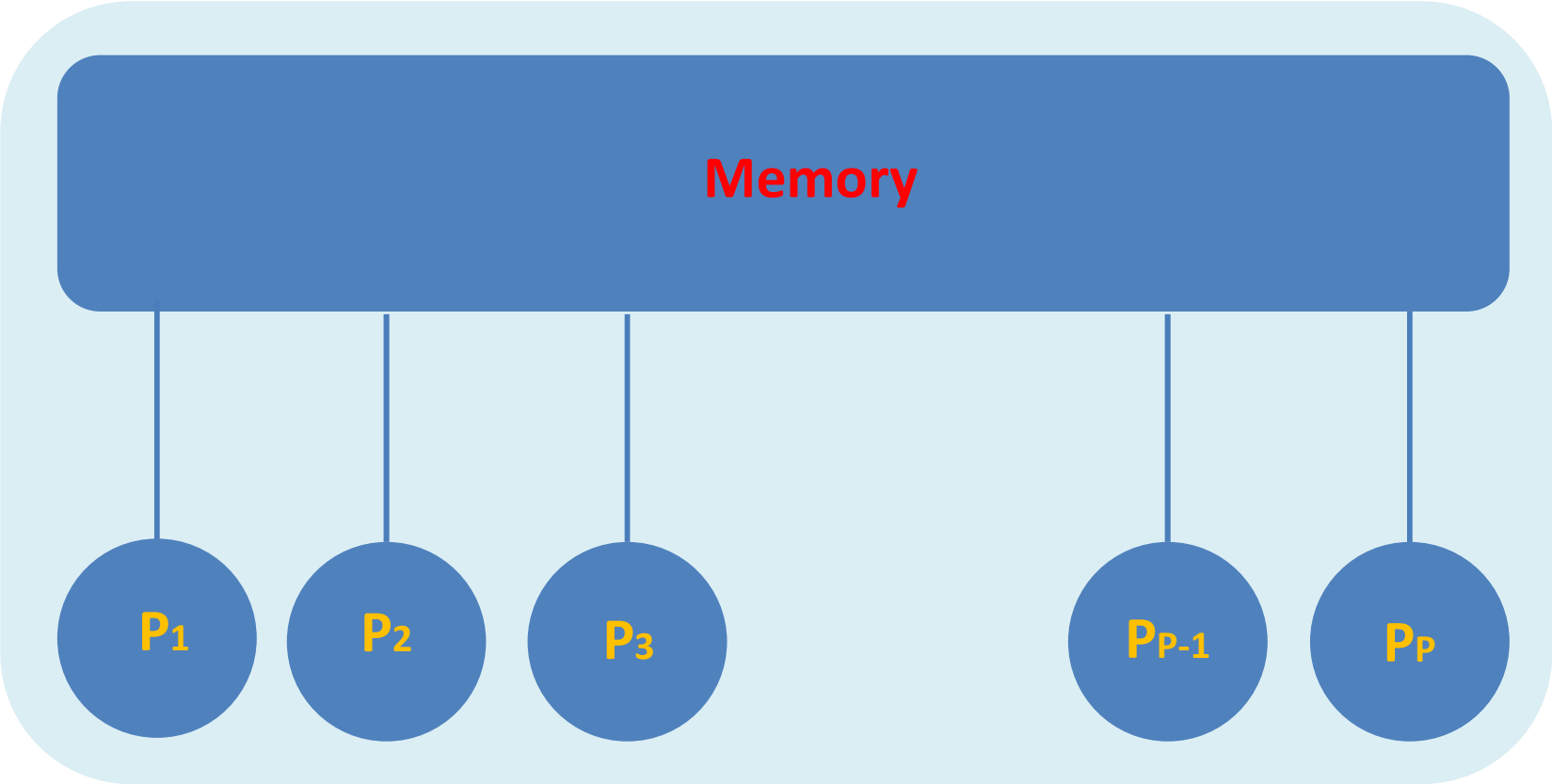


Good Parallelization ;

1) Small amount of computation in one process
2) Small communicational cost

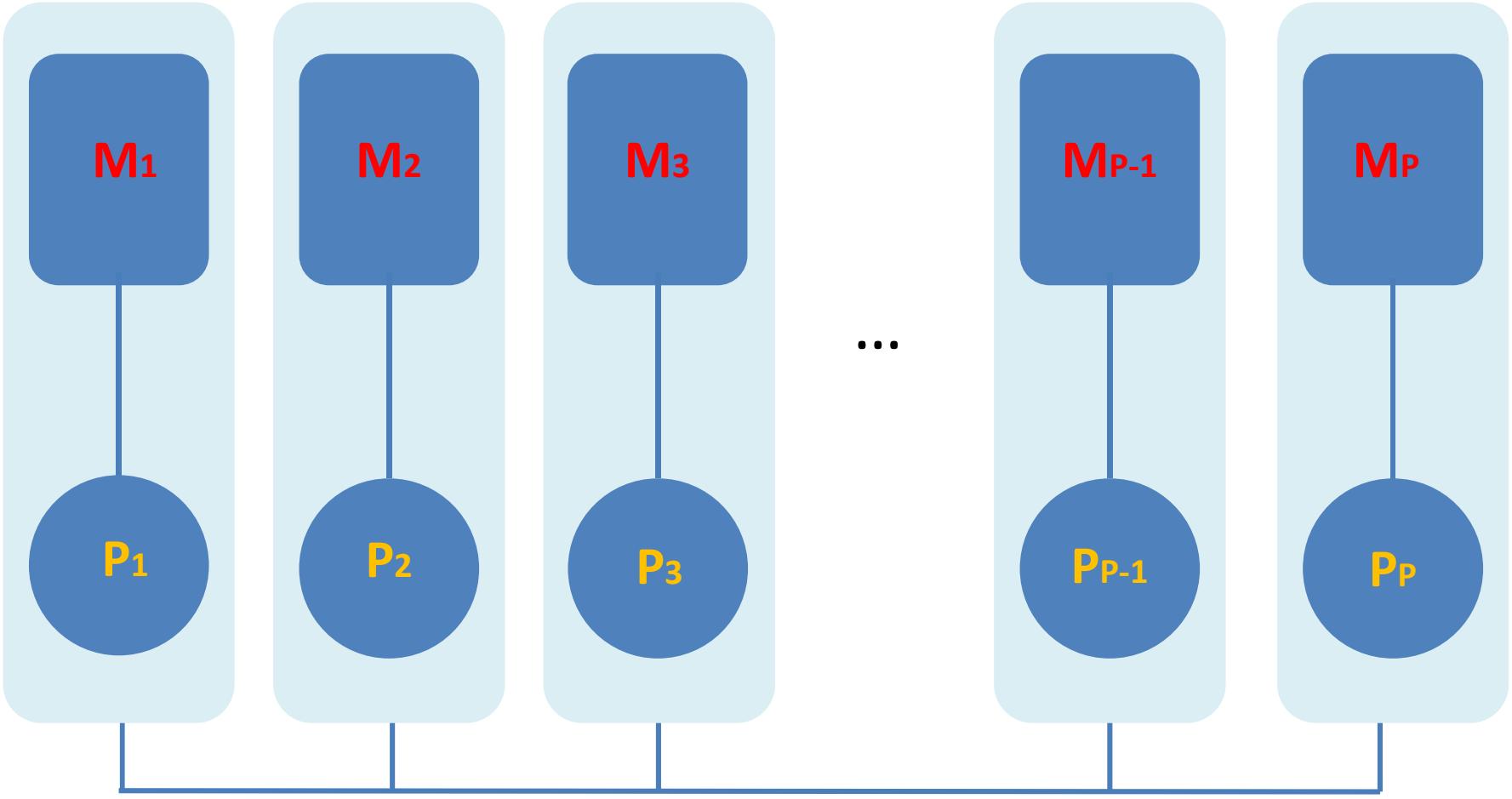
Parallelization

Shared memory parallelization (OpenMP)



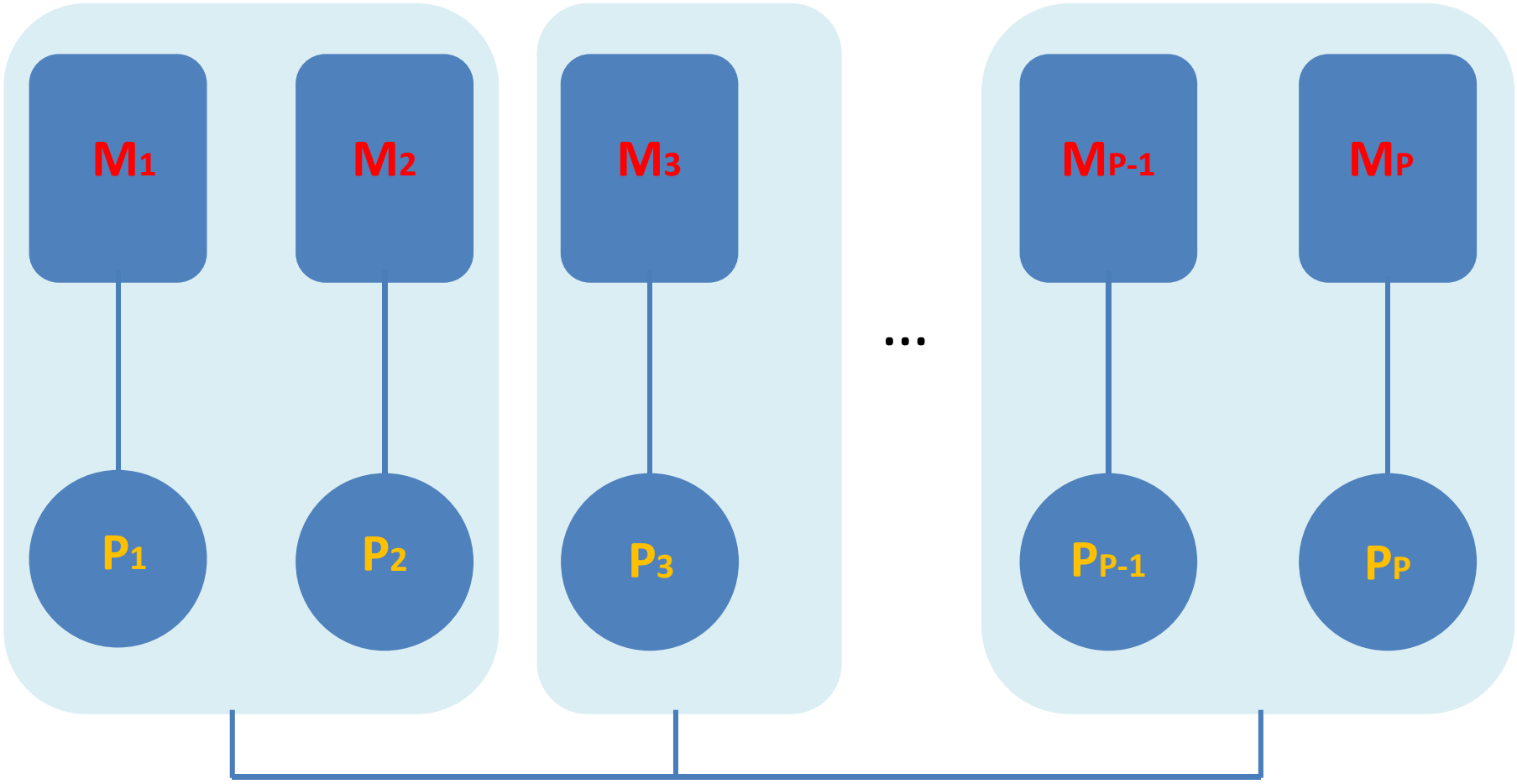
- All processes share data in memory
- For efficient parallelization, processes should not access the same memory address.
- It is only available for multi-processors in a physical node

Distributed memory parallelization (MPI)



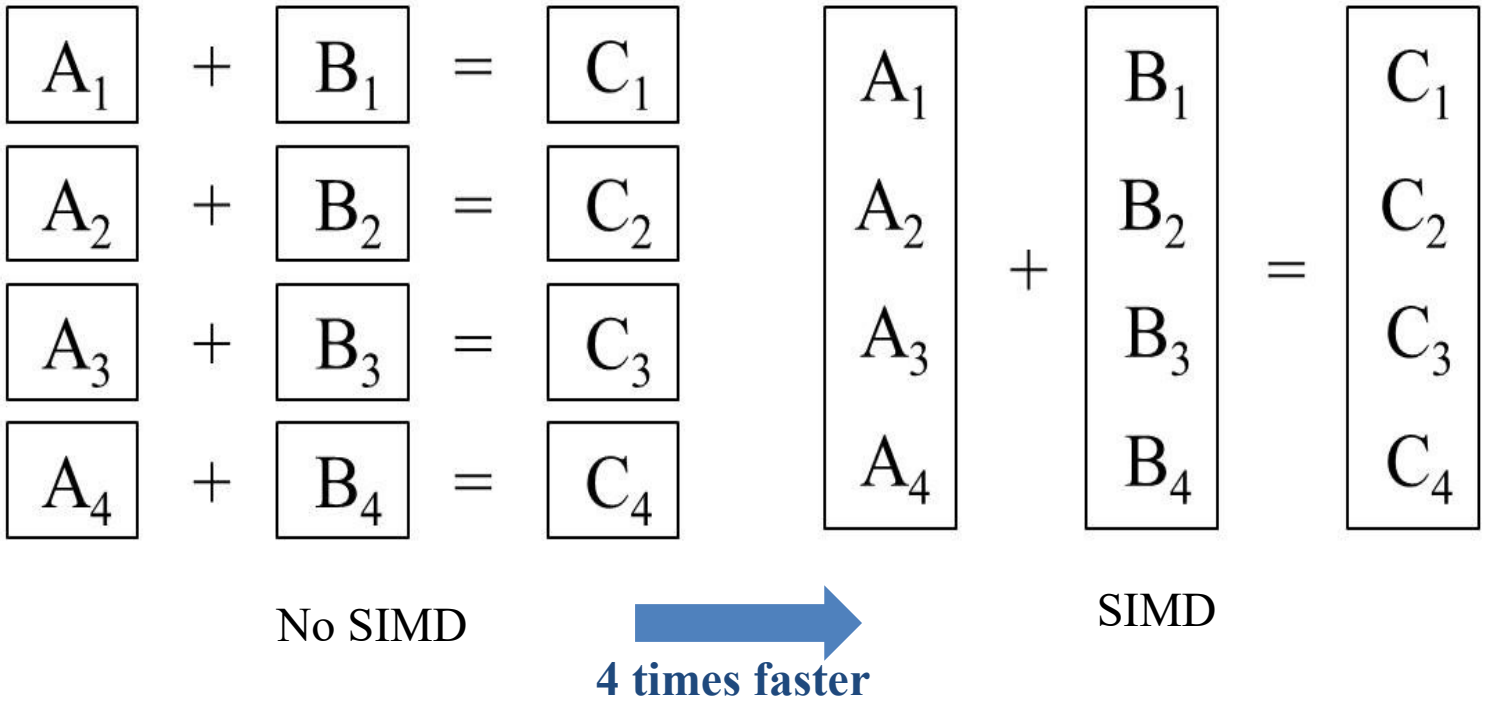
- Processors do not share data in memory
- We need to send/receive data via communications
- For efficient parallelization, the amount of communication data should be minimized

Hybrid parallelization (MPI+OpenMP)



- Combination of shared memory and distributed memory parallelization.
- It is useful for minimizing communicational cost with very large number of processors

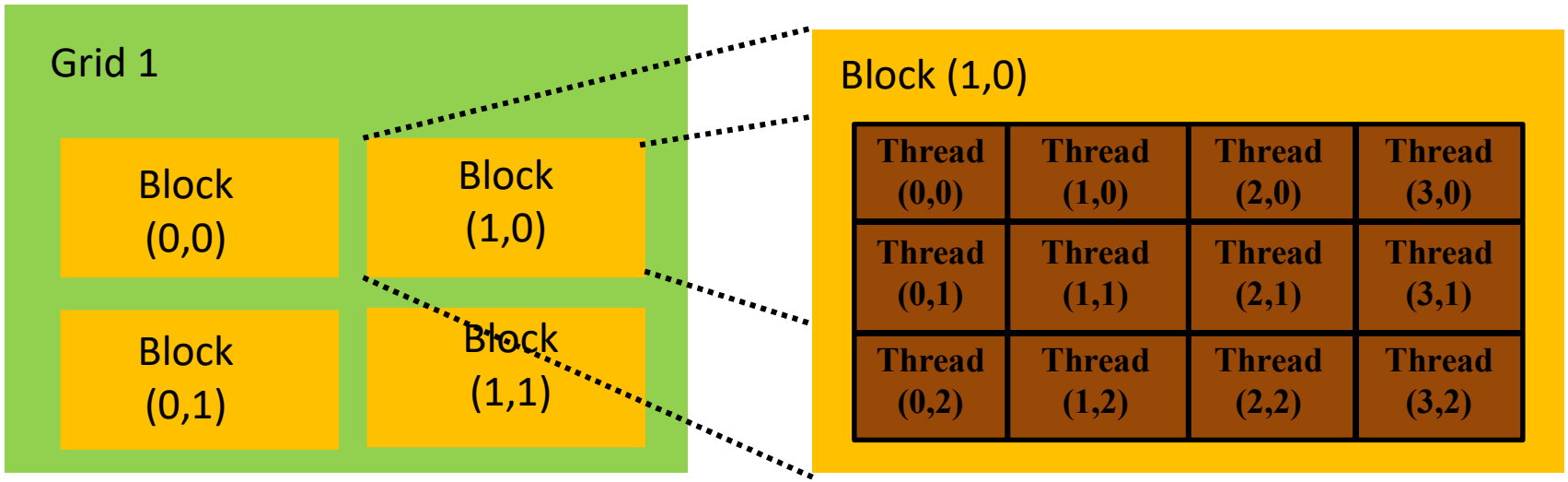
SIMD (Single instruction, multiple data)



- Same operation on multiple data points simultaneously
- Usually applicable to common tasks like adjusting graphic image or volume
- In most MD programs, SIMD becomes the one of the important topics to increase the performance

SIMT (Single instruction, multiple threads)

- SIMD combined with multithreading
- SIMT execution model is usually implemented on GPUs and related with GPGPU (General Purpose computing on Graphics Processing Units)
- Currently, CUDA allows 32 threads for SIMT (warp size =32)

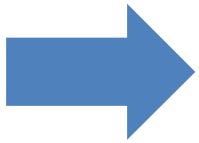


MPI Parallelization of MD
(non-bonded interaction in real space)

Parallelization scheme 1: Replicated data approach

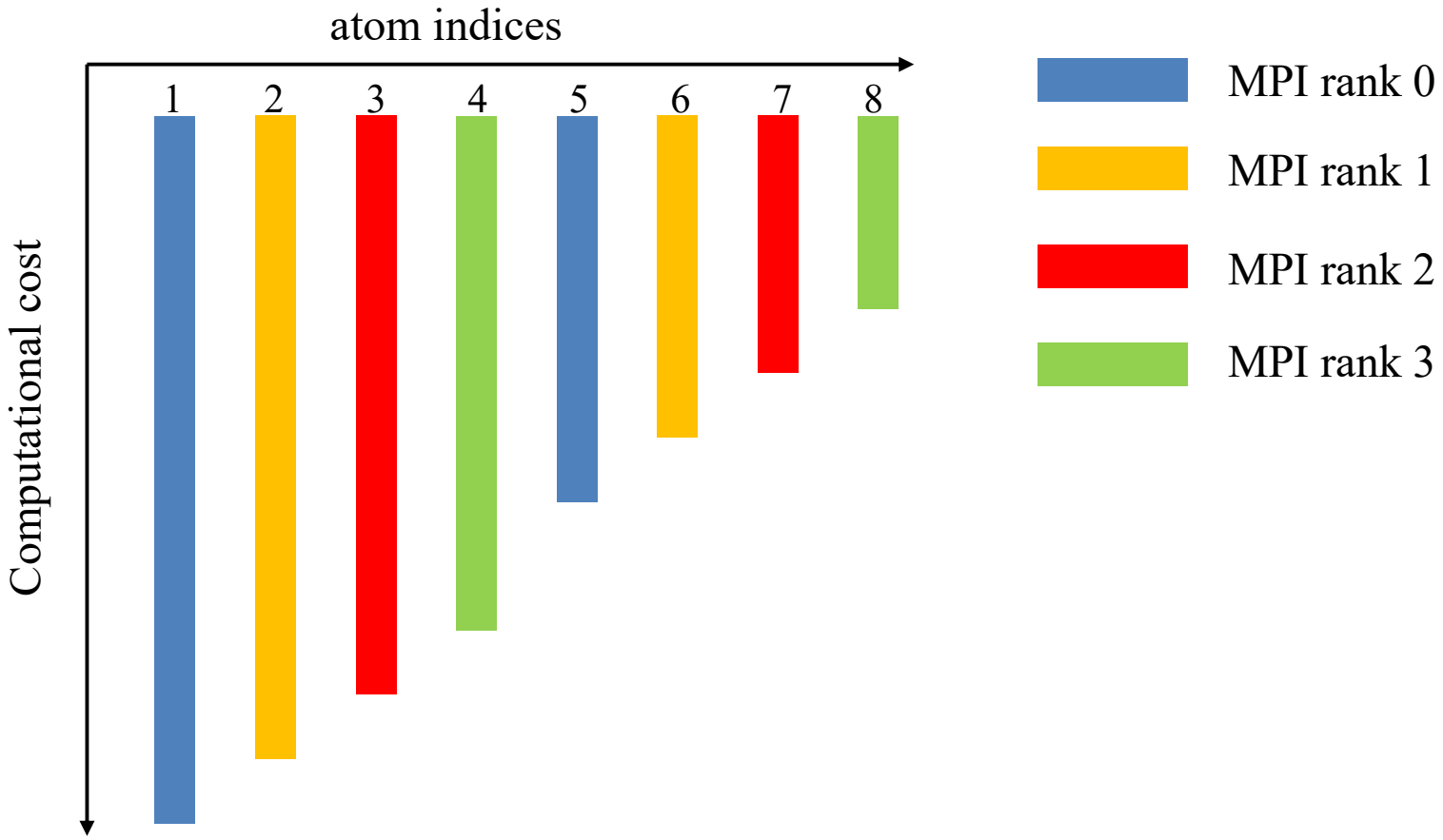
1. Each process has a copy of all particle data.
2. Each process works only part of the whole works by proper assign in do loops.

```
do i = 1, N
  do j = i+1, N

    energy(i,j)
    force(i,j)

  end do
end do
```



```
my_rank = MPI_Rank
proc = total MPI

do i = my_rank+1, N, proc
  do j = i+1, N

    energy(i,j)
    force(i,j)

  end do
end do

MPI reduction (energy,force)
```

Parallelization scheme 1: Replicated data approach



Perfect load balance is not guaranteed in this parallelization scheme

Hybrid (MPI+OpenMP) parallelization of the Replicated data approach

1. Works are distributed over MPI and OpenMP threads.
2. Parallelization is increased by reducing the number of MPIs involved in communications.

```
my_rank = MPI_Rank
proc = total MPI

do i = my_rank+1, N, proc
  do j = i+1, N

    energy(i,j)
    force(i,j)

  end do
end do

MPI reduction
(energy,force)
```

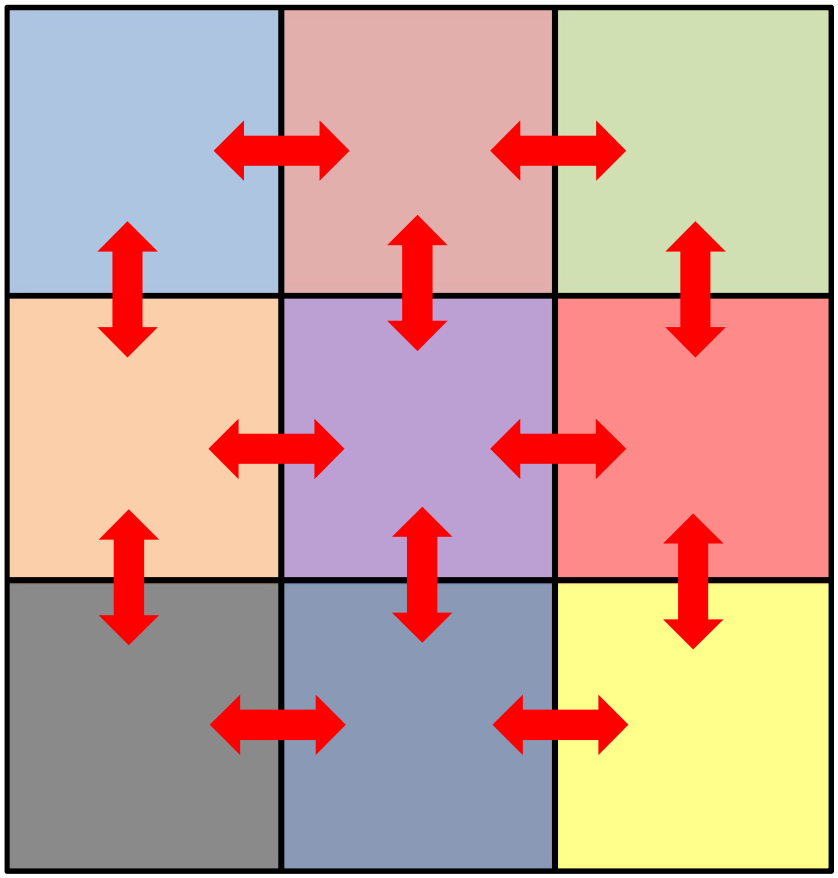
```
my_rank = MPI_Rank
proc = total MPI
nthread = total OMP thread
!$omp parallel
id = omp thread id
my_id = my_rank*nthread + id
do i = my_id+1,N,proc*nthread
  do j = i+1, N
    energy(i,j)
    force(i,j)
  end do
end do
Openmp reduciton
!$omp end parallel

MPI reduction (energy,force)
```

Pros and Cons of the Replicated data approach

1. **Pros** : easy to implement

2. **Cons**
   - Parallel efficiency is not good
     - No perfect load balance
     - Communication cost is not reduced by increasing the number of processes
     - We can parallelize only for energy calculation (with MPI, parallelization of integration is not so much efficient)
   - Needs a lot of memory
     - Usage of global data

Parallelization scheme 2: Domain decomposition



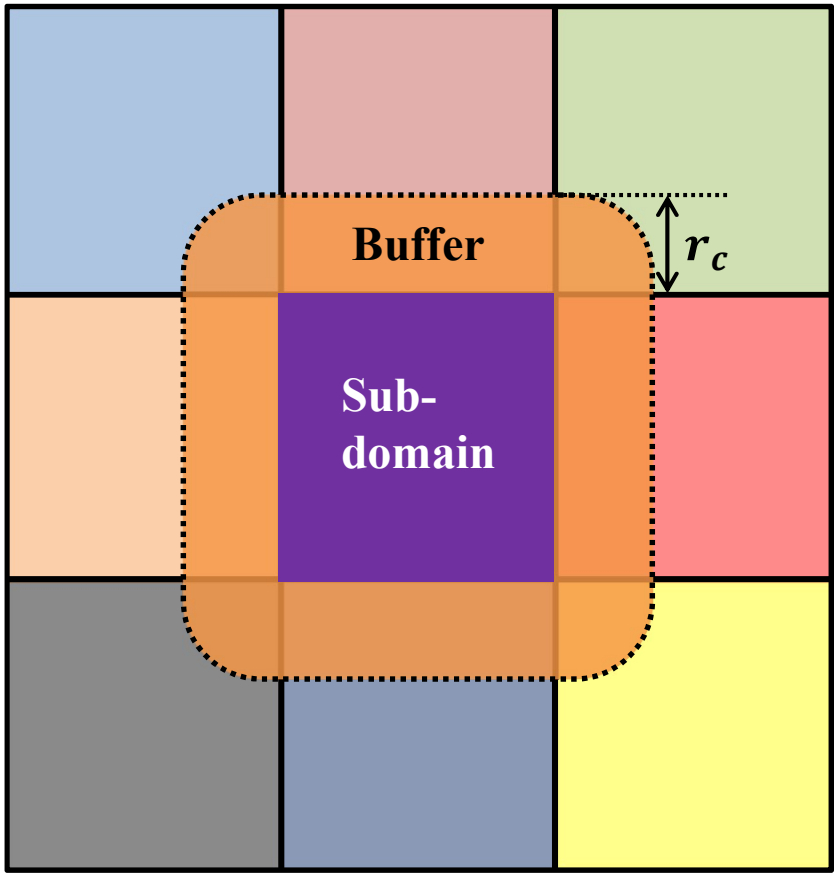
1. The simulation space is divided into subdomains according to MPI (different colors for different MPIs).

2. Each MPI only considers the corresponding subdomain.

3. MPI communications only among neighboring processes.

Communications between processors

Parallelization scheme 2: Domain decomposition



1. For the interaction between different subdomains, it is necessary to have the data of the buffer region of each subdomain.
2. The size of the buffer region is dependent on the cutoff values.
3. The interaction between particles in different subdomains should be considered very carefully.
4. The size of the subdomain and buffer region is decreased by increasing the number of processes.

Pros and Cons of the domain decomposition approach

1. Pros

- Good parallel efficiency
  - Reduced computational cost by increasing the number of processes
  - We can easily parallelize not only energy but also integration
- Availability of a huge system
  - Data size is decreased by increasing the number of processes

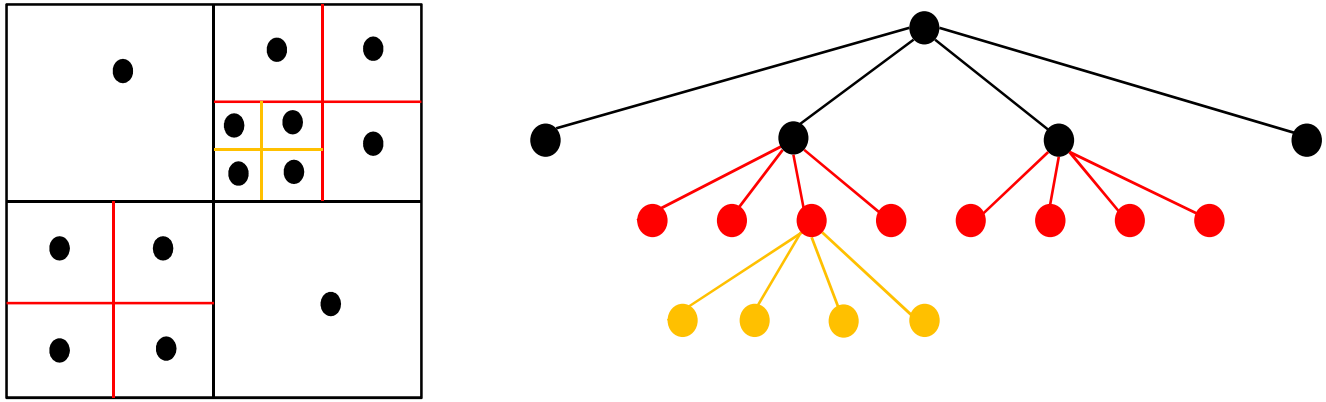
2. Cons

- Implementation is not easy
  - Domain decomposition scheme is highly depend on the potential energy type, cutoff and so on
- Good performance cannot be obtained for nonuniform particle distributions

Special treatment for nonuniform distributions

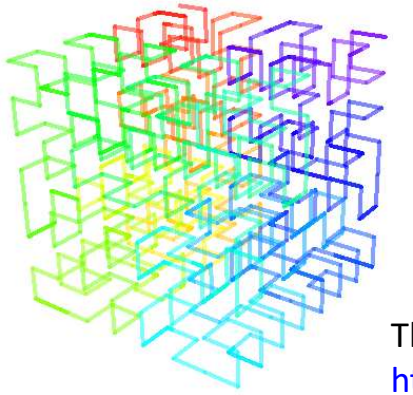
Tree method

: Subdomain size is adjusted to have the same number of particles



Hilbert space filling curve

: a map that relates multi-dimensional space to one-dimensional curve



The above figure is from Wikipedia
https://en.wikipedia.org/wiki/Hilbert_curve

Comparison of two parallelization scheme

	Computation	Communication	Memory
Replicated data	$O(N/P)$	$O(N)$	$O(N)$
Domain decomposition	$O(N/P)$	$O((N/P)^{2/3})$	$O(N/P)$

N: system size
P: number of processes

MPI Parallelization of MD (reciprocal space)

Smooth particle mesh Ewald method

$$E(\mathbf{r}) = \frac{1}{2}\sum_{\mathbf{n}'}\sum_{\substack{i,j=1 \\ |\mathbf{r}_i-\mathbf{r}_j+\mathbf{n}'|<r_c}}^{N} \frac{q_i q_j \operatorname{erfc}\left(\alpha|\mathbf{r}_i-\mathbf{r}_j+\mathbf{n}|\right)}{|\mathbf{r}_i-\mathbf{r}_j+\mathbf{n}|} + \frac{1}{2\pi V}\sum_{\mathbf{k}\neq\mathbf{0}} \frac{\exp(-\pi^2\mathbf{k}^2/\alpha^2)}{\mathbf{k}^2}|S(\mathbf{k})|^2 - \frac{\alpha}{\sqrt{\pi}}\sum_{i=1}^{N} q_i^2$$

Real part | Reciprocal part | Self energy

The structure factor in the reciprocal part is approximated as

$$S(k_1,k_2,k_3) = b_1(k_1)b_2(k_2)b_3(k_3)F(Q)(k_1,k_2,k_3)$$

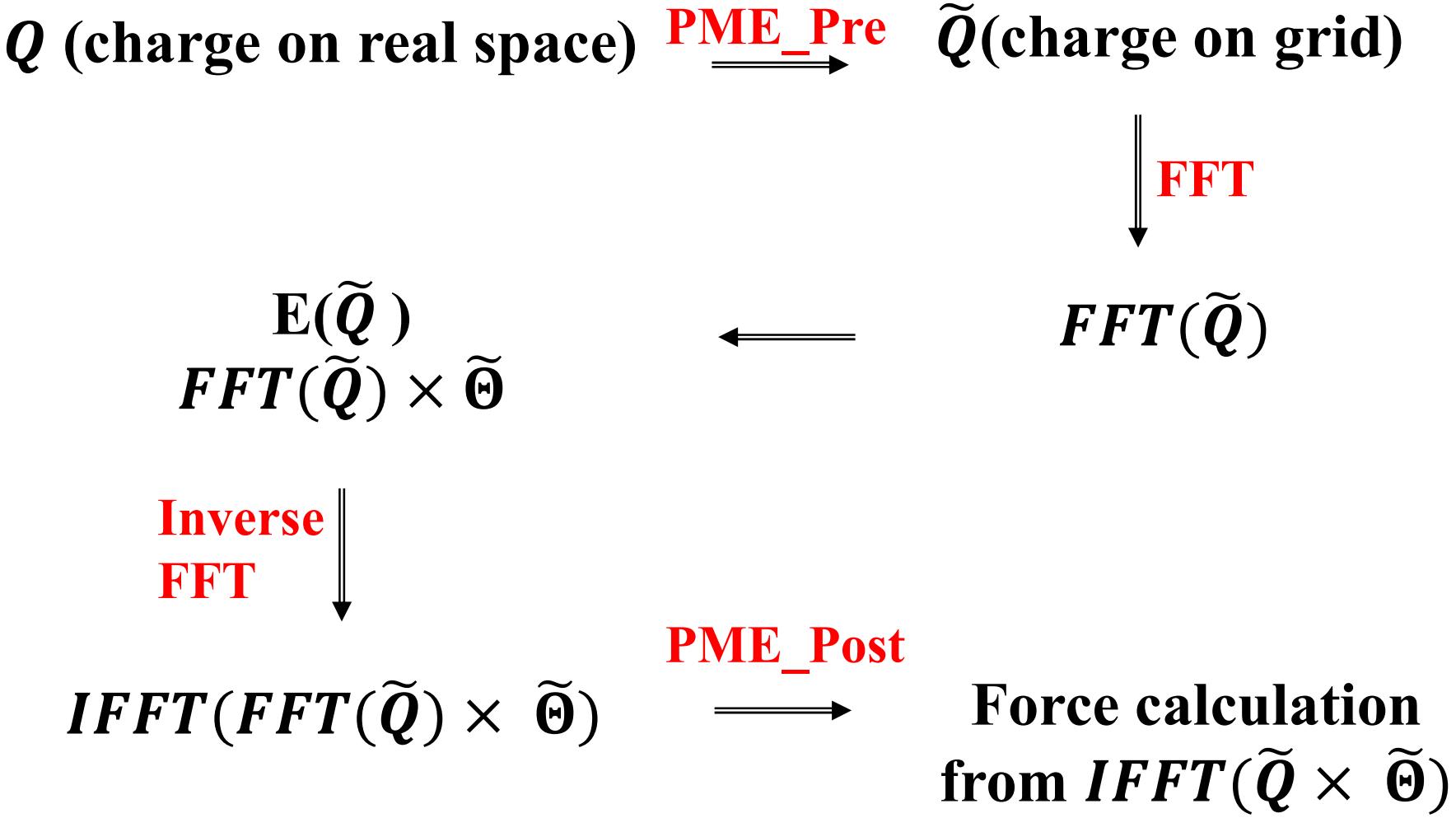
Using Cardinal B-splines of order n

Fourier Transform of charge

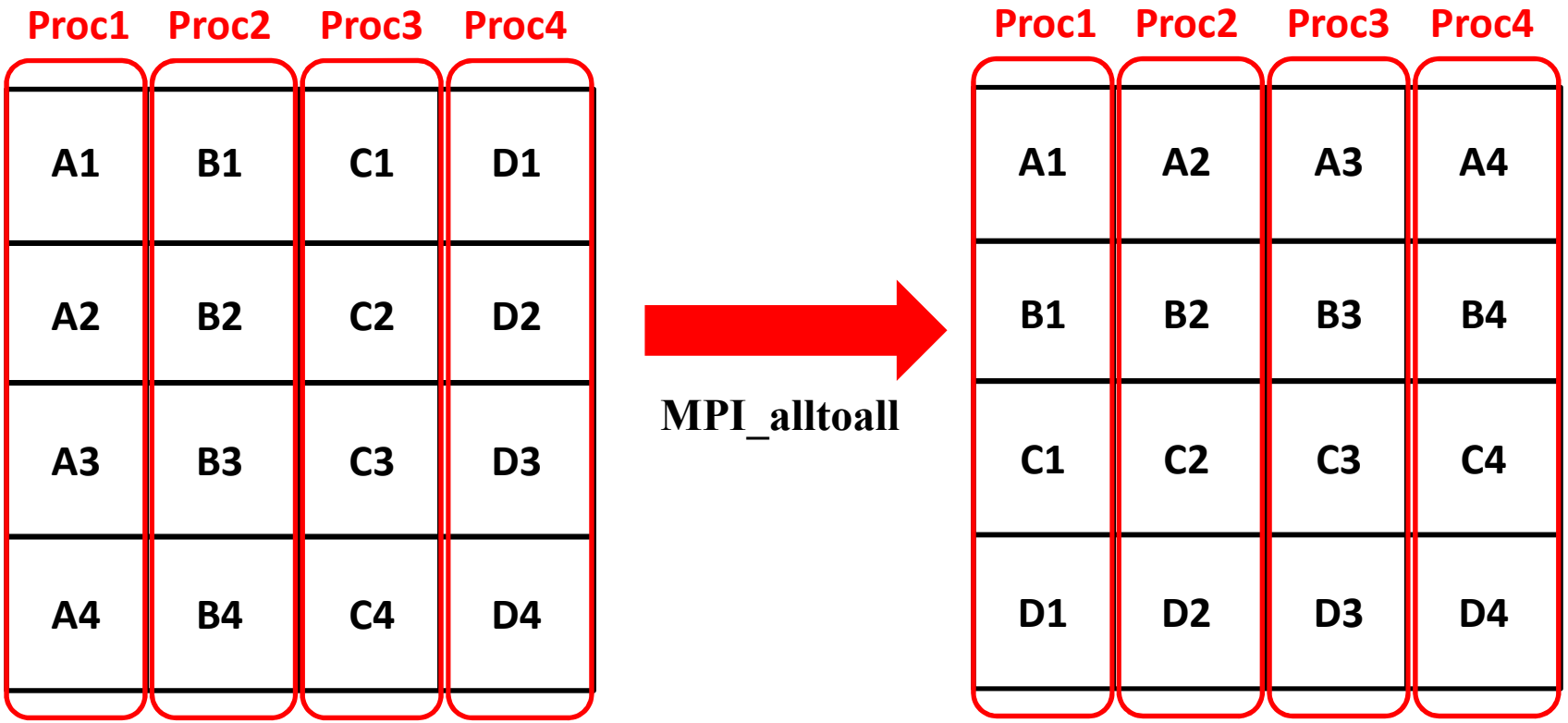
It is important to parallelize the Fast Fourier transform efficiently in PME!!

Ref : U. Essmann et al, J. Chem. Phys. 103, 8577 (1995)

Overall procedure of reciprocal space calculation

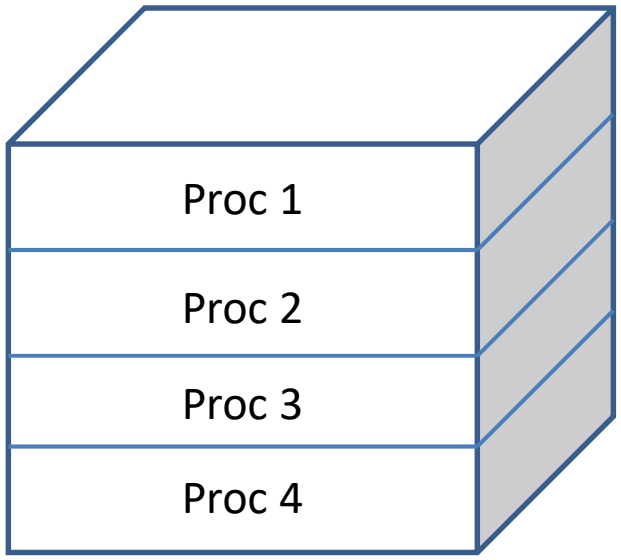


Simple note of MPI_alltoall communications



MPI_alltoall is same as matrix transpose!!

Parallel 3D FFT – slab (1D) decomposition

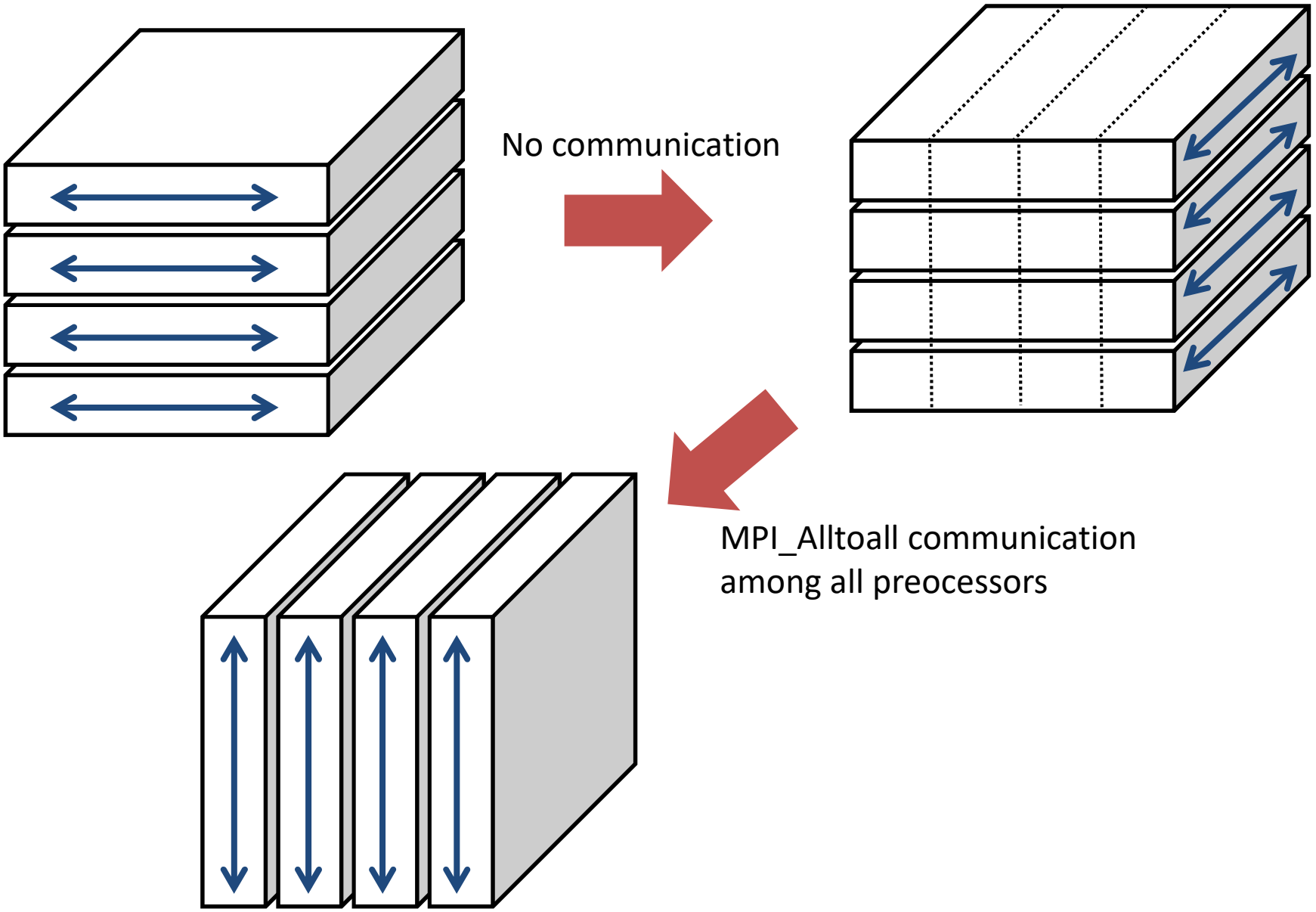


1. Each process is assigned a slob of size $N \times N \times N/P$ for computing FFT of $N \times N \times N$ grids on P processes.

2. The scalability is limited by N

3. N should be divisible by P

Parallel 3D FFT – slab (1D) decomposition



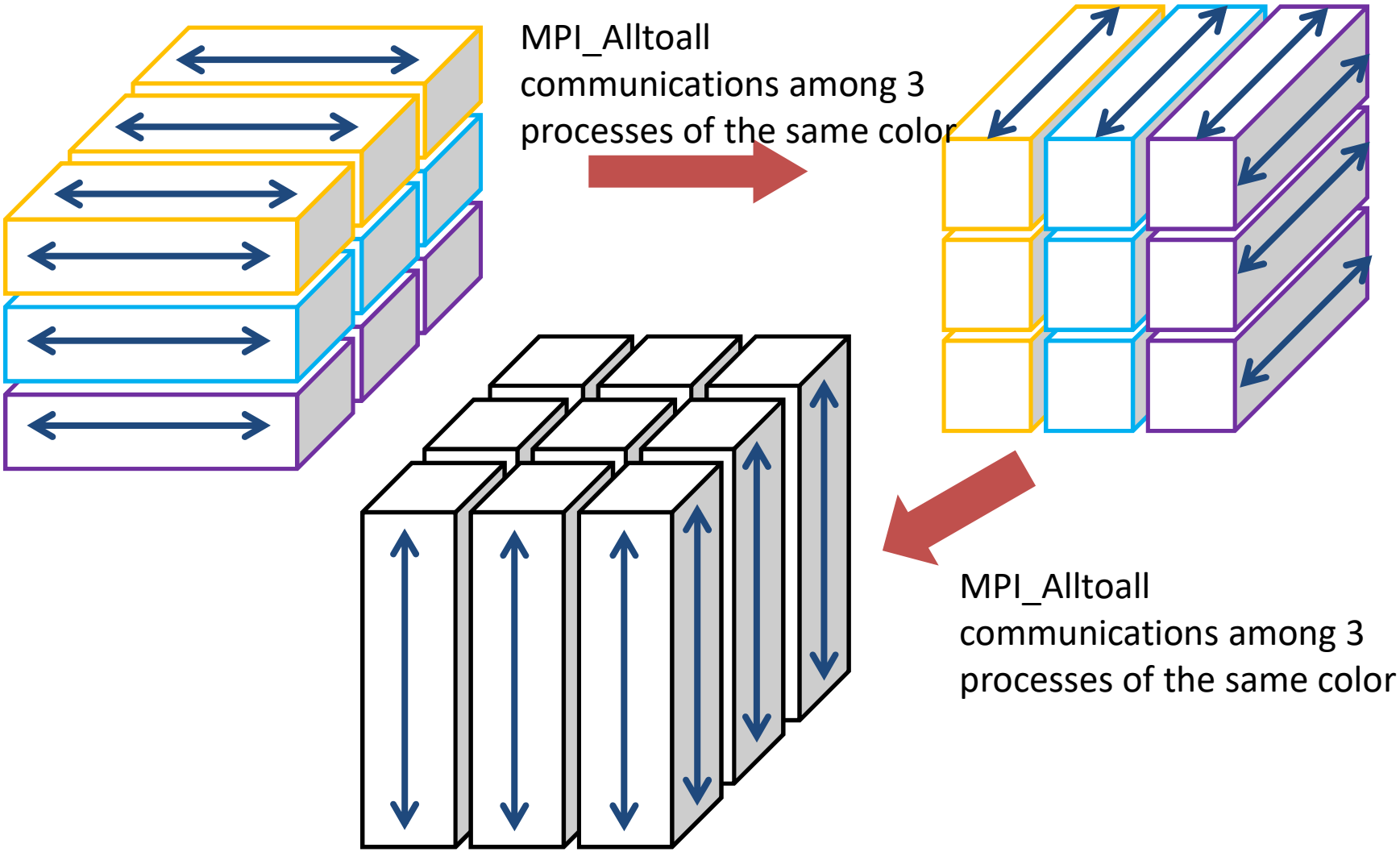
Parallel 3D FFT – slab (1D) decomposition (continued)

1. **Slab decomposition of 3D FFT has three steps**
    - 2D FFT (or two 1D FFT) along the two local dimension
    - Global transpose (communication)
    - 1D FFT along third dimension

2. **Pros**
    - fast using small number of processes

3. **Cons**
    - Limitation of the number of processes

Parallel 3D FFT –2D decomposition



Each process is assigned a data of size $N \times N/P \times N/Q$ for computing a FFT of $N \times N \times N$ grids on $P \times Q$ processes.

Parallel 3D FFT –2D decomposition (continued)

1. 2D decomposition of 3D FFT has five steps
    - 1D FFT along the local dimension
    - Global transpose
    - 1D FFT along the second dimension
    - Global transpose
    - 1D FFT along the third dimension

2. The global transpose requires communication only between subgroups of all nodes
3. Cons: Slower than 1D decomposition for a small number of processes
4. Pros : Maximum parallelization is increased

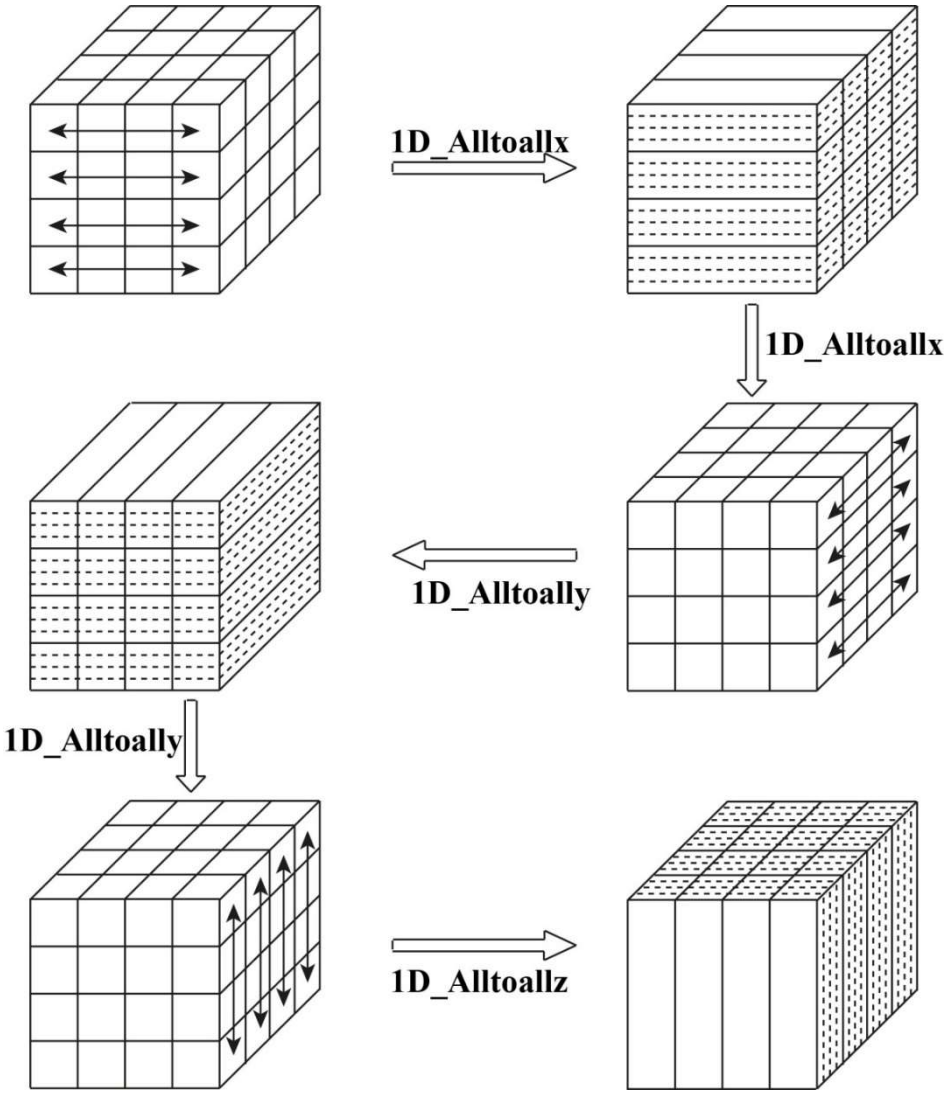
Pseudo code of reciprocal space calculation with 2D decomposition of 3D FFT

```
! compute Q factor
do i = 1, natom/P
  compute Q_orig
end do
call mpi_alltoall(Q_orig, Q_new, …)
accumulate Q from Q_new

!FFT : F(Q)
do iz = 1, zgrid(local)
  do iy = 1, ygrid(local)
     work_local = Q(my_rank)
     call fft(work_local)
     Q(my_rank) = work_local
   end do
end do
call mpi_alltoall(Q, Q_new,…)
do iz = 1, zgrid(local)
  do ix = 1, xgrid(local)
     work_local = Q_new(my_rank)
     call fft(work_local)
     Q(my_rank) = work_local
   end do
end do
call mpi_alltoall(Q,Q_new,..)
```

```
do iy = 1, ygrid(local)
  do ix = 1, xgrid(local)
     work_local = Q(my_rank)
     call fft(work_local)
     Q(my_rank) = work_local
   end do
end do

! compute energy and virial
do iz = 1, zgrid
  do iy = 1, ygrid(local)
     do ix = 1, xgrid(local)
        energy = energy +  sum(Th*Q)
        virial = viral + ..
     end do
   end do
end do

! X=F_1(Th)*F_1(Q)

! FFT (F(X))
```

```
do iy = 1, ygrid(local)
  do ix = 1, xgrid(local)
     work_local = Q(my_rank)
     call fft(work_local)
     Q(my_rank) = work_local
   end do
end do
call mpi_alltoall(Q,Q_new,..)
do iz = 1, zgrid(local)
  do ix = 1, xgrid(local)
     work_local = Q(my_rank)
     call fft(work_local)
     Q(my_rank) = work_local
   end do
end do
call mpi_alltoall(Q,Q_new)
do iz = 1, zgrid(local)
  do iy = 1, ygrid(local)
     work_local = Q(my_rank)
     call fft(work_local)
     Q(my_rank) = work_local
   end do
end do

!compute force
```

Parallel 3D FFT – 3D decomposition

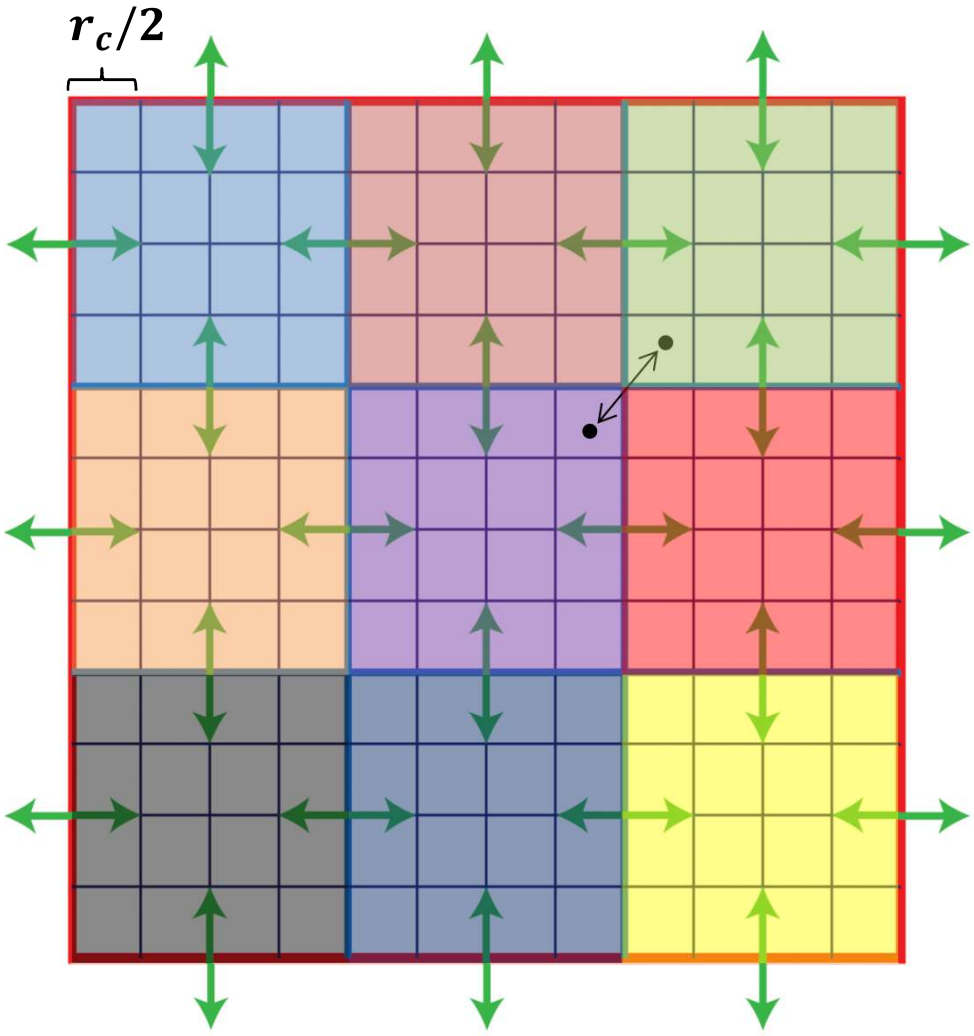


- More communications than existing FFT
- MPI_Alltoall communications only in one dimensional space
- Reduce communicational cost for large number of processes

Ref) J. Jung et al. Comput. Phys. Comm. 200, 57-65 (2016)

Case Study:
Parallelization in GENESIS MD software

Domain decomposition in GENESIS SPDYN

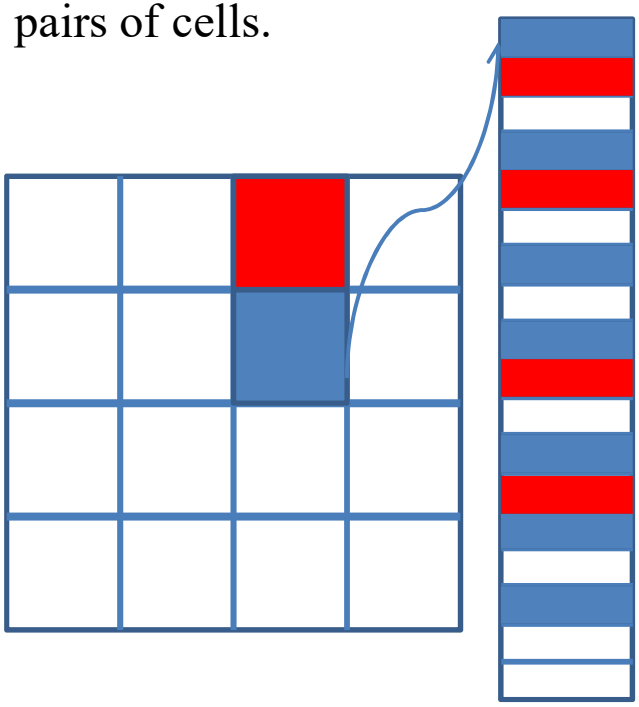


1. The simulation space is divided into subdomain according to the number of MPI processes.
2. Each subdomain is further divided into the unit domain (named cell).
3. Particle data are grouped together in each cell.
4. Communications are considered only between neighboring processes.
5. In each subdomain, OpenMP parallelization is used.

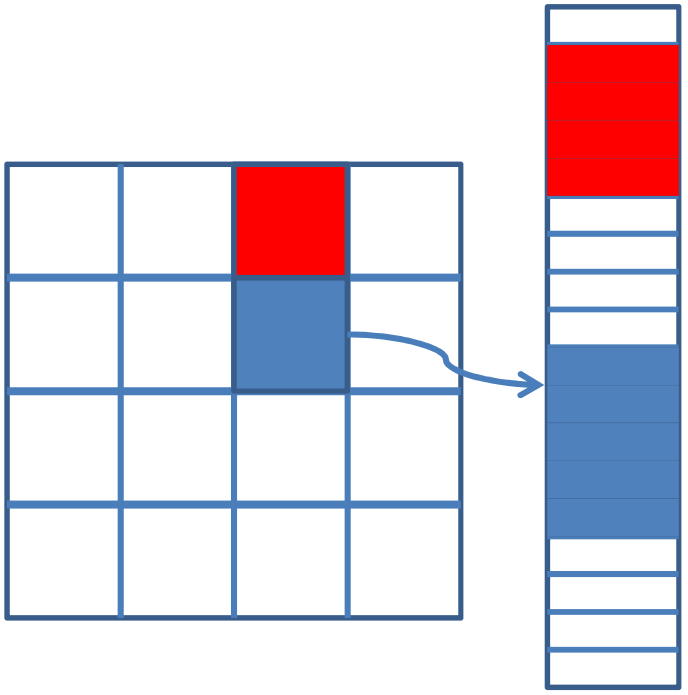
Efficient shared memory calculation
=>Cell-wise particle data

1. Each cell contains an array with the data of the particles that reside within it

2. This improves the locality of the particle data for operations on individual cells or pairs of cells.



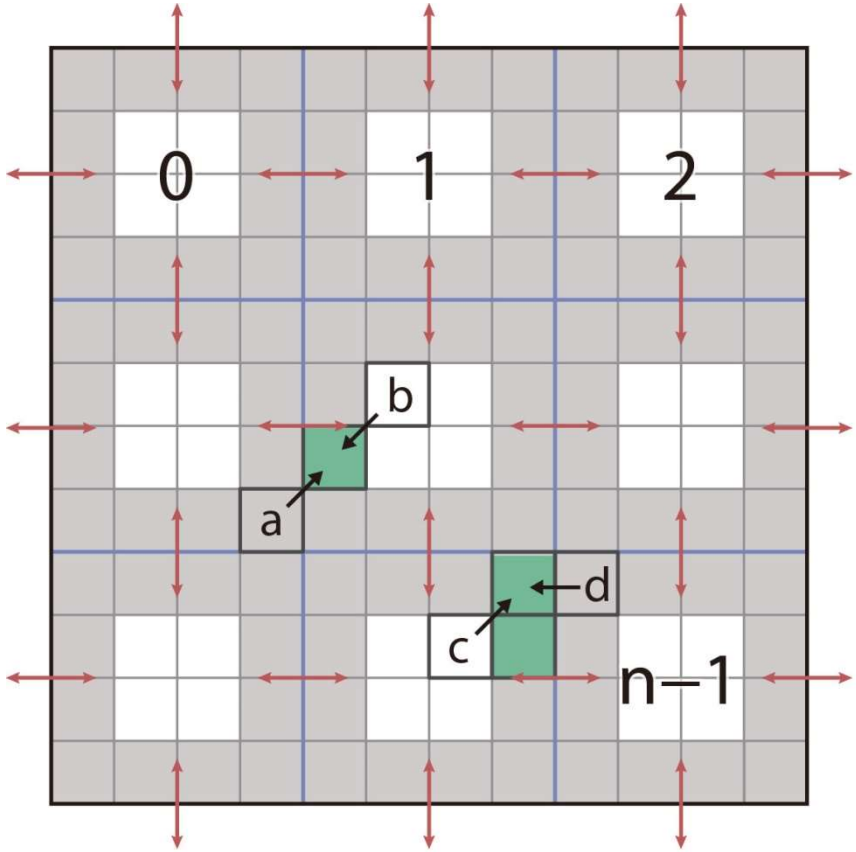
particle data in traditional cell lists

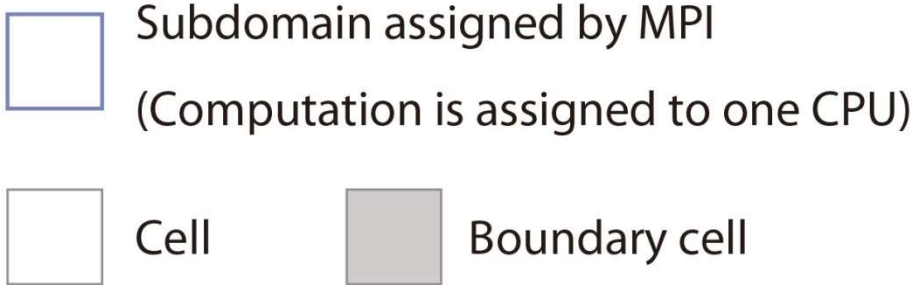


cell-wise arrays of particle data

Ref : P. Gonnet, JCC 33, 76-81 (2012)

Midpoint cell method



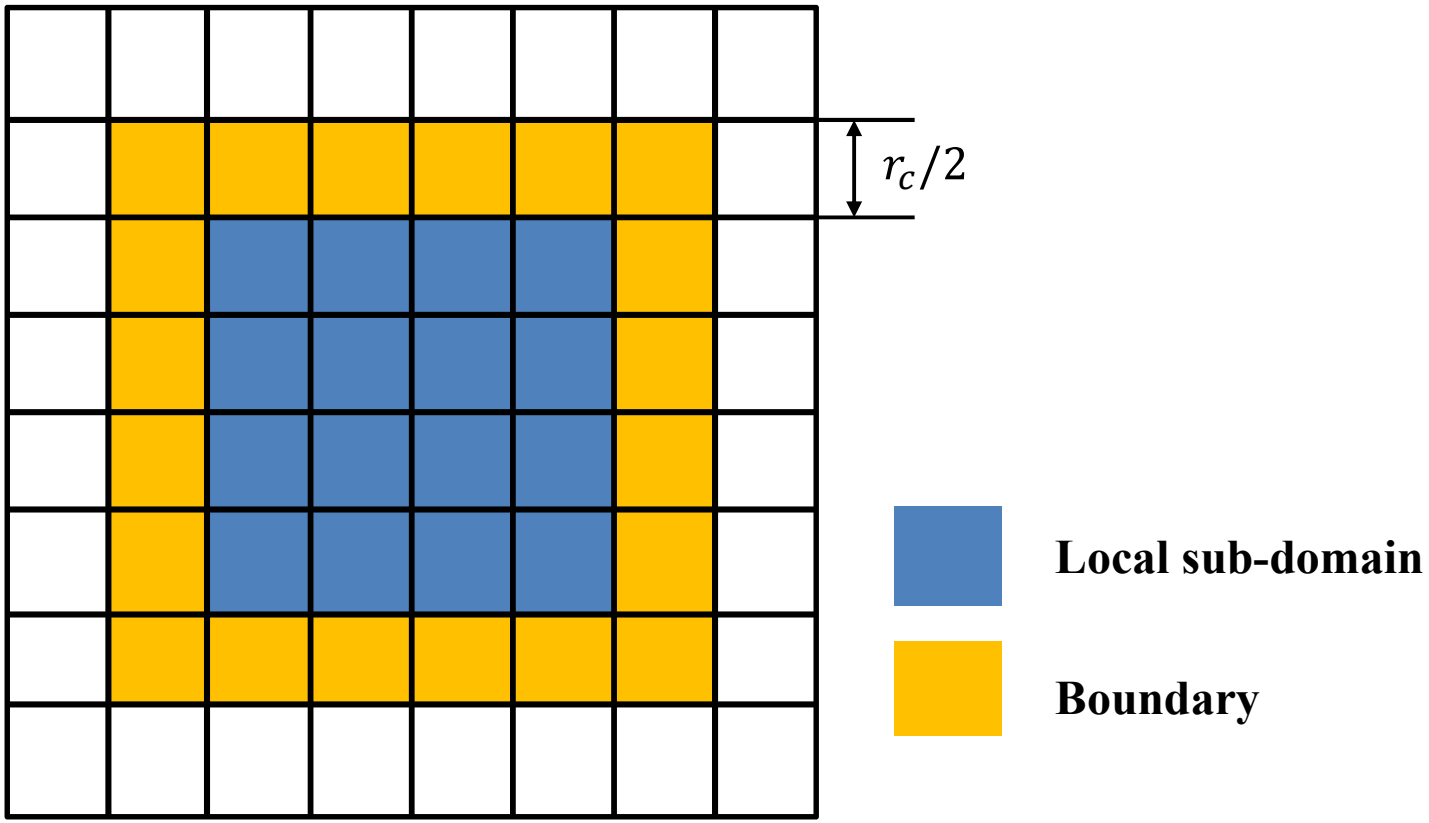


- For every cell pair, midpoint cell is decided.
- If midpoint cell is not uniquely decided, we decide it by considering load balance.
- For every particle pairs, interaction subdomain is decided from the midpoint cell.
- With this scheme, we can only communicate data of each subdomain only considering the adjacent cells of each subdomain.

Subdomain structure with midpoint cell method

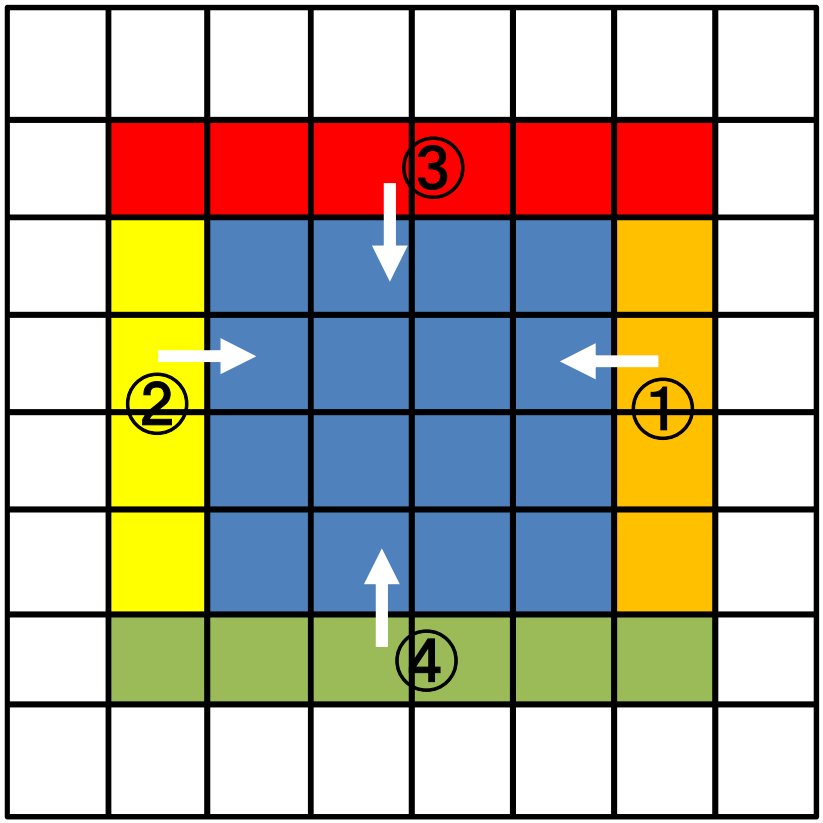
1. Each MPI has local subdomain which is surrounded by boundary cells.

2. At least, the local subdomain has at least two cells in each direction



Communication pattern (example of coordinates)

1. Each subdomain can evaluate interactions by obtaining the data of boundary region from other processes
2. Using the communication pattern shown below, we can minimize the frequency of communication.



① Receive from the upper process in x dimension

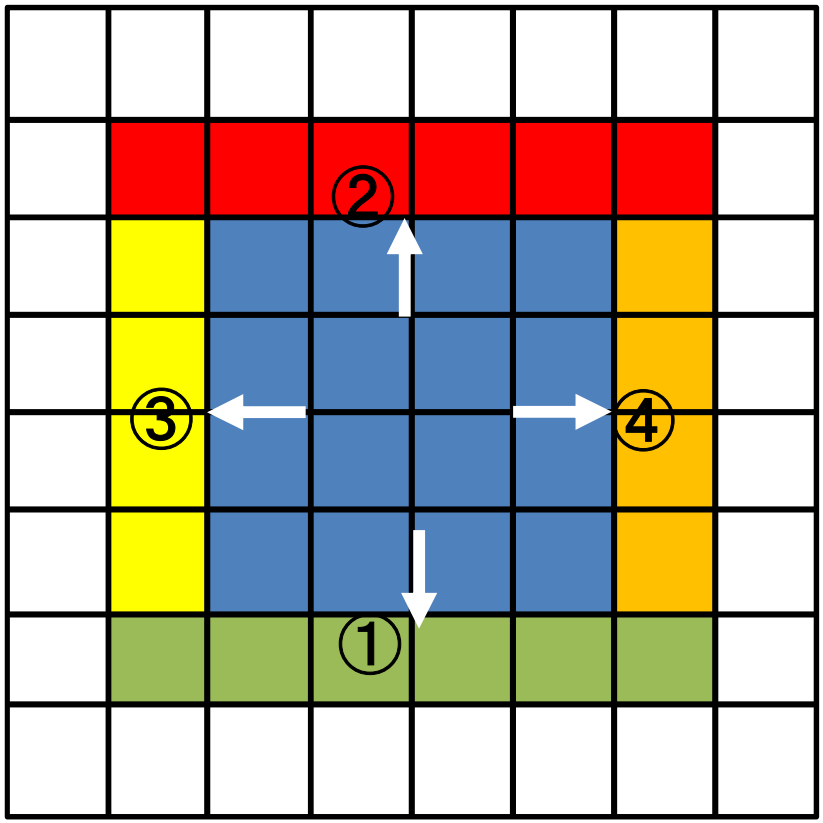
② Receive from the lower process in x dimension

③ Receive from the upper process in y dimension

④ Receive from the lower process in y dimension

Communication pattern (example of forces)

1. Communication pattern of forces is opposite to the communication pattern of coordinates.
2. Each process send the force data in boundary cells.



① Send to the lower process in y dimension

② Send to the upper process in y dimension

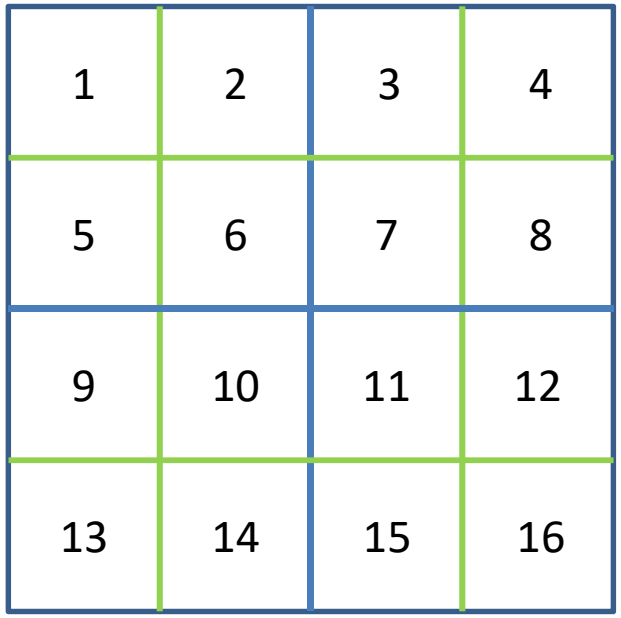
③ Send to the lower process in x dimension

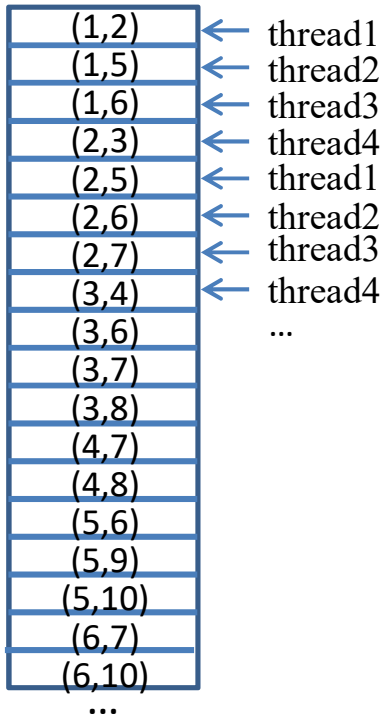
④ Send to the upper process in x dimension

OpenMP parallelization in GENESIS

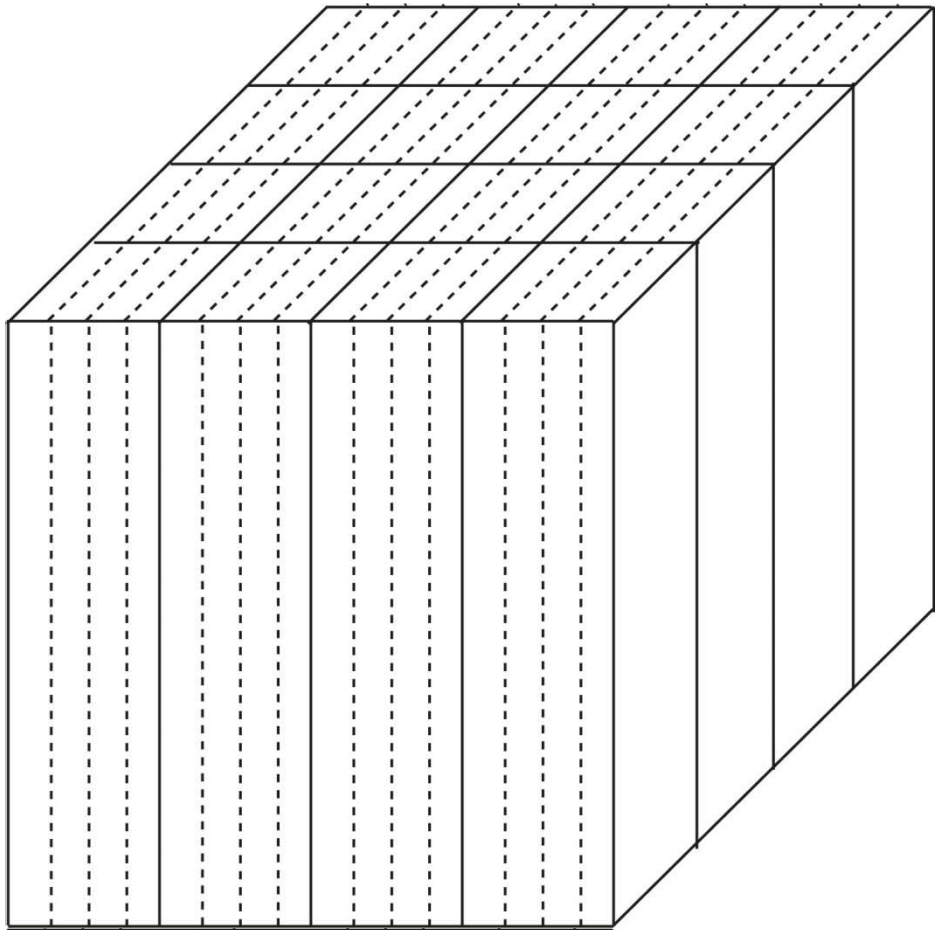
1. In the case of integrator, every cell indices are divided according to the thread id.

2. As for the non-bond interaction, cell-pair lists are first identified and cell-pair lists are distributed to each thread

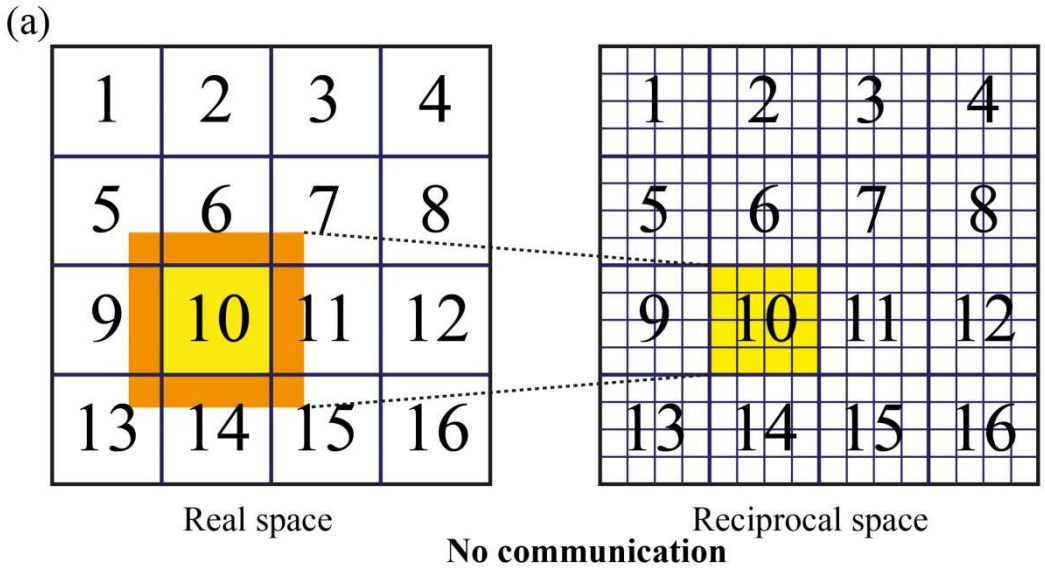




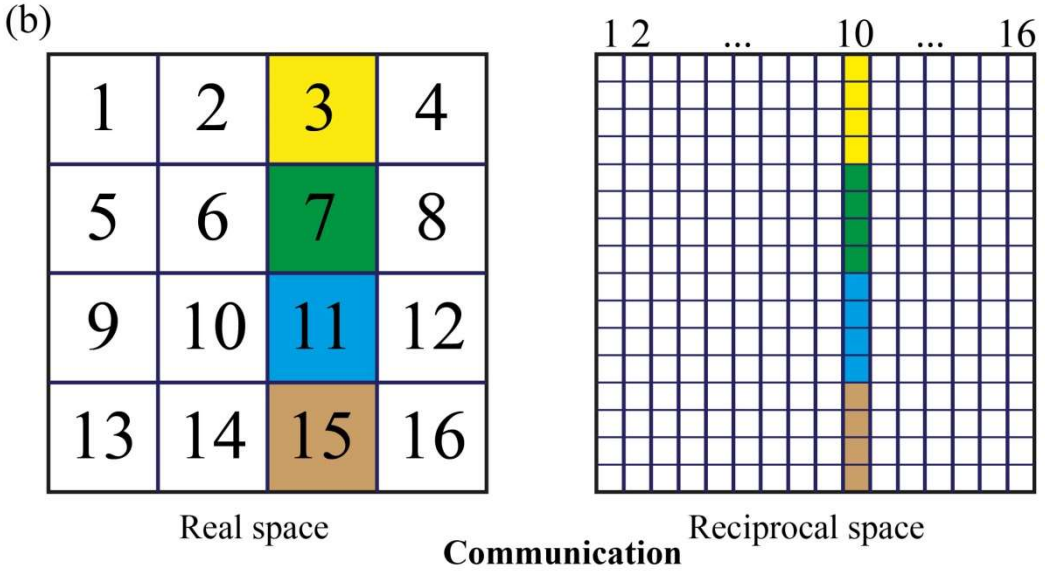
Parallelization of FFT in GENESIS



FFT in GENESIS (2 dimensional view)

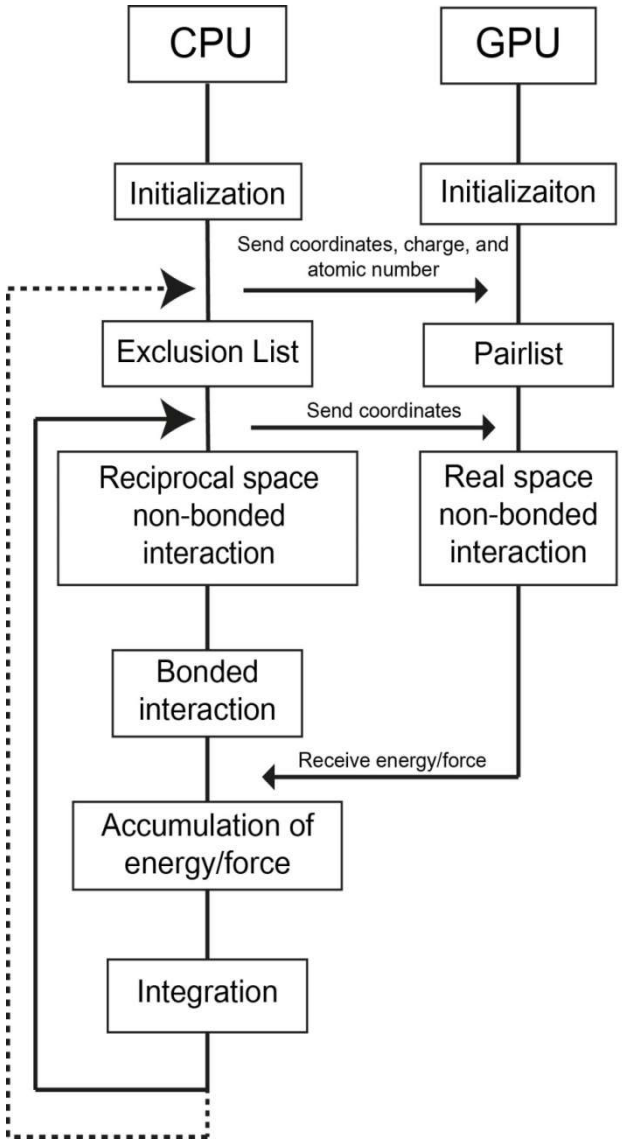


GENESIS
(Identical domain decomposition between two space)



NAMD, Gromacs
(Different domain decomposition)

Parallelization based on CPU/GPU calculation

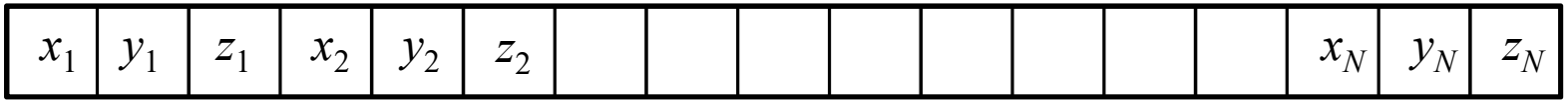


Computationally expensive part: GPU

Communicationally expensive part: CPU

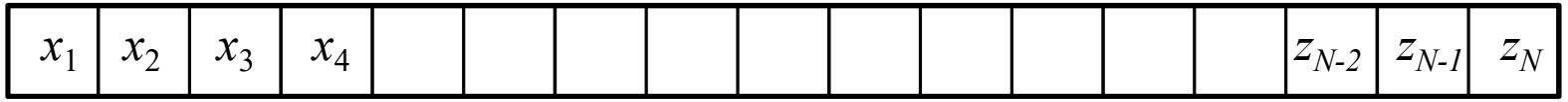
SIMD in GENESIS (developer version)

- Array of Structure (AoS)



In GENESIS, it is expressed as `coord_pbc(1:3,1:natom,1:ncell)`

- Structure of Array (SoA)

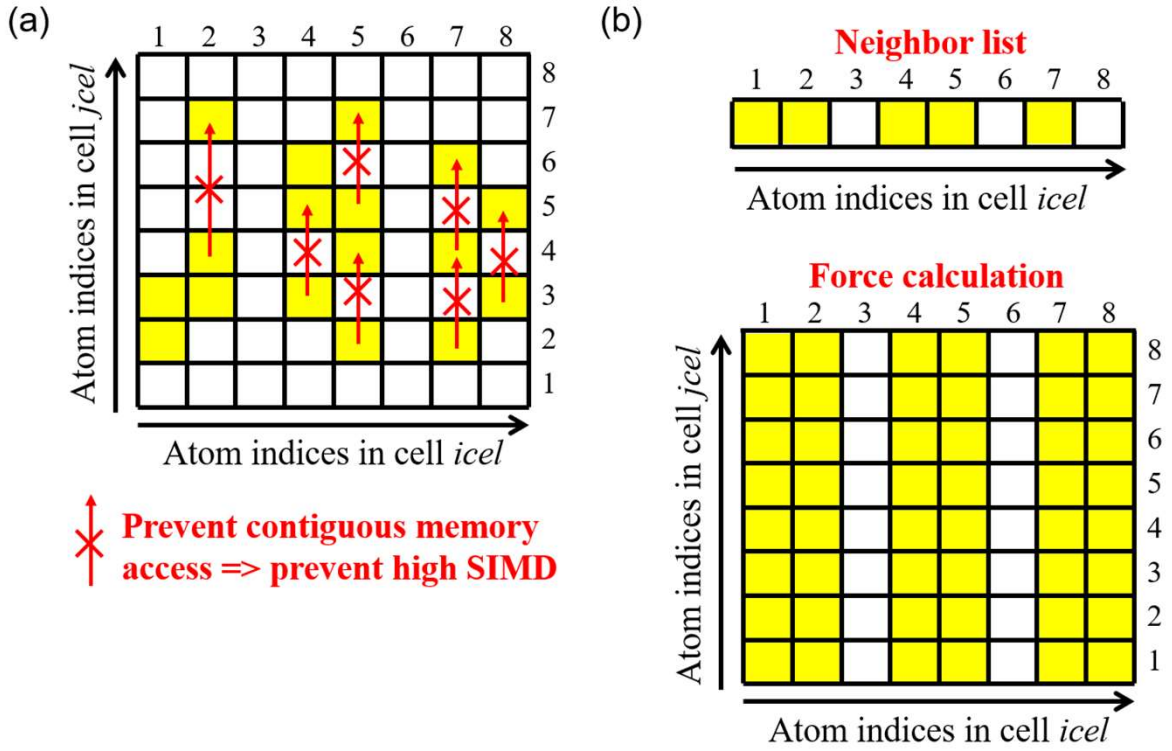


In updated GENESIS source code for KNL, it will be expressed as
`coord_pbc(1:natom,1:3,1:ncell)`

- For Haswell/Broadwell/KNL machines, SoA shows better performance than AoS due to efficient vectorization (SIMD)

GENESIS developments on KNL

Simple pair-list scheme with better performance



- Pair-list scheme in GENESIS 1.X does not show good performance due to non-contiguous memory access (Figure (a))

GENESIS developments on KNL

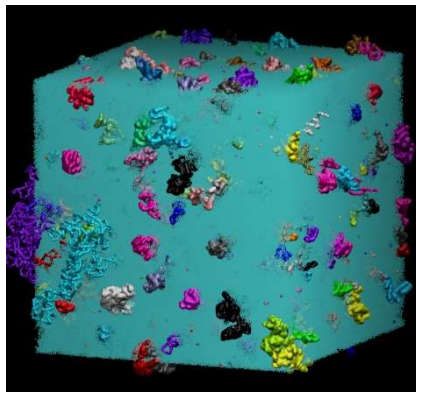
	Memory usage	Number of interactions	CPU time (pair-list)	CPU time (non-bonded)	CPU time (total)
Algorithm 1 (AoS)	305.88 MiB	47364770	9.95	36.42	46.37
Algorithm 1 (SoA)	305.88 MiB	47364770	6.60	33.71	40.31
Algorithm 2 (AoS)	48.4 MiB	122404436	4.91	102.46	107.37
Algorithm 2 (SoA)	48.4 MiB	122404436	4.15	27.86	32.01

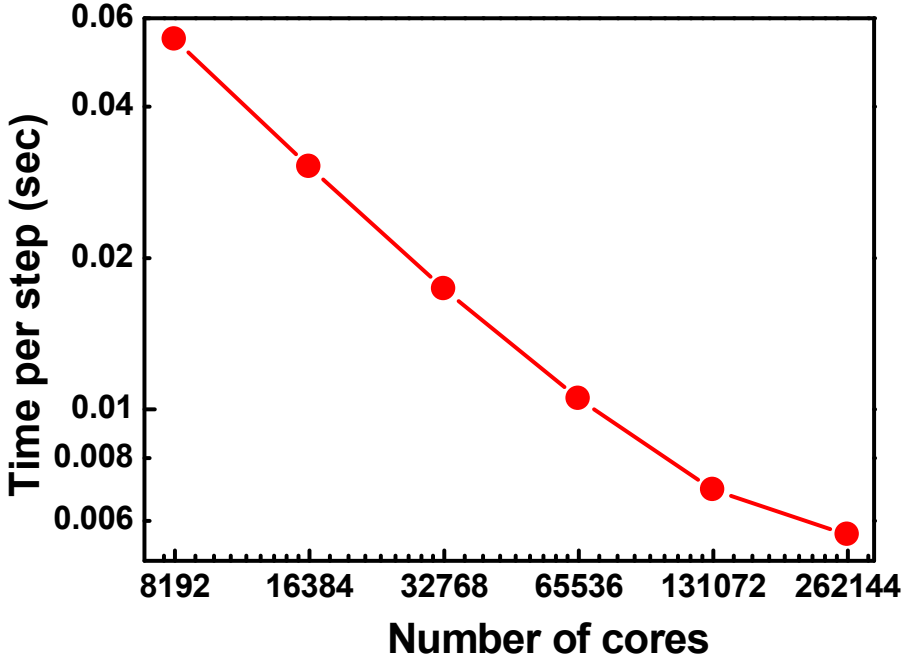
- Algorithm 1: pair-list algorithm used in GENESIS 1.X
- Algorithm 2: New pair-list scheme
- Target system : ApoA1 (about 90,000 atoms, 2000 time steps)

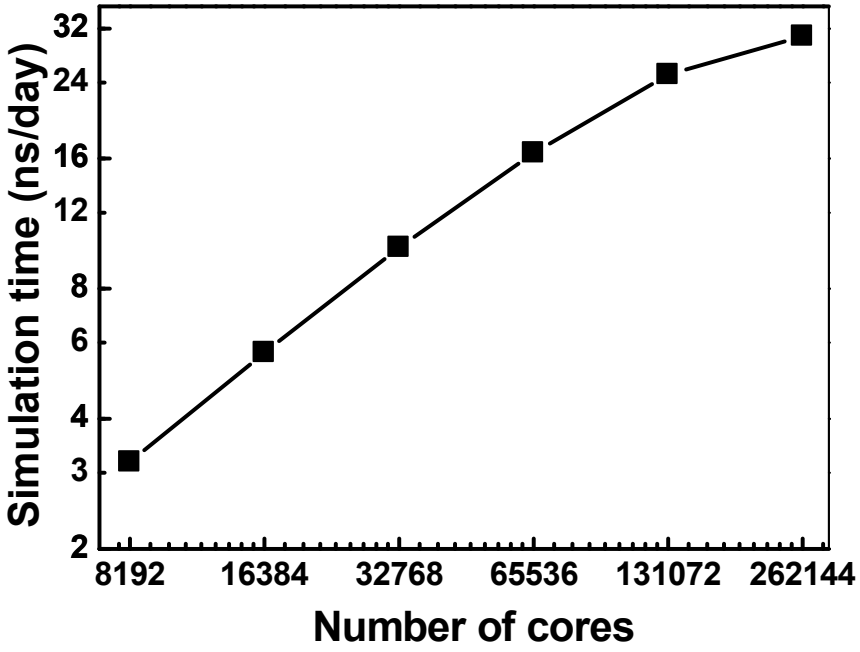
Benchmark performance of GENESIS

GENESIS 1.0 performance on K (1)

- atom size: 11737298
- macro molecule size: 216 (43 type)
- metabolites size: 4212 (76 type)
- ion size: 23049 (Na^+, Cl^-, Mg^{2+}, K^+)
- water size: 2944143
- PBC size: 480 x 480 x 480($Å^3$)

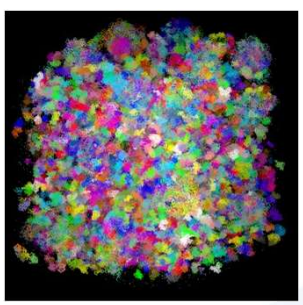


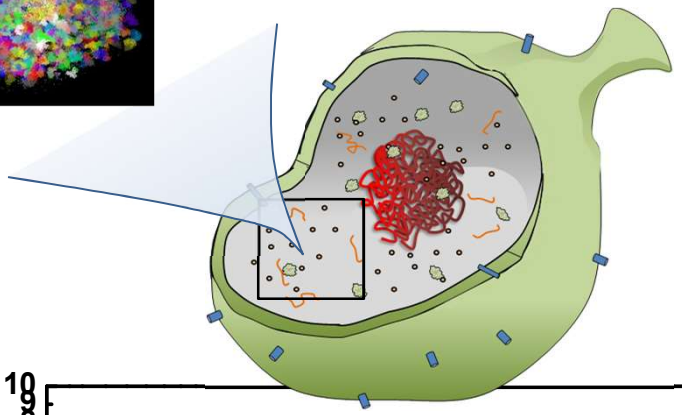


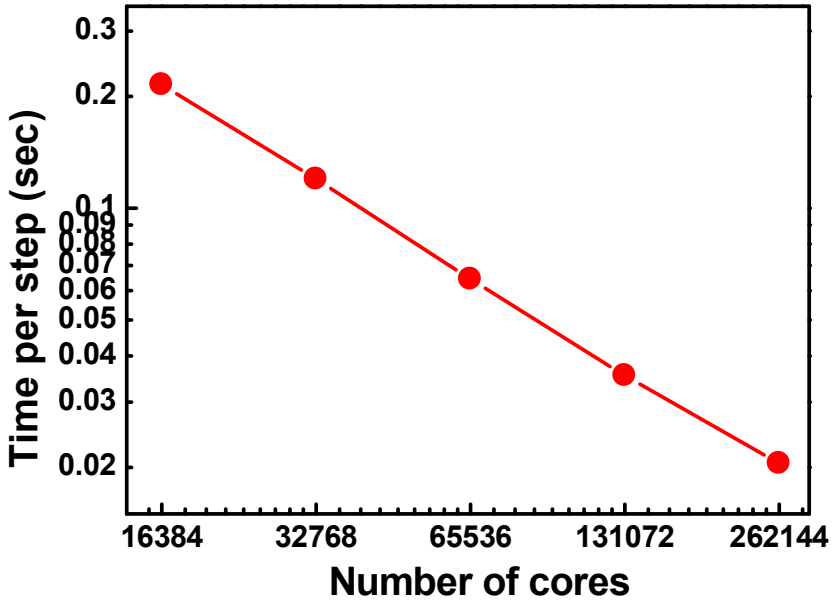


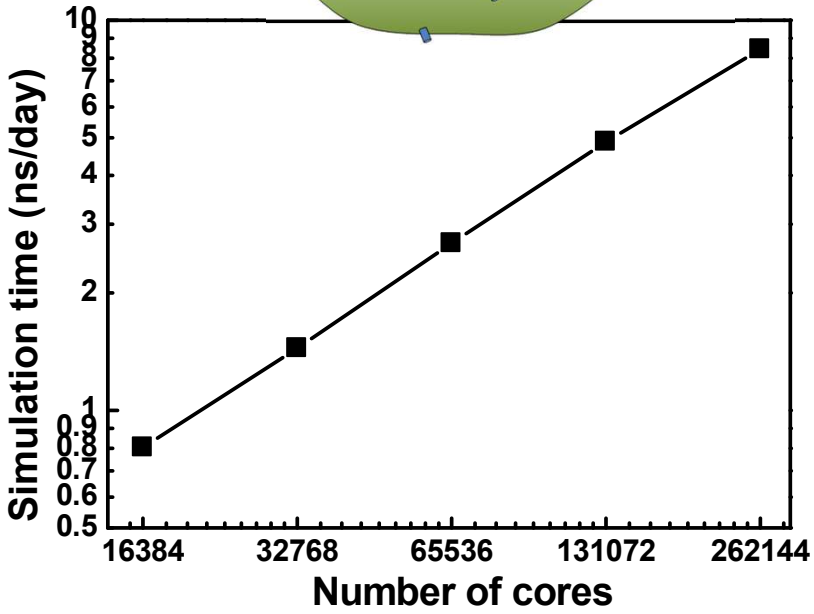
GENESIS 1.0 performance on K (2)

100nm = 1/10000mm

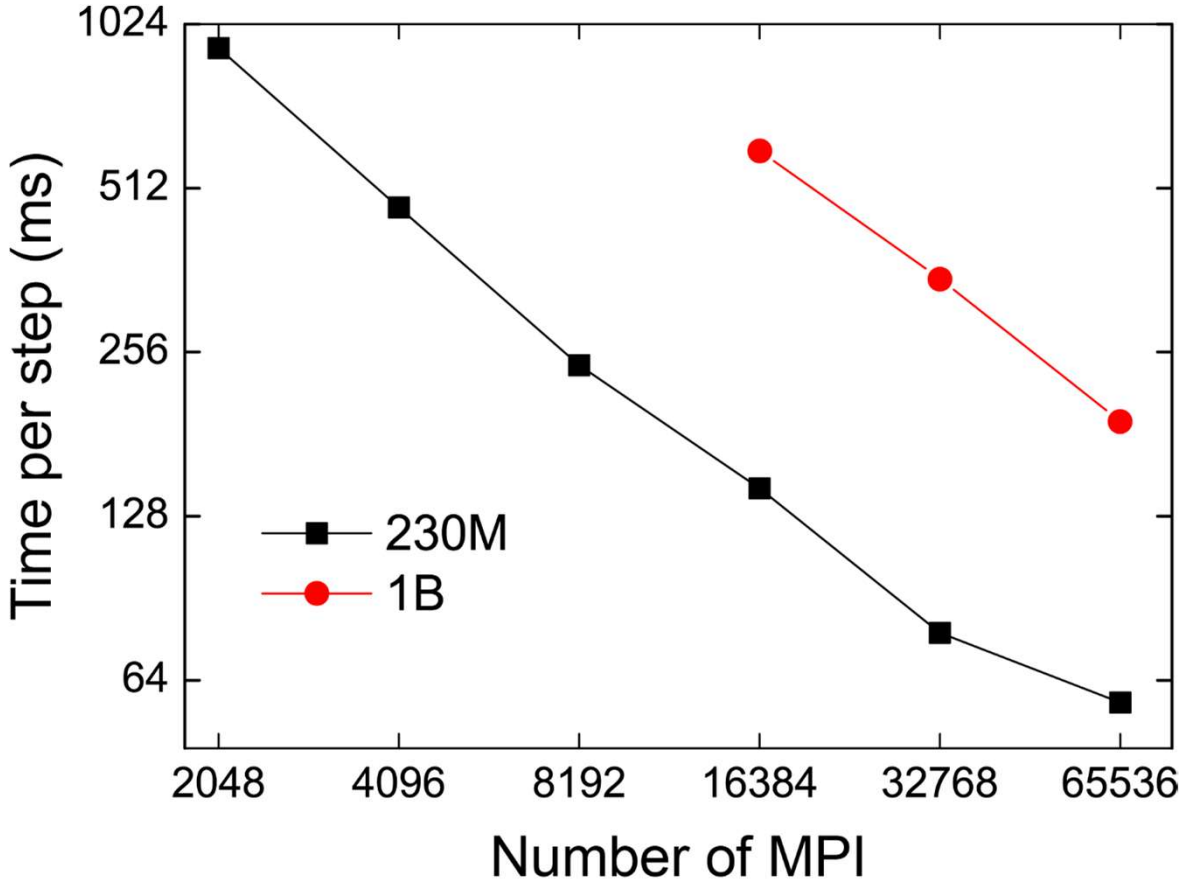




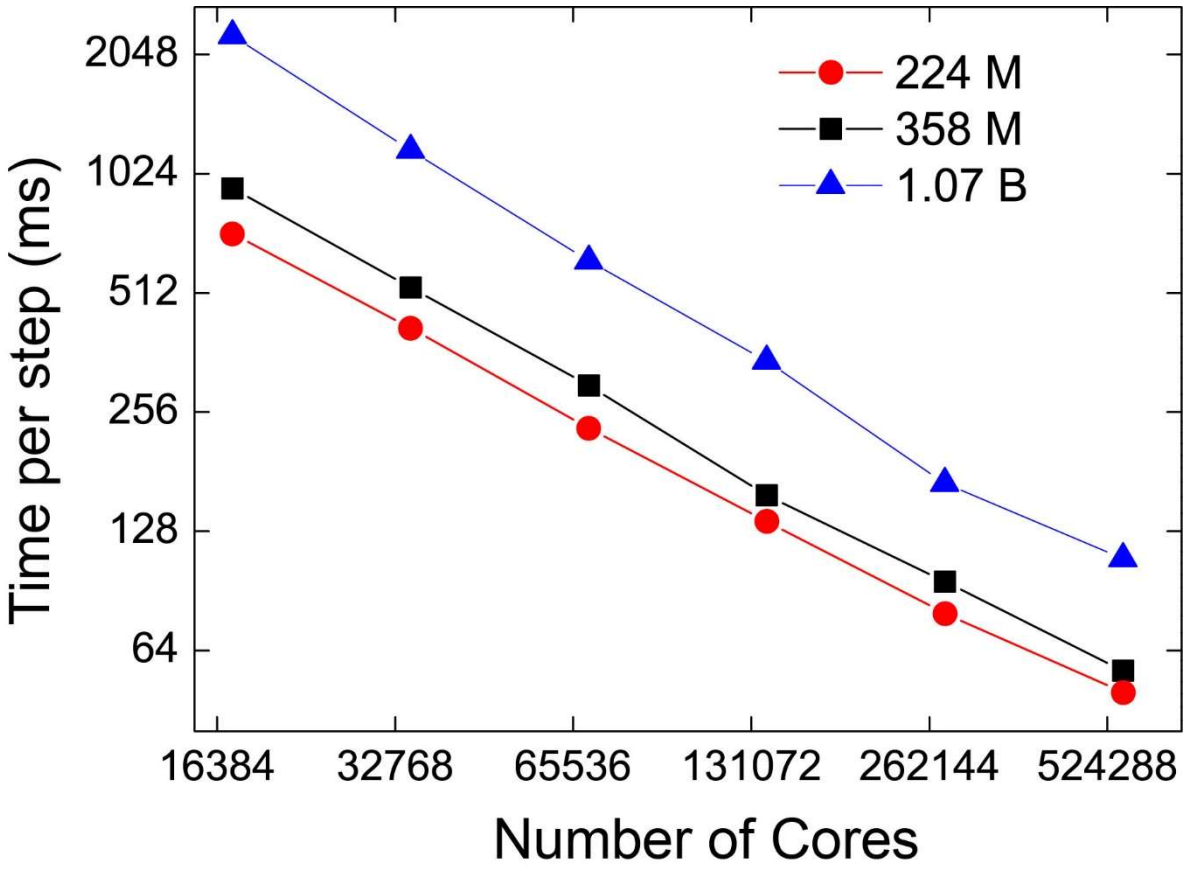




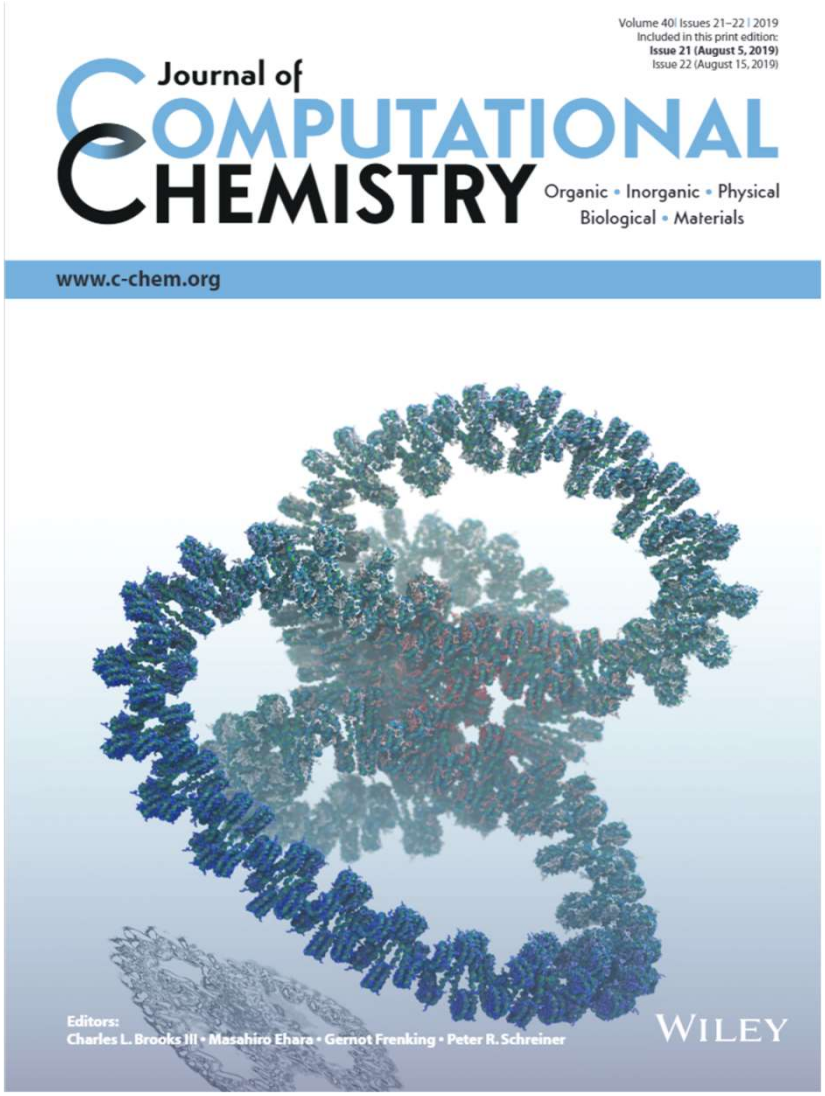
GENESIS performance on KNL (Oak-foreast)



GENESIS performance on KNL (LANL Trinity)



Efficient parallelization enable us to perform the MD simulations of the world largest system



Popular Mechanics

This Is the First-Ever Simulation of an Entire Gene

Popular Mechanics, 29 Apr 2019

Los Alamos National Laboratory A group of scientists have created the world's first computer simulation of an entire gene.

Primeur Weekly! Magazine

Scientists create first billion-atom biomolecular simulation ...

Primeur Magazine, 29 Apr 2019

A Los Alamos-led team created the largest simulation to date of an entire gene of DNA, a feat that required one billion atoms...

TECH TIMES

Researchers Create Model That Simulates Each And Every Atom In A Single Human Gene

Tech Times, 25 Apr 2019

Scientists at Los Alamos National Laboratory utilized the Trinity Supercomputer to create the first billion-atom biomolecular...

Forbes

What Does DNA Look Like? After 66 Years, We're Still Learning More.

Forbes, 25 Apr 2019

, Francis Crick and James Watson published their famous article that showed that the shape of DNA is a double helix.

Popular Mechanics

Here's the First-Ever Simulation of an Entire Gene

Popular Mechanics, 24 Apr 2019

Los Alamos National Laboratory A group of scientists have created the world's first computer simulation of an entire gene.

This Is the First-Ever Simulation of an Entire Gene

Summary

- Parallelization: MPI and OpenMP
  - MPI: Distributed memory parallelization
  - OpenMP: Shared memory parallelization
- Parallelization of MD: mainly by domain decomposition with FFT parallelization
- Key issues in parallelization: minimizing communication cost to maximize the parallel efficiency