21st Summer School of PARALLEL COMPUTING

July 2 - 13, 2012 (Italian)
September 3 - 14, 2012 (English)

Parallel Molecular Dynamics

CINECA

Agenda

- *Introduction to Classical Molecular Dynamics*
- *Parallelisation of Molecular Dynamics*
  - *Atom and Force Decomposition*
  - *Domain Decomposition*
  - *Parallelisation of the electrostatics calculation*
- *Why do Molecular Dynamics stop scaling*
  - *Effects of system size*
  - *How to improve scaling using replicas*
- *Recent Advances in Molecular Dynamics*
  - *Novel decomposition schemes*
  - *Partitioning schemes and MPI/OpenMP for GROMACS*
  - *Million atom simulations with NAMD*
- *Conclusions*

Objective

Molecular Dynamics (MD) programs are large, complex programs so the aim of this course is to understand how they work so they can be used efficiently and possibly modified. Writing a parallel MD program from scratch is not recommended!

Many of the techniques described here are common to parallel codes in other fields, not just MD or chemical systems (e.g. domain decomposition for fluid dynamics, astrophysics,…)

NB: In this course we will discuss only classical molecular dynamics

Introduction to Classical Molecular Dynamics

Molecular Dynamics simulation is a technique for computing the equilibrium and transport properties of a classical many body system.

For a system containing N particles, successive configurations of the system are generated by integrating Newton's laws of motion for these particles. The main result is a ***trajectory*** *that specifies how the positions and velocities of the particles vary with time.*

The longer the trajectory the wider the range of phenomena that can be studied. Today, simulations typically output trajectories of 10-100 ns (1ns=10^{-9} s) – such simulations would be unthinkable without access to parallel computers!

But there is still some way to go – simulations of protein folding would need trajectories corresponding to μs, ms or even minutes!

Molecular Dynamics milestones

1959: *First MD simulation (Alder and Wainwright)*

- *Hard spheres at constant velocity. 500 particles on IBM-704. Simulation time >2 weeks*

1964: *First MD of a continuous potential (A. Rahman)*

- *Lennard-Jones spheres (Argon), 864 particles on a CDC3600. 50,000 timesteps > 3 weeks*

1977: *First large biomolecule (McCammon, Gelin and Karplus).*

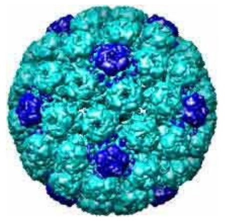
- *Bovine Pancreatic Trypsine inhibitor. 500 atoms, 9.2ps*

1998: *First μs simulation (Duan and Kollman)*

- *villin headpiece subdomain HP-36. Simulation time on Cray T3D/T3E ~ several months*

2006. *MD simulation of the complete satellite tobacco mosaic virus (STMV)*

- *1 million atoms, 50ns using NAMD on 46 AMD and 128 Altix nodes*

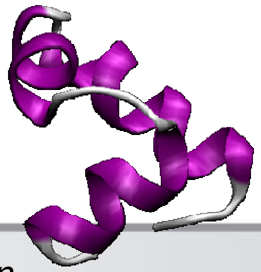


2006: *Longest run. Folding@home (computers supplied by general public!)*

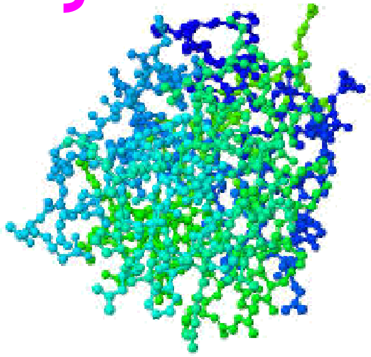
- *500 μs of Villin Headpiece protein (34 residues).*

Folding@home distributed computing

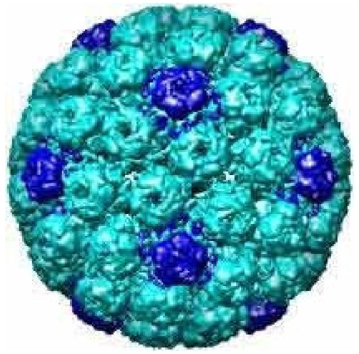
folding@home equivalent to peak ~150 Tflops (Wikipedia)



Biomolecular MD Simulation – system sizes

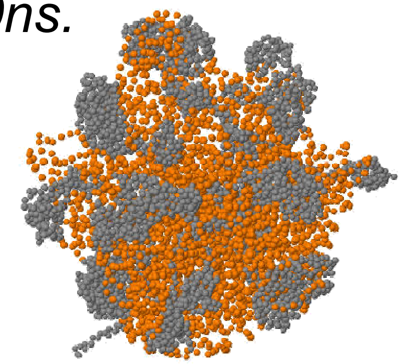


early 1990s. Lysozyme, 40k atoms

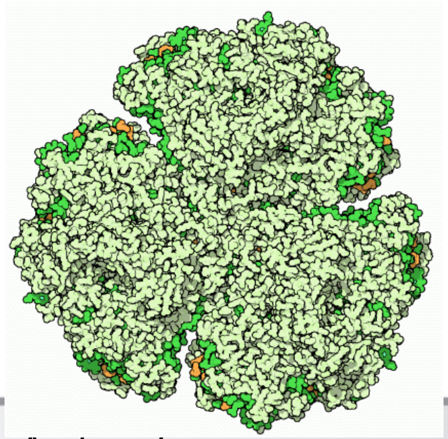


2006. Satellite tobacco mosaic virus (STMV). 1M atoms, 50ns

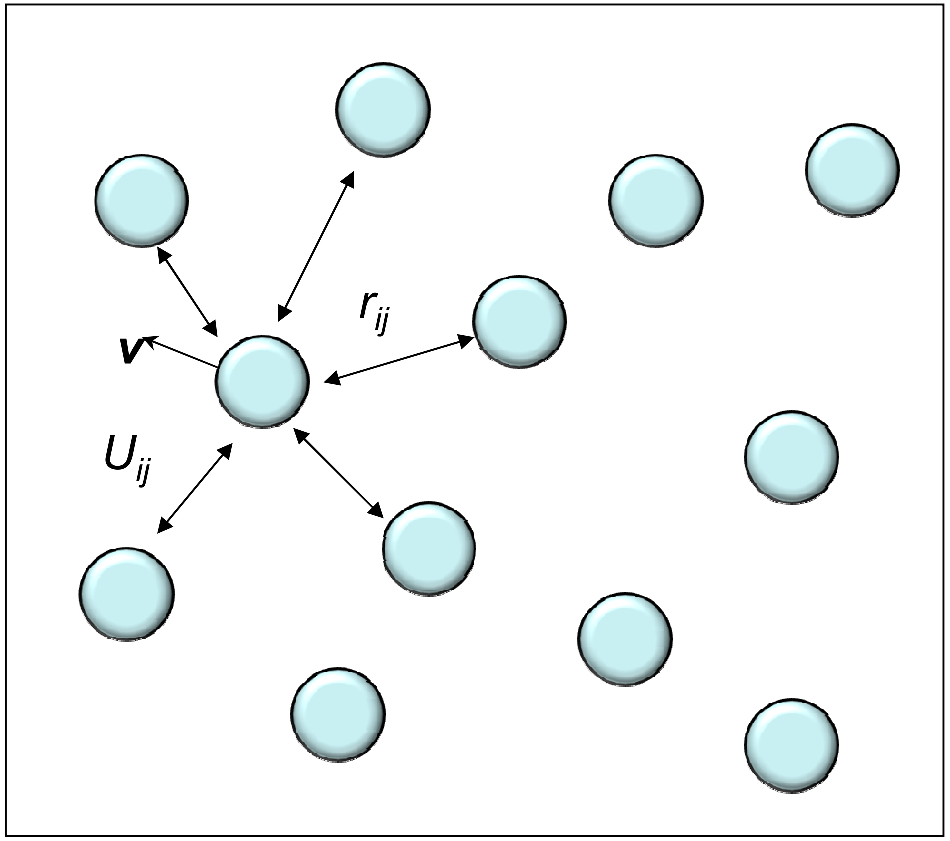
2008. Ribosome. 3.2M atoms, 230ns.



2011. Chromatophore, 100M atoms (SC 2011)



Basic principles - Simple example of interacting atoms



Usually assume pairwise interactions:

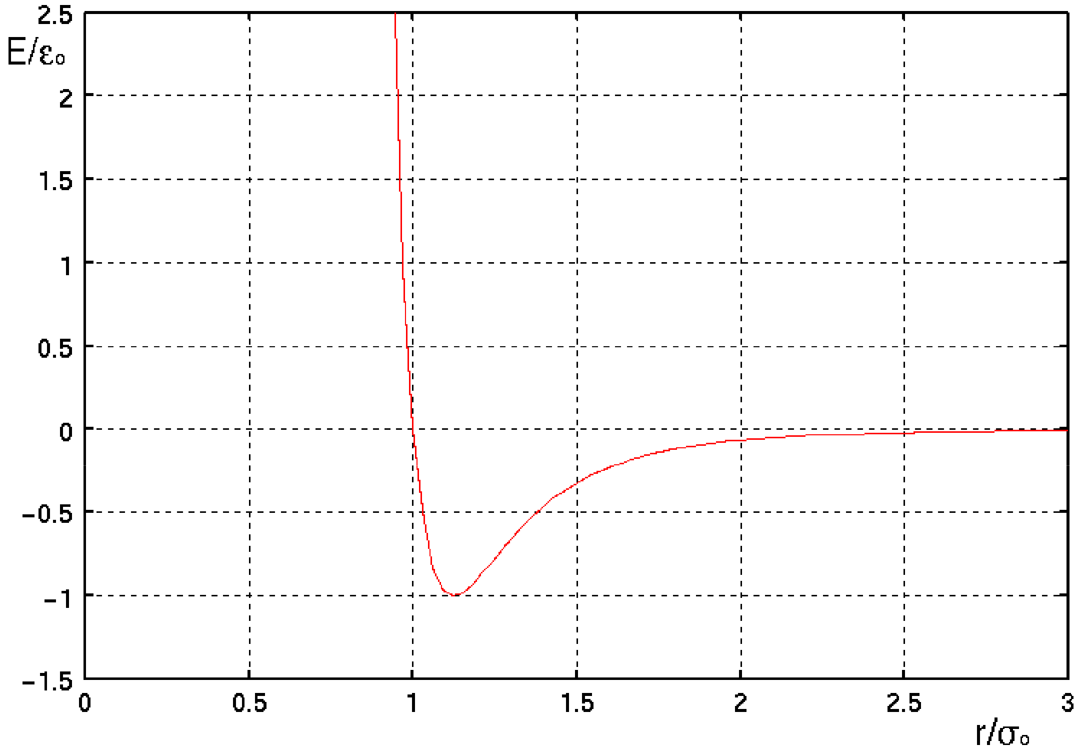
$$U = \sum_{i<j} U_{ij}$$

$$F_{ij} = -\frac{\partial U_{ij}}{\partial r_{ij}}$$

$$F_{ji} = -F_{ij}$$

Newton's third law

Example: the Lennard-Jones potential



$$U(r) = 4\varepsilon_o\left(\left(\frac{\sigma_o}{r}\right)^{12} - \left(\frac{\sigma_o}{r}\right)^{6}\right)$$

Force calculation

$$f_x(r) = -\frac{\partial U(r)}{\partial x} = -\frac{x}{r} \cdot \frac{\partial U(r)}{\partial r}$$

which for the Lennard-Jones potential:

$$f_x(r) = \frac{48x}{r^2} \cdot \left(\frac{1}{r^{12}} - \frac{1}{2}\frac{1}{r^6} \right)$$

In more complex potentials LJ normally used for the non-bonded (non-electrostatic) interactions.

Molecular Dynamics program

```
call init
T=0
do while (T.lt.Tmax)
   call compute_forces()
   call integrate_motion()
   call save_crds()
   call sample_averages()
   T = T + DT
enddo
call save_state()
stop
end
```

```
subroutine compute_energy_forces

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

subroutine integrate_motion
do i=1,N
  r(i)=r(i)+verlet(F(i))
  v(i)=v(i)+verlet(F(i))
enddo
```

Newton's third law

$$F_{ij} = -F_{ji}$$

```
subroutine compute_forces
do i=1,N
  F(i) = 0.0
enddo
do i=1,N-1
  do j=i+1,N
    fij = force(i,j)
    F(i)=F(i) + fij
    F(j)=F(j) - fij
    endif
  enddo
enddo
```

Numerical integration

Once the forces have been calculated the new positions and velocities are calculated by applying Newton's Second Law:

$$F = m \cdot a$$

within a numerical integration scheme.
A common technique for solving such differential equations is with a *finite difference method.*

$$r(t+\Delta t) \approx 2r(t) - r(t-\Delta t) + \frac{f(t)}{m}\Delta t^2 + O\left(\Delta t^4\right)$$

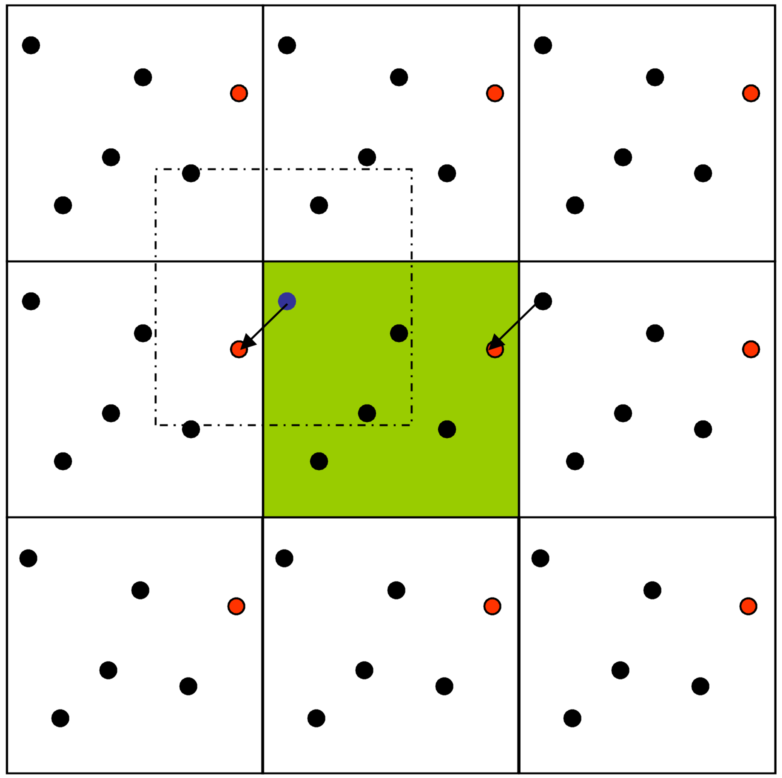
$$v(t) = \frac{r(t+\Delta t) - r(t-\Delta t)}{2\Delta t} + O\left(\Delta t^2\right)$$

* *The equations of motion are coupled so cannot be solved analytically.*

Periodic Boundary Conditions (PBC)

To avoid surface effects the simulation box is usually surrounded by exact replicas of itself. If a particle leaves the box it is replaced by its image coming in from the opposite side.

Each particle interacts with the others in the periodic system according to *the minimum image convention*.



Real molecules …

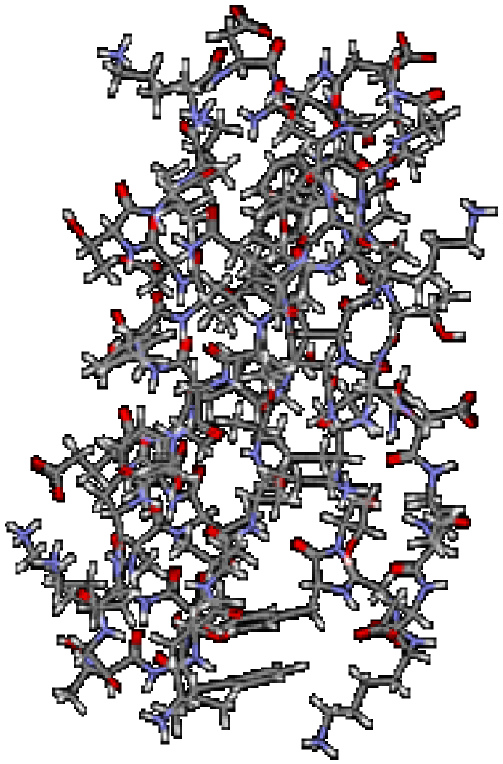
Force-fields representing complex interactions:

Algorithms for calculating electrostatic interactions (Ewald sum)

Constraints methods for fixing bond lengths (Shake)

Temperature and pressure control

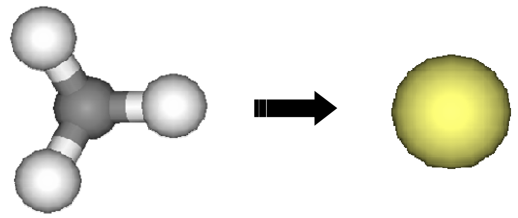
Addition of solvents and external fields (e.g. electric)

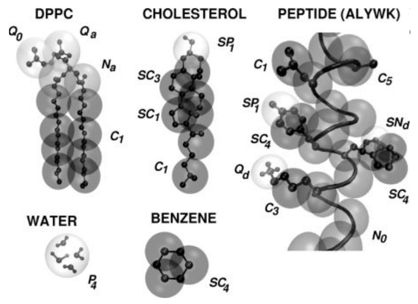


$$E_{tot} = E_{bonds} + E_{angles} + E_{dihedrals} + E_{non-bonded} + E_{electrostatic}$$

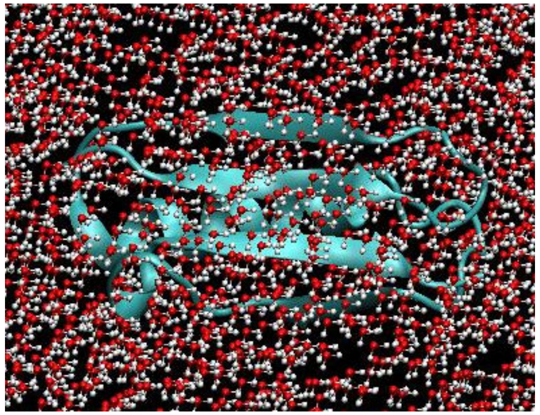
e.g. Lennard-Jones

Tricks and shortcuts

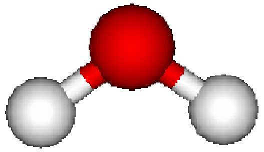




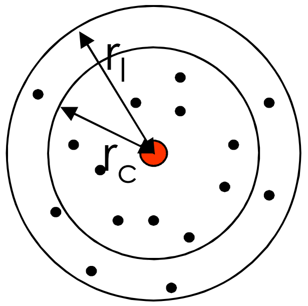
United atoms or coarse grain (e.g. Martini model)



implicit solvent (as opposed to explicit)

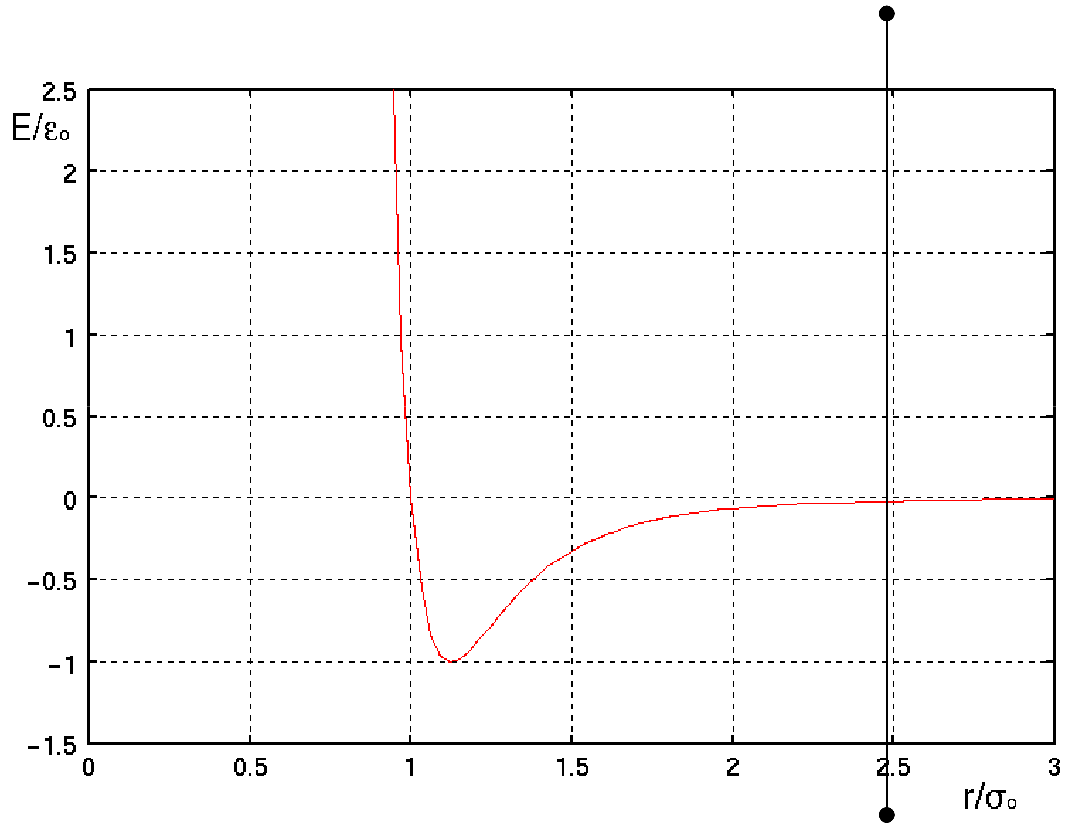


holonomic constraints (e.g. SHAKE) Δt=1fs → Δt=2fs



neighbour lists

Potential cutoff



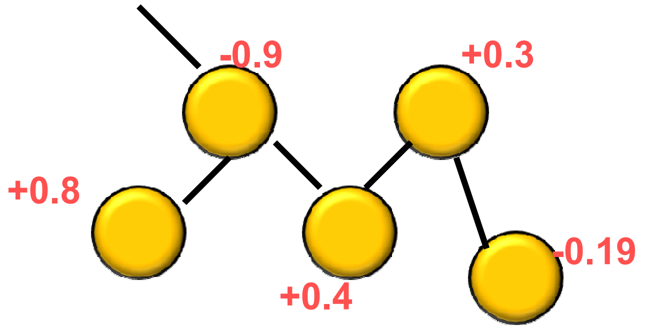
$r_C=2.5\sigma_o$

For most separations the interaction potential will be very close to zero. It is usual to "cut" the potential at some distance (the *cutoff*), ignoring all interactions greater than the cutoff distance r_c

With PBC and the minimum image convention $r_c <$ box side/2.

Electrostatic Interactions

For complex molecules electrostatic interactions are usually calculated by assigning each atom a partial charge:



The partial charges are defined by the force-field, usually via QM calculations.

The interaction energy between two isolated charges is known (Coulomb):

$$V_{ij} = \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}}$$

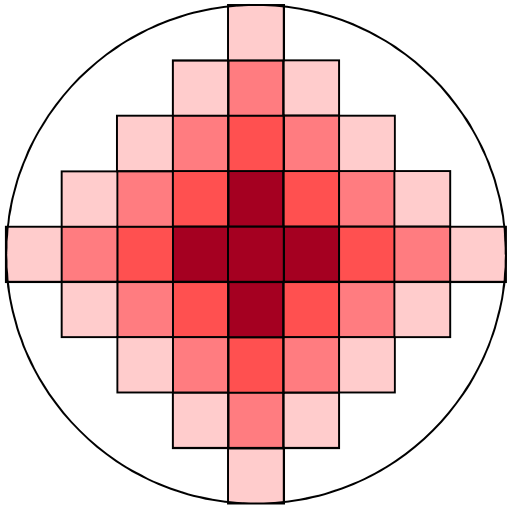
Problem: *This is a long range interaction, varying with ~1/r. (c.f LJ, ~1/r^6) and so decays to zero slowly. The box cannot be made large enough without making the simulation impracticable.*

Electrostatic Interactions -Ewald Sum (1921)

Solution for periodic systems first suggested by Ewald and others from their work on ionic crystals. Start with the interaction of a particle with all the other particles, including their images:

$$V = \frac{1}{2}\sum_{\mathbf{n}}\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{q_i q_j}{|\mathbf{r}_{ij}+\mathbf{n}|}$$

$\mathbf{n}=(n_xL, n_yL, n_zL)$



For large n the cell distribution is spherical

Electrostatic interactions - Ewald Sum

This pairwise summation converges slowly, but by assuming gaussian charge distributions around each charge it can be converted into faster converging real space (short range) and reciprocal space (long range) sums:

V = real space sum + reciprocal space sum + constant corrections

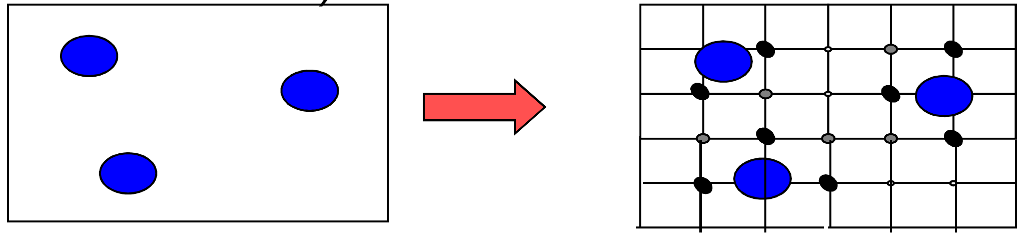
The real space term (which contains erfc(x)) can be calculated quite easily with standard libraries and usually a cutoff is applied (e.g. 9 Å).

Particle Mesh Ewald

The second term converges quickly in reciprocal space but is computationally expensive:

$$V = \frac{1}{2}\sum_{k \neq 0}\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{4\pi^2 q_i q_j}{L^3 k^2}\exp\left(-\frac{k^2}{4\alpha^2}\right)\cos(\mathbf{k}\cdot \mathrm{r}_{ij})$$

This is an N^2 problem but by replacing the point charges by a grid-based charge distribution one can use discrete FFT (Fast Fourier Transform) which scales as N lnN (e.g. Particle Mesh Ewald).



Electrostatics – Particle Mesh Ewald

Some considerations:

- *The system must be periodic. Not a problem if we use Periodic Boundary Conditions.*
- *The simulation box must be neutral overall, otherwise formally the sums do not converge. Add counterions (e.g Na+ or Cl-) to neutralise any net charge.*
- *By re-distributing charge on a grid ("mesh") can use the Fast Fourier Transform algorithm to reduce problem size to N ln N.*
- *PME used not only in electrostatics/chemistry but in any situation with long-range forces (e.g. gravitation).*

Despite PME and FFT, the electrostatic calculations are still computationally expensive. Fortunately, they can be parallelised (although still a bottleneck).

Parallel Molecular Dynamics

In a (serial) molecular dynamics program often 70-90% of the CPU time is spent in the calculation of energies and forces -> this is the first place to look when optimising or parallelising a program.

Molecular dynamics is inherently a parallel problem.

There are two main classes of algorithm used to parallelise an MD program:

- *Atom and Force decomposition (also called "Replicated data")*
- *Spatial (or Domain) decomposition*

see. S. Plimpton et al., 1995

GROMACS timings

```
Computing:                               M-Number         M-Flops  % Flops
-----------------------------------------------------------------------------
 LJ                                   66460.022385     2193180.739     2.8
 Coul(T)                              67295.126727     2826395.323     3.6
 Coul(T) [W3]                          1361.881485      170235.186     0.2
 Coul(T) + LJ                        113027.749257     6216526.209     7.9
 Coul(T) + LJ [W3]                    21305.487096     2940157.219     3.7
 Coul(T) + LJ [W3-W3]                 67057.921884    25616126.160    32.5
 Outer nonbonded loop                 16258.069653      162580.697     0.2
 1,4 nonbonded interactions            1814.923008      163343.071     0.2
 Calc Weights                         11664.933552      419937.608     0.5
 Spread Q Bspline                    248851.915776      497703.832     0.6
 Gather F Bspline                    248851.915776     1493111.495     1.9
 3D-FFT                             4145210.365398    33161682.923    42.1
 Solve PME                              819.609600       52455.014     0.1
 NS-Pairs                             72105.130813     1514207.747     1.9
 Reset In Box                           264.244768         792.734     0.0
 CG-CoM                                 650.966640        1952.900     0.0
 Angles                                1587.865536      266761.410     0.3
 Propers                                397.158480       90949.292     0.1
 Impropers                               88.972464       18506.273     0.0
 RB-Dihedrals                          1408.960128      348013.152     0.4
 Virial                                 652.323390       11741.821     0.0
 Stop-CM                                 79.670544         796.705     0.0
 Calc-Ekin                             3890.254368      105036.868     0.1
 Lincs                                 1446.799037       86807.942     0.1
```

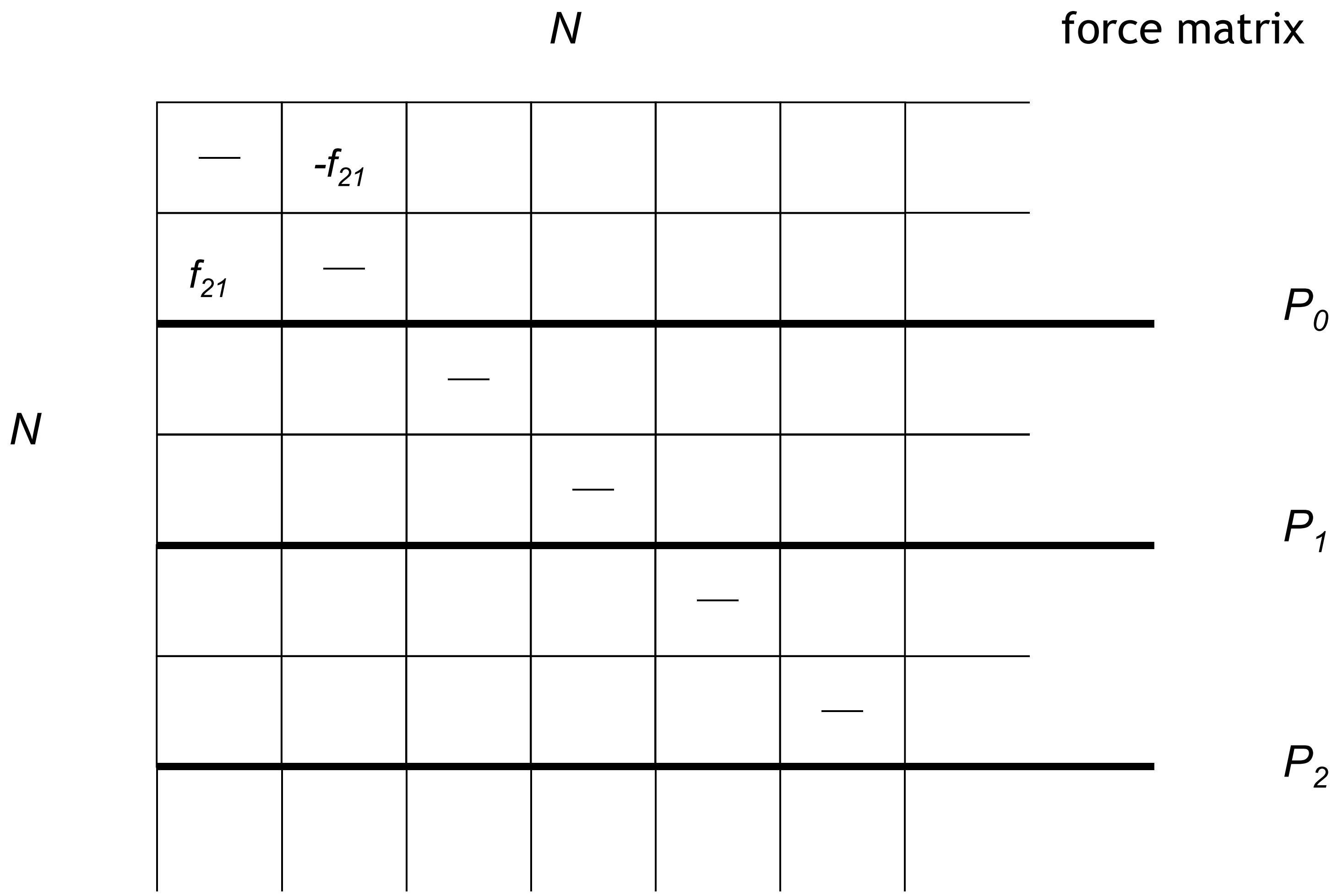
Atom decomposition algorithm

For every MD cycle:

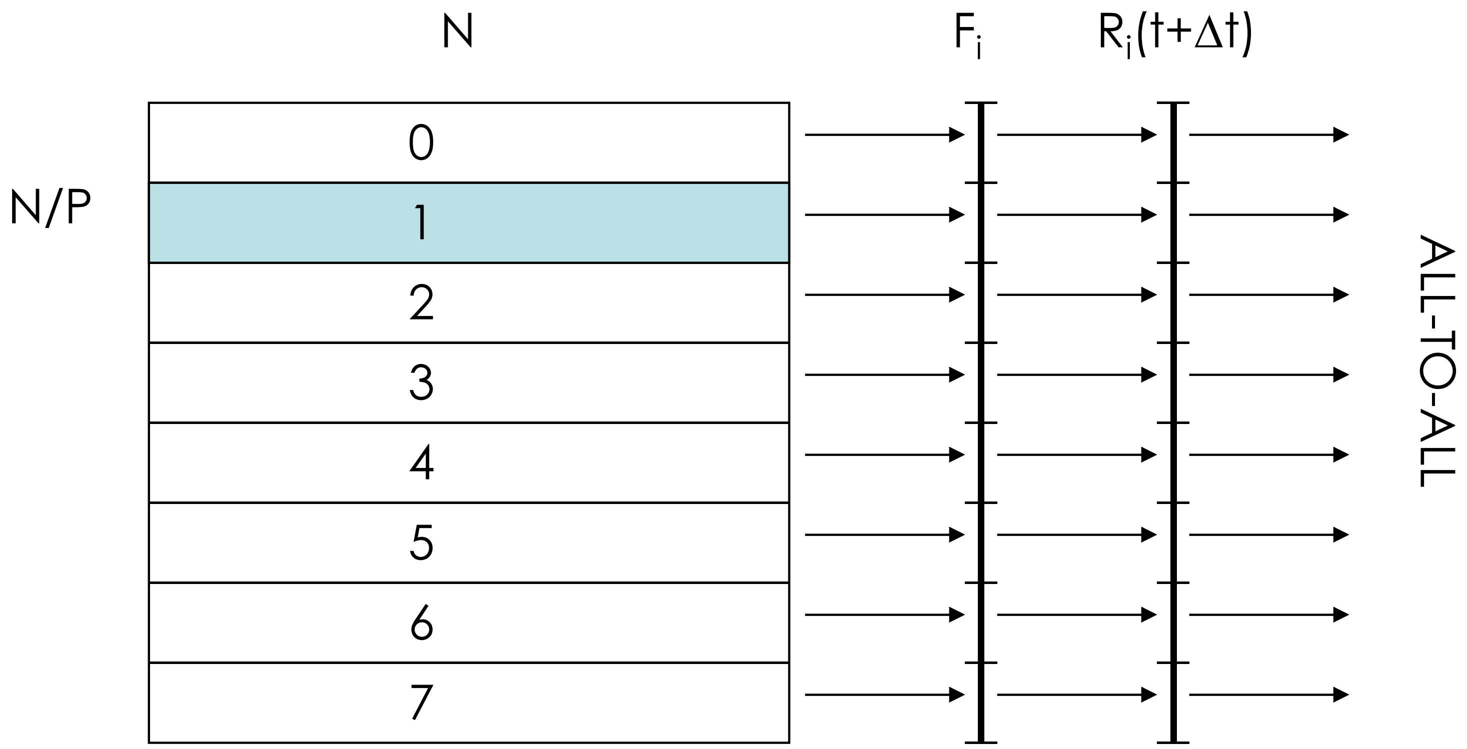
- *Each proc contains a copy of the system*
- *A group of N/P atoms is assigned to each proc (no spatial relation between the assigned atoms)*
- *Each proc calculates the interactions of each of its N/P atoms with the other N-1*
- *Each proc integrates the equations of motion for its N/P atoms*
- *Each proc communicates the updated positions to all the other procs*

N=no. of atoms
P=no. of processors

Atom decomposition



Atom decomposition



All-to-All

Every processor must have all the positions from all the other processors.
It is a common communication type. Corresponds to the MPI_ALLGATHER call (not MPI_ALLTOALL!):
After the call all the processors have the same image in memory.

Atom decomposition - summary

The MD force computation and integration is evenly shared across the processors -

- but the algorithms require global communication as each processor needs the information of all the other processors – communication scales as N (not P)

simple to implement

- *change loops in N to loops in N/P*

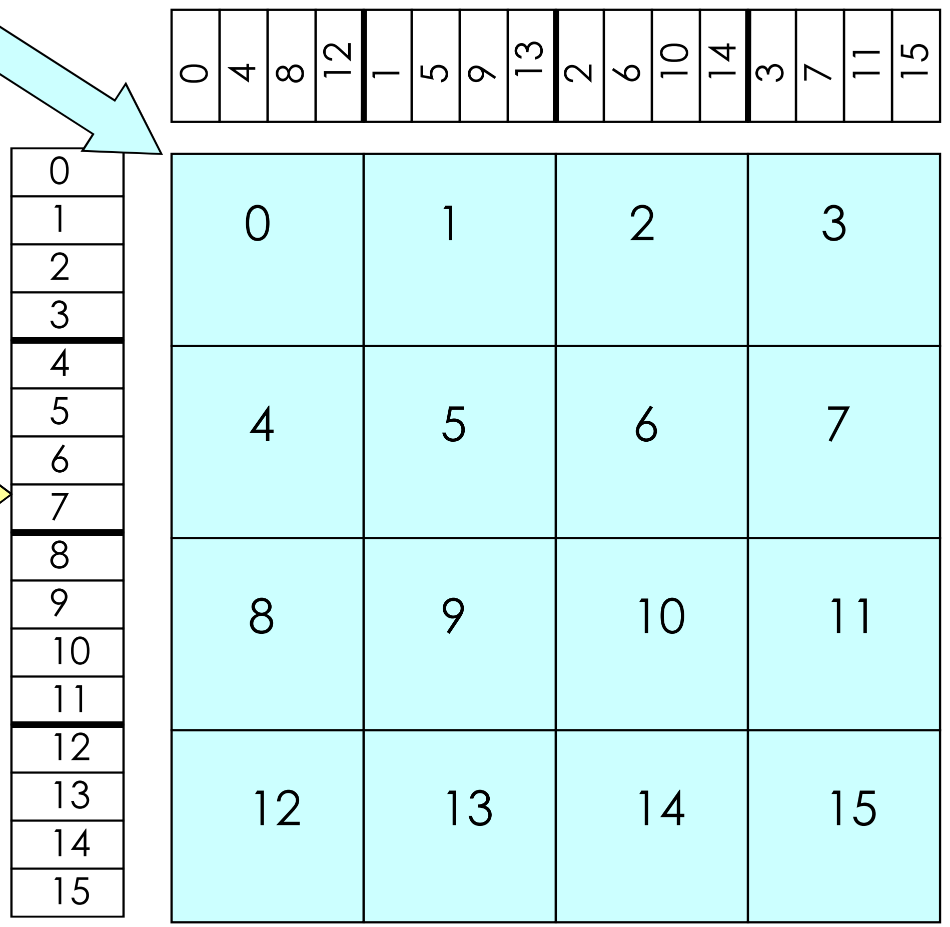
Force decomposition algorithm

- *In this algorithm the parallelism is based on the block decomposition of the force matrix rather than on dividing up the atoms (which corresponds to the row-wise decomposition of the force matrix).*
- *Processors are assigned to square areas of the force matrix, with size N^2/P. This reduces communication costs since communication is now between particular rows and columns, rather than over the whole matrix.*
- *Communication costs for integrating the equations of motion can also be reduced by rearranging the columns in a particular way.*

Force decomposition algorithm

distribution of the work between 16 procs for calculating the forces

distribution of the work between 16 procs for integrating the equations of motion



Force decomposition

Like Atom decomposition, MD computations are distributed evenly across the processors

But less information is required by each processor O(N/√P), so lower memory and communication costs.

Also relatively simple to code, although some pre-processing may be necessary to ensure load balancing.

But both Atom and Force decomposition require global-communications.

These algorithms suffer from the fact that they don't exploit the locality of interatomic interactions, many of which are short-ranged.

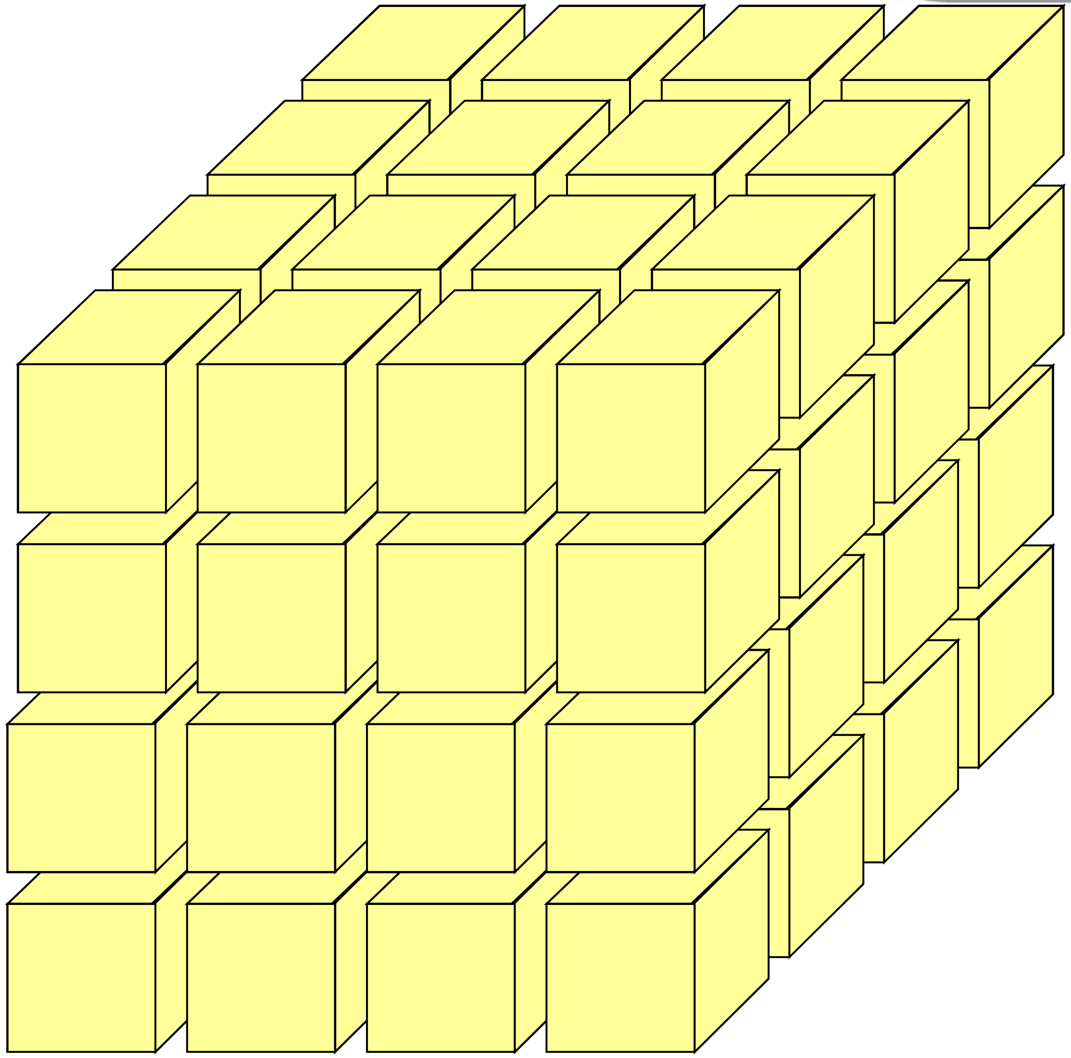
Spatial (or domain) decomposition algorithm

Here each processor is assigned to a spatial region of the simulation box and stores only a portion of the whole system. This has two components:

- The atoms which lie in that region and the forces between them.
- Atom positions and forces from neighbouring regions owned by other processors.

In order to minimise the surface with respect to the volume, and hence the communications, it is important to use regions that are as cubic as possible. In any case the communications are reduced since it is not necessary to update the whole system in local memory.

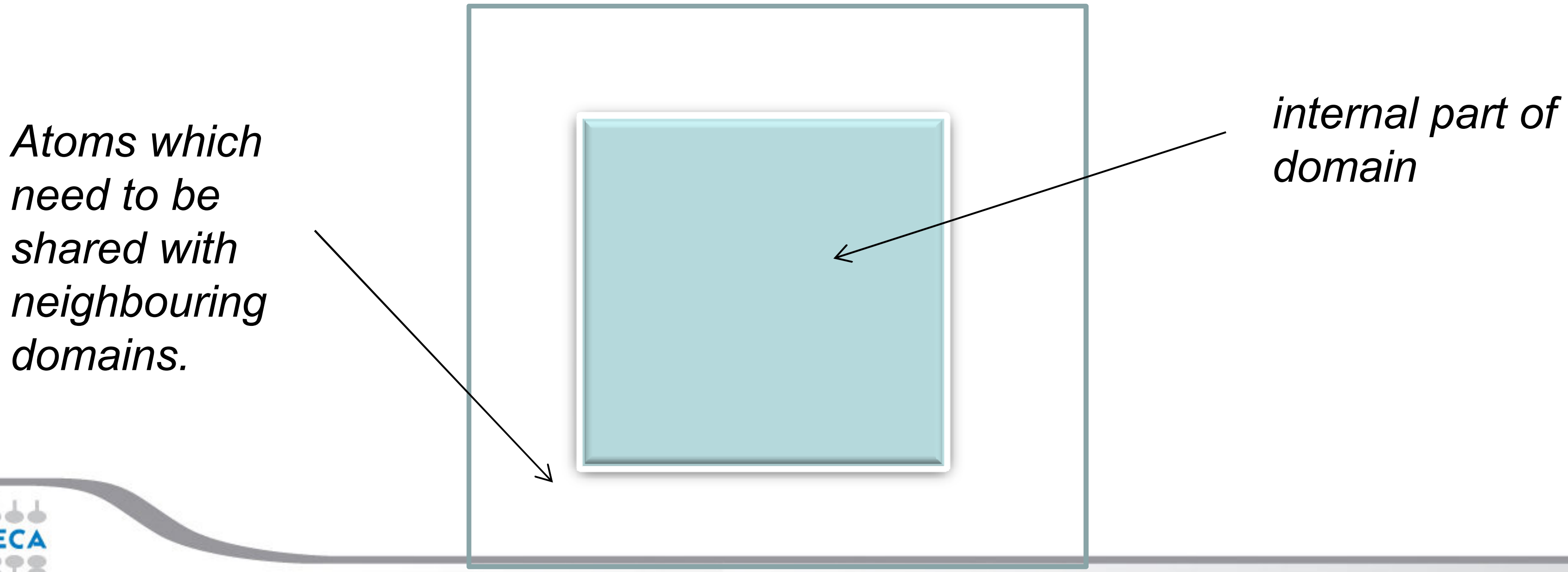
Domain decomposition



Domain decomposition

Each domain will have 2 types of atoms:

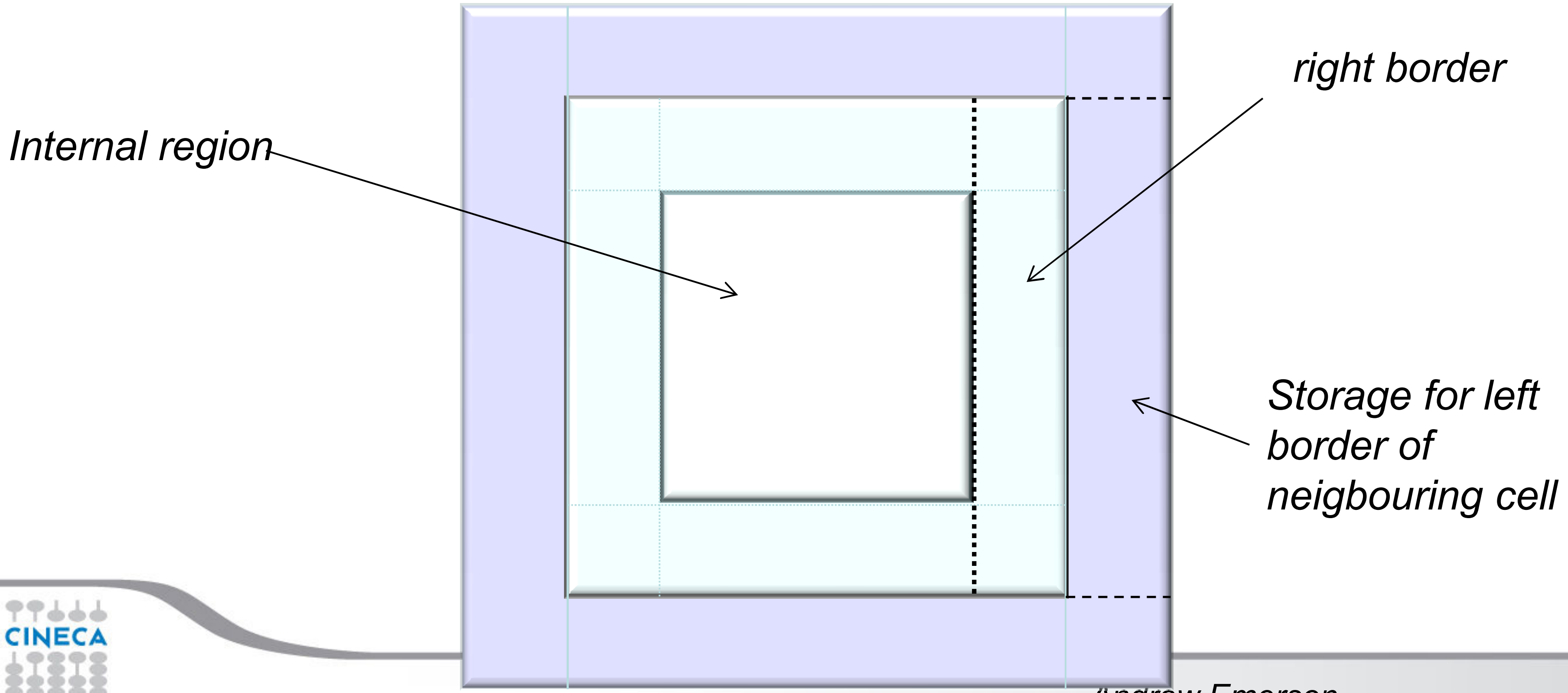
1. *Those which can be entirely managed by the processor, i.e. all the interactions are within the domain*
2. *Those for which the interactions extend outside the domain. Here data have to be sent/received to/from neighbouring domains.*



domain decomposition

The difficult part of DD is then to communicate coords 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).

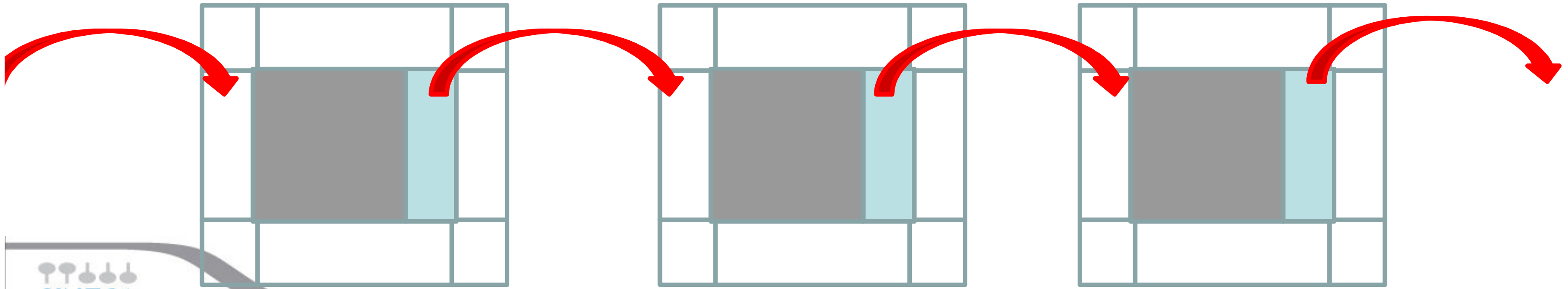


domain decomposition - neighbour communication

neighbour coords in each dimension conveniently exchanged via
mpi_cart_shift *and* **mpi_sendrecv** *calls*

First pass, x-direction, left to right

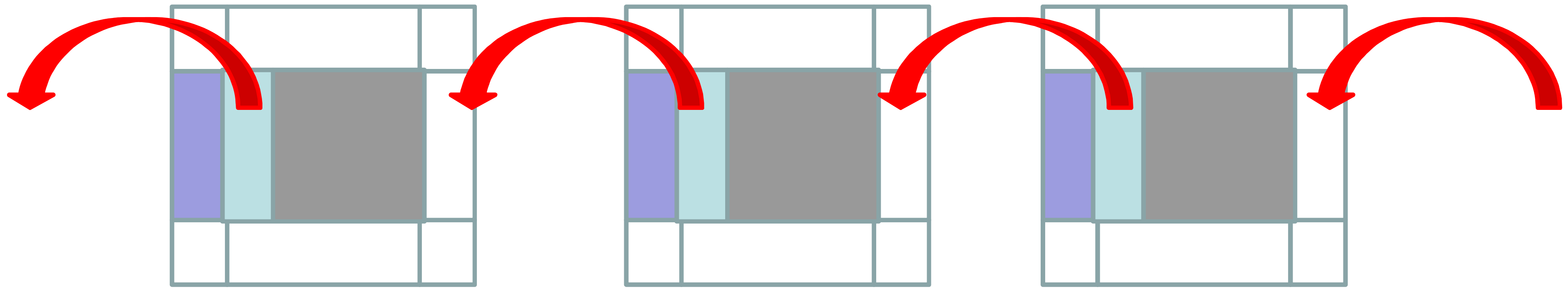
```
call mpi_cart_shift(mpi_box,1,1,proc_left,proc_right,ierror)

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



domain decomposition

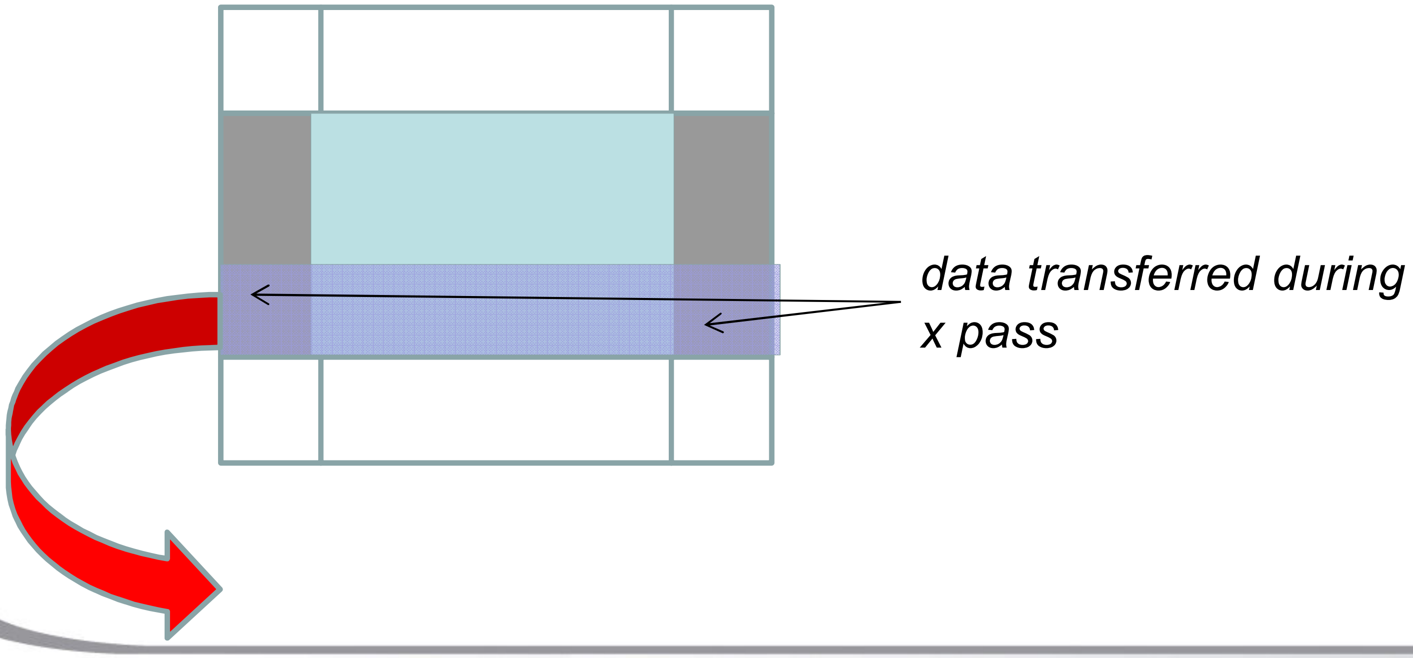
Then right to left

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



domain decomposition

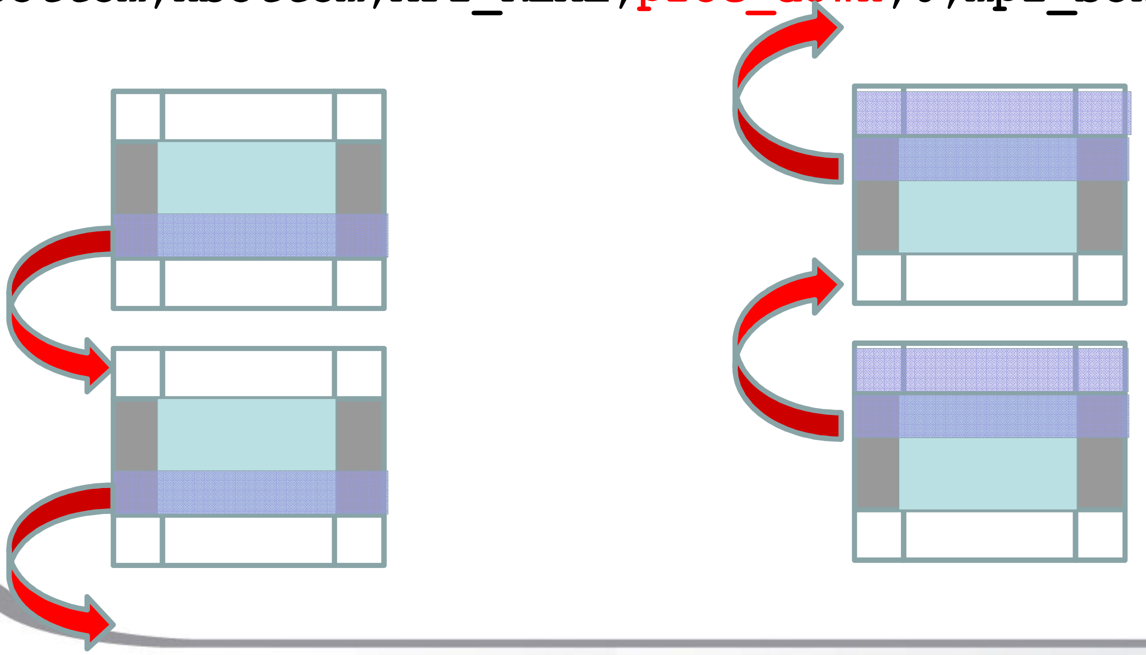
We can repeat in the y direction but to ensure we transfer the corners we need to include data transferred in the x pass



21st Summer School of PARALLEL COMPUTING

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.

Next step, solve equations of motion and repartition coordinates as before.

Domain decomposition

Advantages

- *Exploits* **locality** *of atomic interactions, minimizing communications (no All-to-All) and memory required per processor*
- *scalable, for large systems*
- *can exploit MPI cartesian topology*

Disadvantages

- *needs large system, otherwise domain size too small. As no. of processors increases eventually stops scaling*
- *for inhomogeneous systems (liquid+vapour) load balancing problems as some procs have too few atoms.*

Parallelisation Conclusions

Atom decomposition

–Simplest to implement, with easy load balancing.

–But (global) communication **O(N)** *costs begin to dominate on large numbers of processors (not the best scaling).*

Force decomposition

–Also relatively simple to code, although requires some pre-processing to avoid load balancing problems.

–Generally scales better than atom decomposition, but communication cost is still high **O(N/√P)**

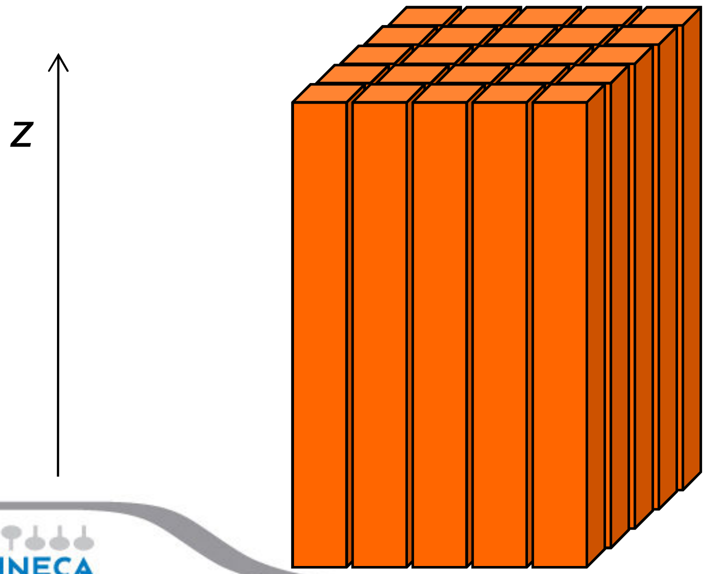
Domain decomposition

*–Most difficult to code, but generally offers best scaling with large number of processors (***local communication***)* **O(N/P)** *. May be possible to exploit cluster topology. PME can also be parallelised although still expensive*

Parallelisation of Electrostatics with Domain Decomposition and PME

PME can be parallelised with a DD scheme but 3D FFT is very inefficient for many processors (or small N) because of all-to-all global communications (MPI_AlltoAll).

GROMACS and NAMD use instead a 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

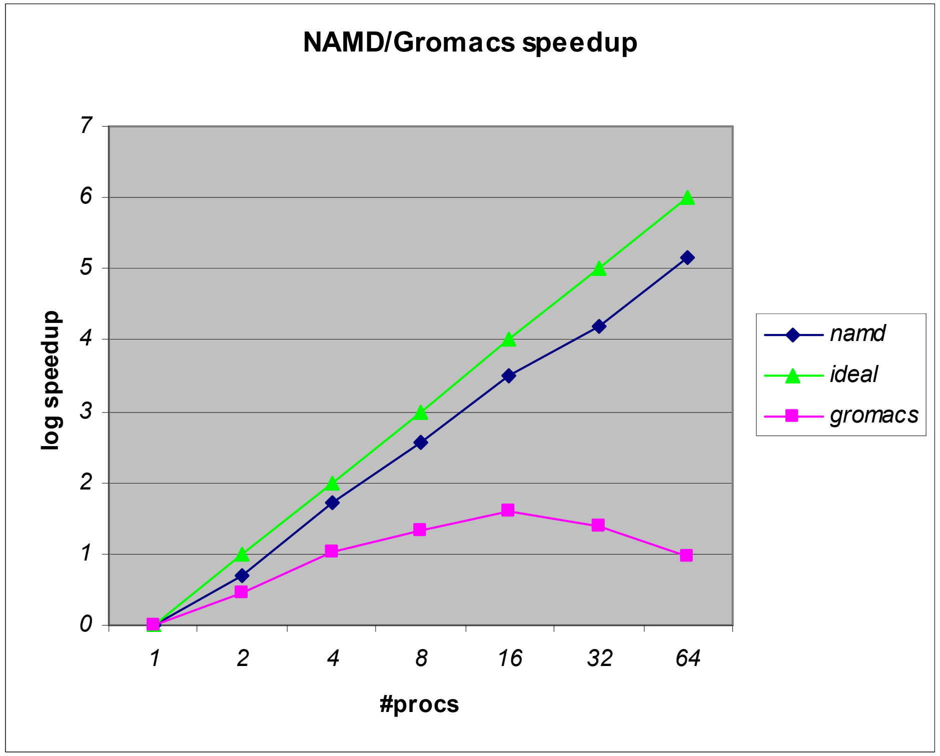
Does Domain Decomposition Work?

Compare

- *GROMACS v 3.x and earlier with force-decomposition schemes*
- *GROMACS v 4.x with domain decomposition*
- *NAMD with domain decomposition*

Disclaimer: *There are many other differences between programs which could affect performance but parallel scaling is a good indicator of the parallelization scheme.*

Some NAMD/Gromacs 3.3 benchmarks

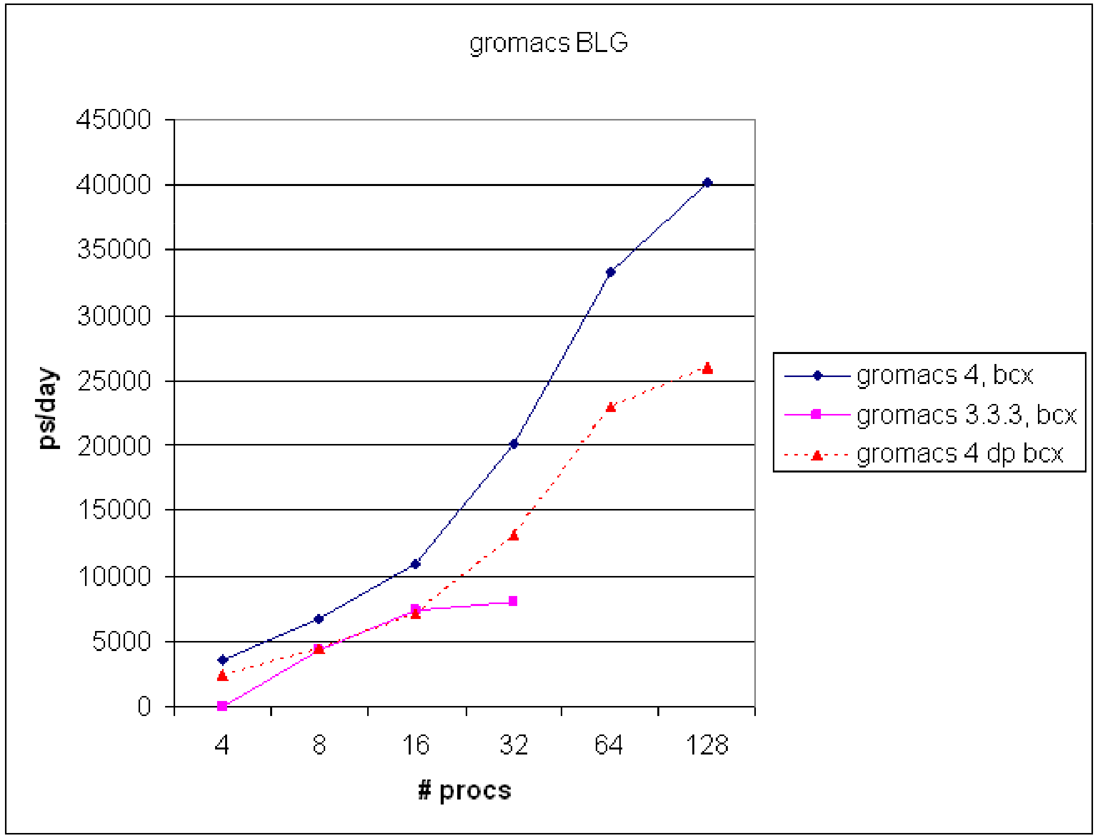


Speedup R

$$R = \frac{P_N}{P_1}$$

where P = performance (e.g. ps/day)

Comparison force-decomposition and domain-decomposition Gromacs 4 (First release 2008)



Why do MD programs stop scaling ?

The system is too small compared to the number of cores -

1. *Limits of domain decomposition –with few particles/proc the domain size becomes too small*

2. *The parallel PME calculation contains all-to-all communication (in the 3D FFT) and this cost varies as N^2 .*

Scalability is generally higher in simulations without PME (for example with an electrostatic cutoff or implicit solvent) so this is usually considered the bottleneck to strong scaling.

As a rule of thumb, 100-200 atoms/core is going to be close to the scaling limit.

Why do MD programs stop scaling ?

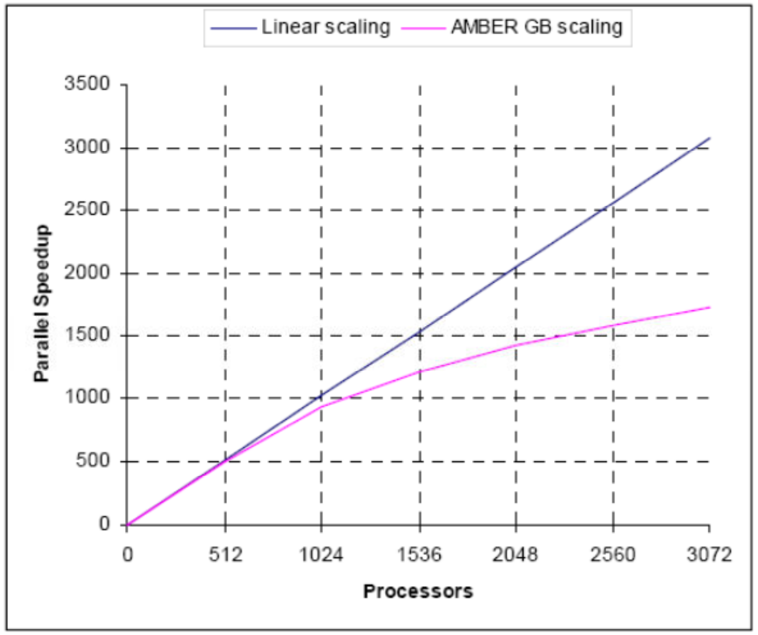


Figure 1. Parallel scaling of AMBER on Blue Gene. The experiment is with an implicit solvent (GB) model of 120,000 atoms (Aon benchmark).

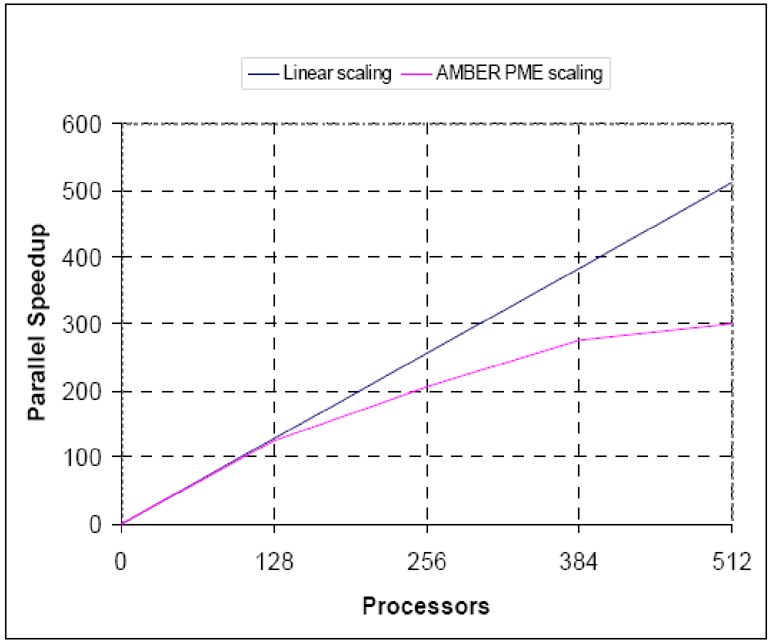


Figure 2. Parallel scaling of AMBER on Blue Gene. The experiment is with an explicit solvent (PME) model of 290,000 atoms (Rubisco).

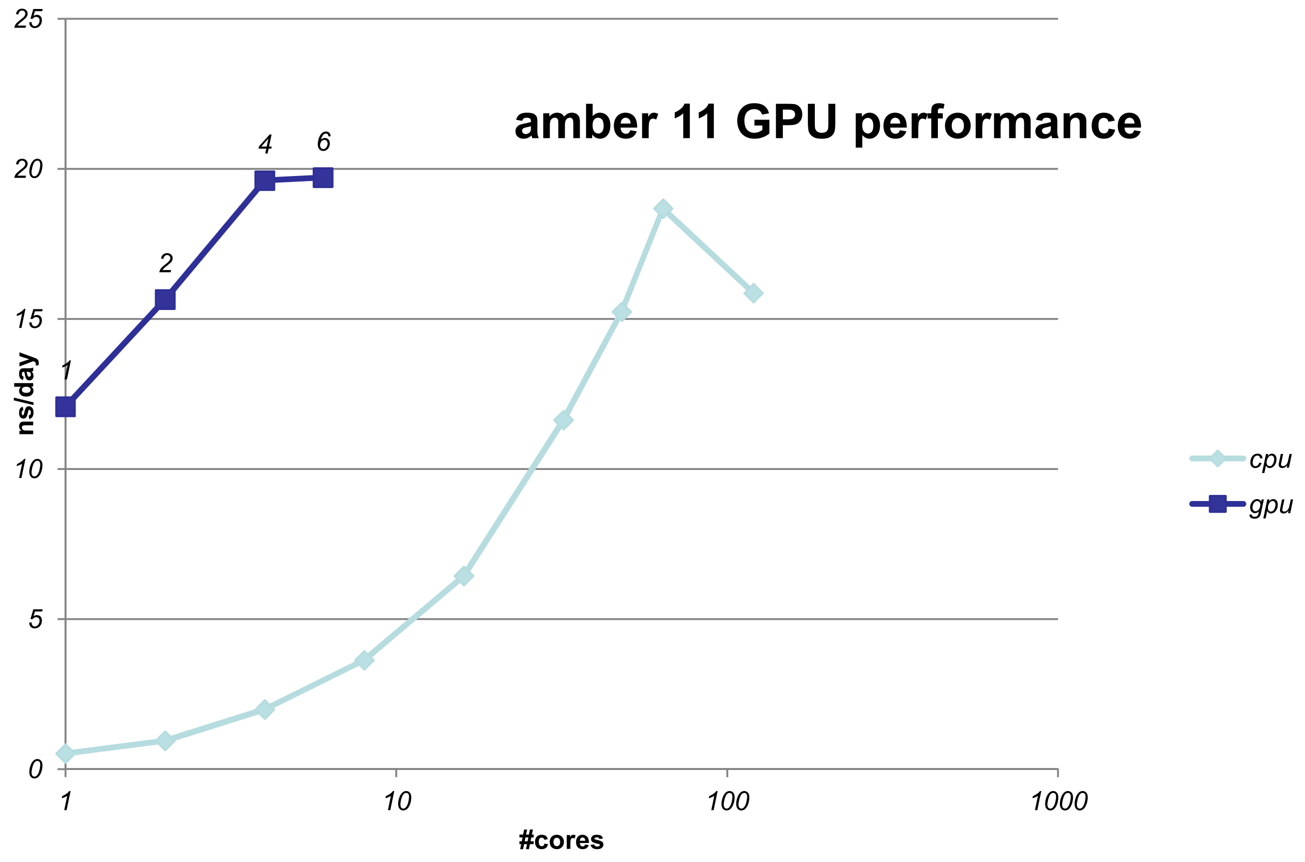
***Life Sciences Molecular Dynamics Applications on the IBM System Blue Gene Solution: Performance Overview*,**
http://www-03.ibm.com/systems/resources/systems_deepcomputing_pdf_lsmdabg.pdf

Why do MD programs stop scaling ?

The Bluegene and other multi-thousand core architectures represent a challenge for projects based on molecular dynamics since often a minimum scaling is required.

Supercomputer	Minimum scaling requirements
JUGENE	*8192*
CURIE	*512 (thin nodes), 2048 (fat nodes)*
HERMIT	*2048*
SuperMUC	*4096*
FERMI (CINECA)	*2048*

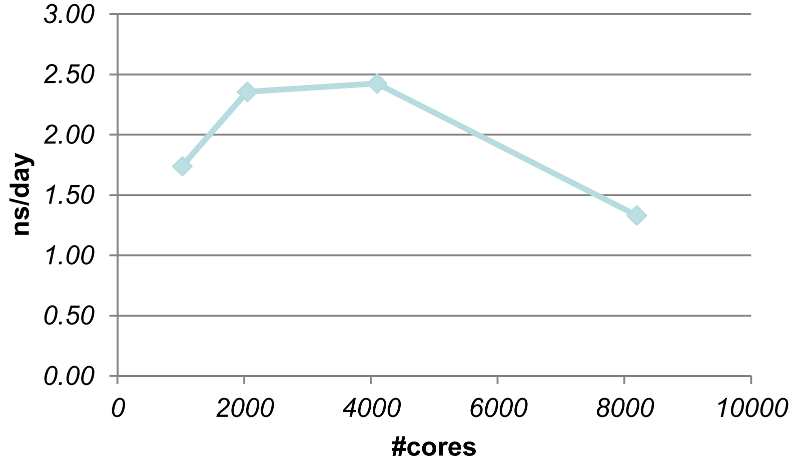
Why do MD programs stop scaling ?



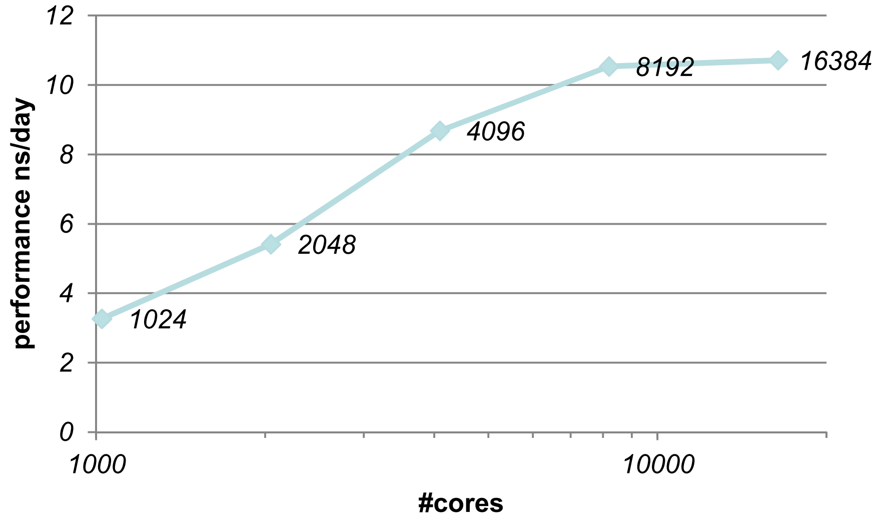
DNA+water (40K atoms)

Why do MD programs stop scaling?

GROMACS BG/P scaling for SPC water (0.5M molecules)



GROMACS BG/P scaling for d.kv12 membrane (1.8M atoms)



For this benchmark we had to duplicate the std GROMACS benchmark d.kv12 ion channel 16 times ! ⟶

How can I increase the parallel scaling ?

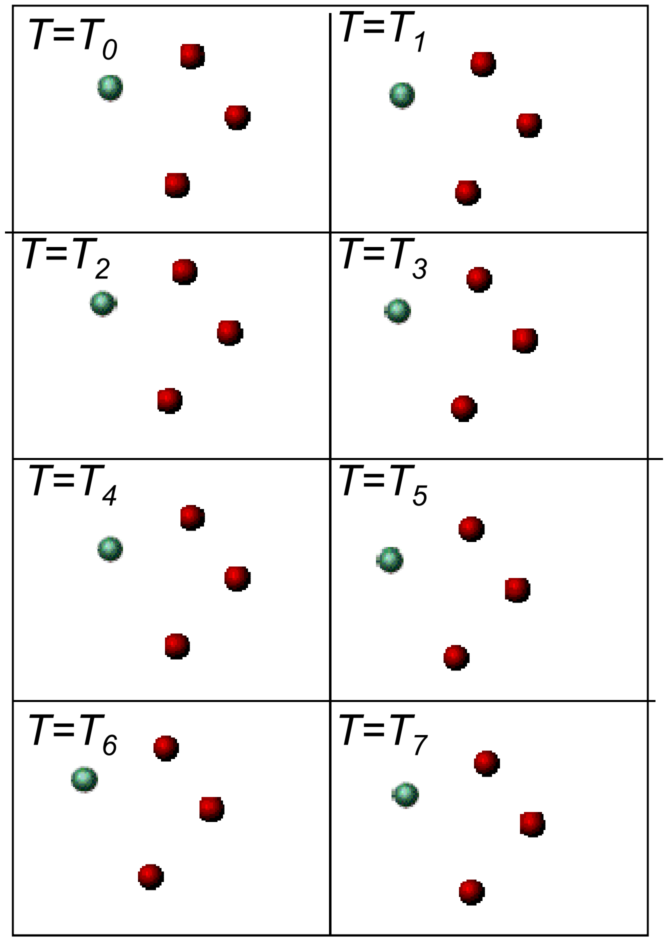
1. *Increase the system size.*
   – *But not always possible if your problem size is "fixed" (i.e. because you are studying a particular molecule)*

2. *Design a project which uses multiple replicas of the same system.*
   – *Examples include replica exchange (REMD), metadynamics, ensemble simulations,..*

Replica Exchange Molecular Dynamics

Replica Exchange Molecular Dynamics

- *Used to prevent simulation from getting “stuck” in local minima.*
- *Run multiple simulations (“replicas”) at different temperatures or with varying potential parameters.*
- *At regular intervals the n replicas exchange coordinates and then re-continue their trajectories.*
- *For a BG with N cores the individual replicas need only scale up to N/n cores for efficient performance.*



Replica simulations

Some advice ....

- *check if the replica model requires shell "scripting" (e.g. NAMD) or is built-in to the executable (e.g. GROMACS). If a script is required then*
    1. *probably wont work on Bluegene systems since scripts cannot be run on nodes (except python for BG/Q)*
    2. *will probably need customization to run replicas in parallel.*

  This also applies to "plugins" such as PLUMED.
- *dont mention the word "ensemble" in project applications as it might be read as "embarassingly parallel simulations"*

Recent advances in Molecular Dynamics -novel decomposition schemes

- *Problem with domain decomposition occurs when density of particles is uneven or fluctuates.*
- *Can be mitigated by "zonal" (or "neutral territory") methods, where forces between particles i and j are not necessarily calculated on a processor where either of particles i or j resides.*
- *Gromacs uses a zonal method called the "eighth-shell" method, with reduced communication wrt standard dd. Other methods incl "midpoint" (Desmond).*
- *Like NAMD, Gromacs 4 now has Dynamic Load Balancing.*

J. Chem. Theory Comput. C

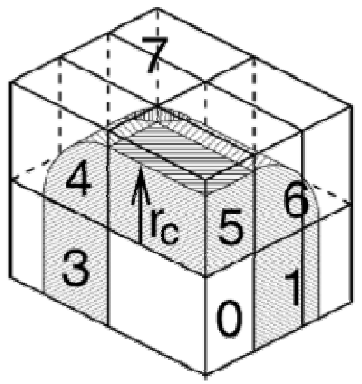


Figure 1. A nonstaggered domain decomposition grid of 3 × 2 × 2 cells. Coordinates in zones 1 to 7 are communicated to the corner cell that has its home particles in zone 0. r_c is the cutoff radius.

Hess et al., J. Chem, Theory Comput. C, 2007

Features of Molecular Dynamic programs

- *Most time consumed in the force-loops*
- *Memory requirements are low*
- *Since all data can be held in memory:*
  - *I/O is limited and restricted to restart and trajectory files which are accessed infrequently*
  - *Not much need for hybrid MPI/OpenMP versions.*

- *For MPI programs, rank 0 normally does I/O and more sophisticated approaches such as MPI-IO or HDF5 rarely used.*

Recent Advances in Molecular Dynamics: GROMACS

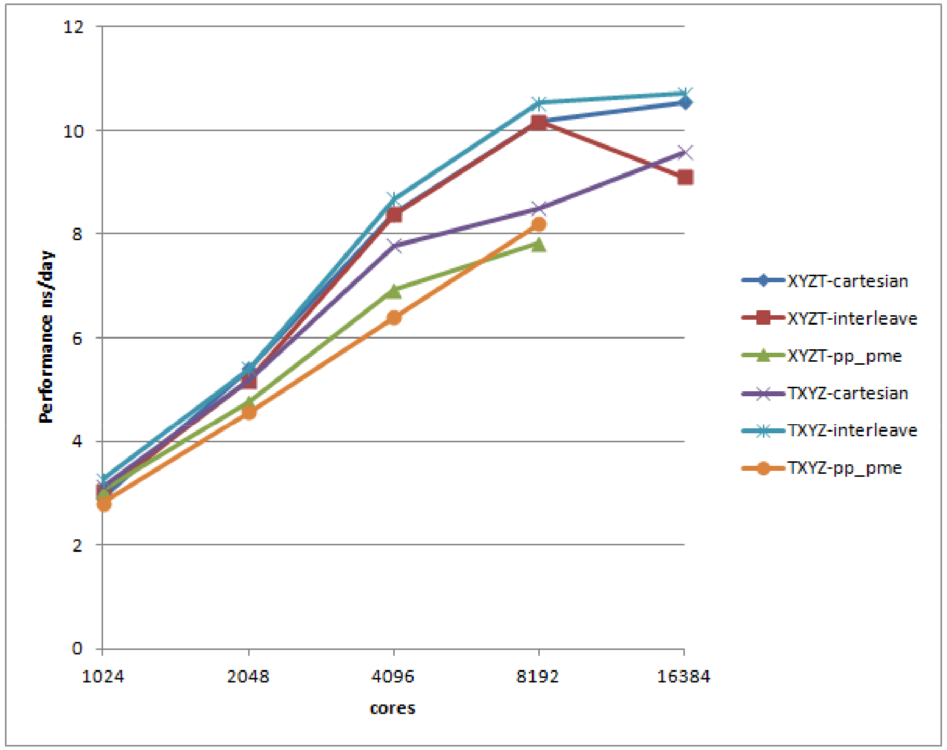
Particle-Particle (PP) and PME interactions can be decoupled so could be beneficial to assign separate nodes to PME part to reduce the communications for FFT.

GROMACS 4.x allows separate nodes to be assigned to PME calculations:

```
mpirun  mdrun -npme 4 md.conf
```

Rule of thumb is PP:PME = 3:1 but `g_pme` *utility allows this to be tested.*

Recent Advances in Molecular Dynamics: GROMACS



GROMACS also allows the partitioning scheme between the PP/PME nodes to be varied.

Can be combined with MPI rank mapping scheme of Bluegene.

PP	*PP*	*PP*	*PME*
PP	*PP*	*PP*	*PME*

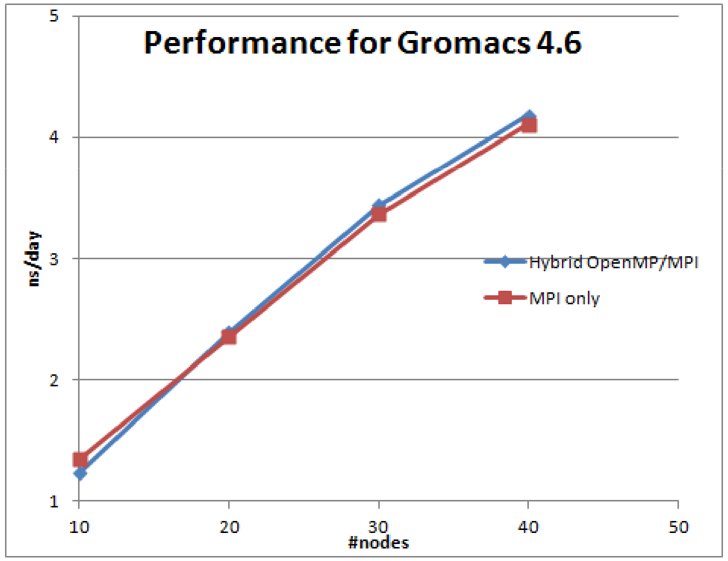
Cartesian, 2 PME nodes

PP	*PME*	*PP*	*PME*
PP	*PME*	*PP*	*PME*

Interleave, 4 PME nodes

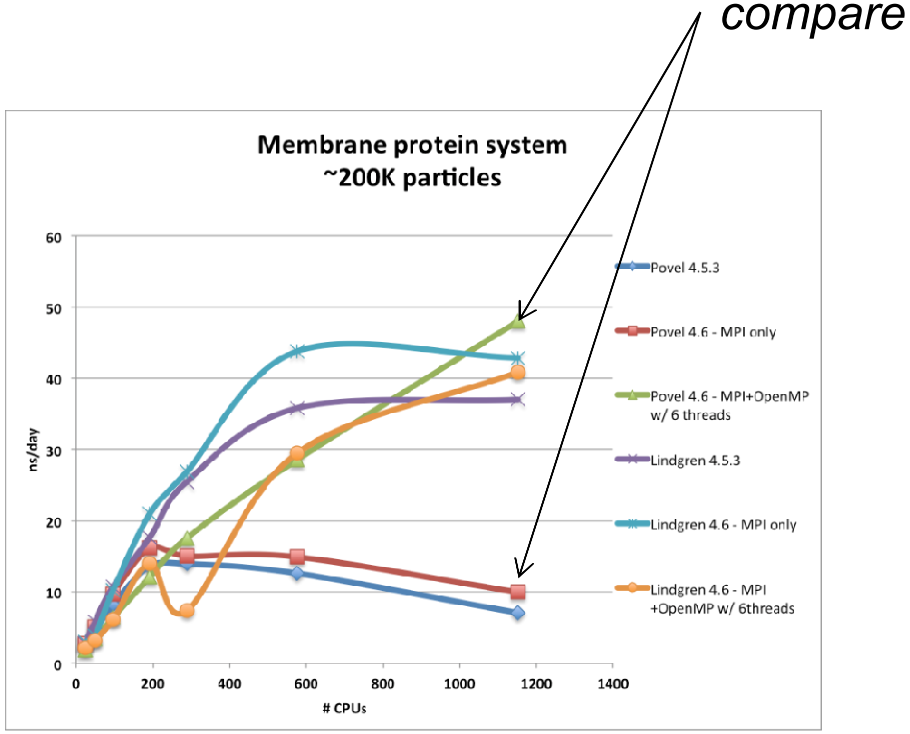
Recent Advances in Molecular Dynamics- hybrid MPI/OpenMP

GROMACS v4.6 can use OpenMP threads for the PME but only makes sense for very high number of cores or slow networks.



PLX – fast network, few nodes → no difference

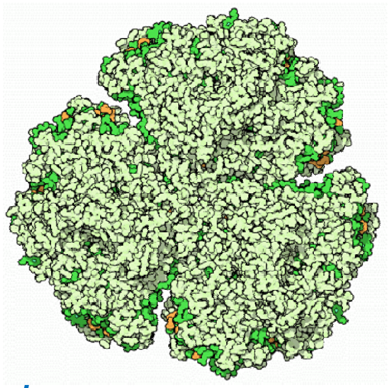
compare



Recent Advances in Molecular Dynamics - NAMD

Very large systems (100 M atoms) pose challenges for conventional MD programs:

- *Initialization step for reading molecular data can become slow and exhaust memory on start-up.*
- *Time required to store restart files and trajectories starts becoming significant*
- *Large memory footprint*
- *Difficult to get strong scaling results*



See: “Enabling and Scaling Biomolecular Simulations..”, Mei et al, Proceedings SC2011

Recent Advances in Molecular Dynamics - NAMD

To address these issues developers have created a new version:

- *Parallel IO:*
  - *Reduce start-up time and initialization memory*
  - *Reduce restart file and trajectory output overheads by writing to multiple files, which are then post-processed into a single file*
- *Multi-threaded runtime:*
  - *Reduce memory footprint and no intra-node comm. Reduces also start-up as fewer MPI processes spawned.*
  - *Allocate threads as communication or compute to reduce communication overheads.*
  - *Explicitly setting core affinity (communication thread on "noisy core 0")*
  - *Adaptive overlap of communication and calculation*

Recent Advances in Molecular Dynamics – NAMD

Communication Optimization

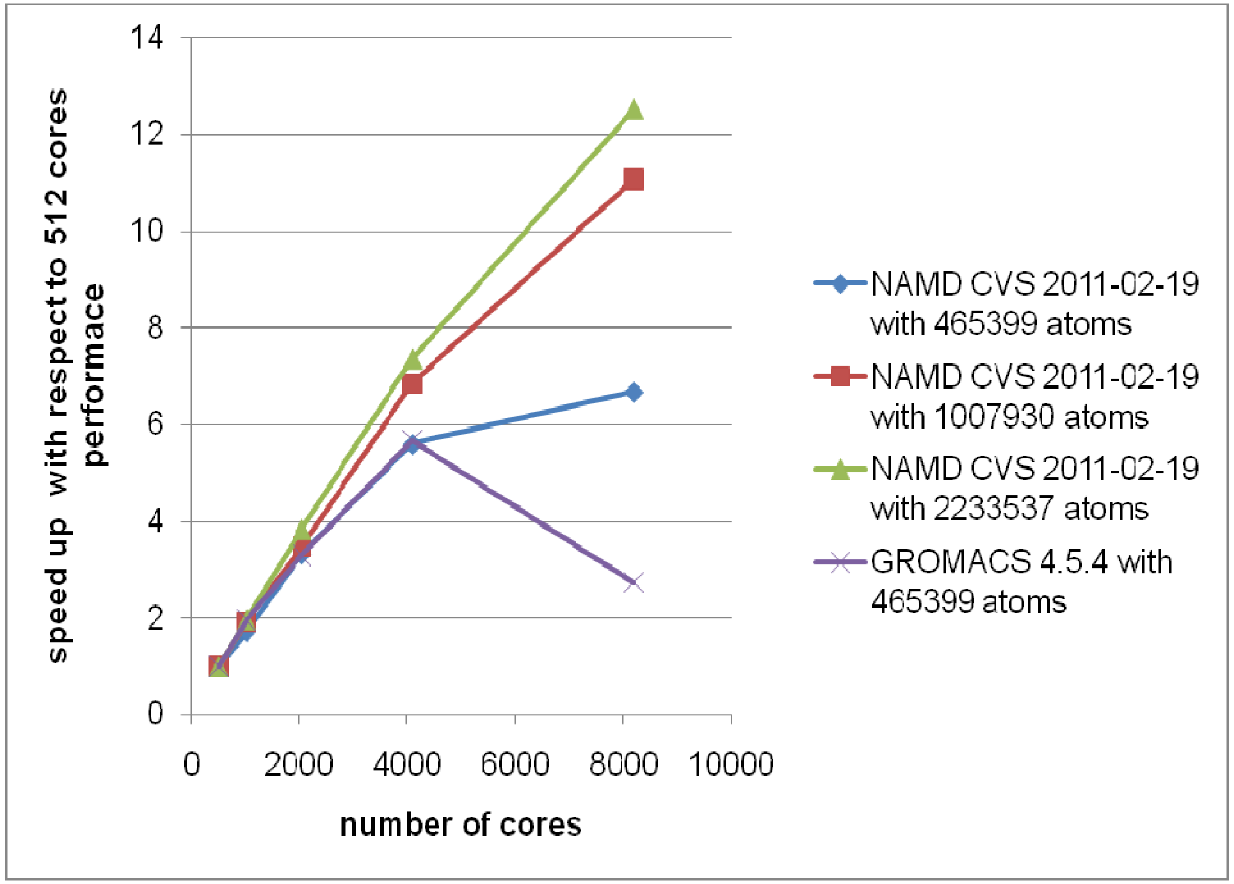
– Node-aware multicast – send message to communication thread on each node.

Hierarchical Load Balancing

– In previous load balancing schemes of NAMD, task 0 does load balancing for whole system. Instead divide total cores in load-balancing groups.

Similar modifications are being applied to GROMACS. For example the use of Global Arrays to distribute atoms among the tasks (PRACE whitepaper).

Make sure you are comparing apples with apples!



Final Conclusions

- *There are many features which affect performance but project proposals for computer time are judged mainly on the parallel scaling*
- *All modern MD programs use domain decomposition for parallelisation.*
- *Parallel scaling strongly influenced by system size due to*
  1. *limits of domain decomposition for non-bonded interactions*
  2. *all-to-all communication in FFT for electrostatics*

  The FFT is the more serious limitation.
- *Many "normal" systems do not scale upto thousands of cores. One workaround is to use "ensemble methods" (e.g. replica exchange, metadynamics). Or GPUs.*
- *Memory not normally a problem but becomes important for million atom systems.*
- *No obvious candidate for beating the scalability barrier*