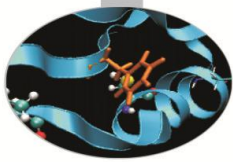
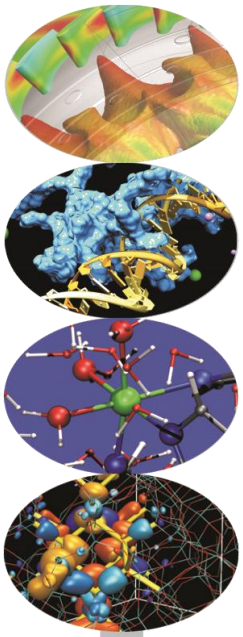
CINECA SCAI
SuperComputing Applications and Innovation



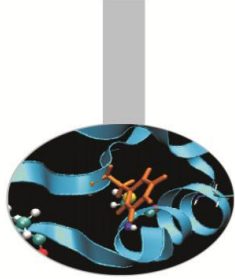
High Performance Molecular Dynamics



Andrew Emerson, Cineca
a.emerson@cineca.it

PRACE
CINECA

Agenda



Parallelisation of Molecular Dynamics

- Atom and Force Decomposition
- Domain Decomposition
- Parallelisation of the electrostatics calculation

Why do Molecular Dynamics programs stop scaling

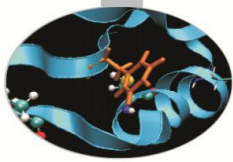
- Effects of system size and electrostatics
- Improving the performance and parallel scalability

Million atom simulations

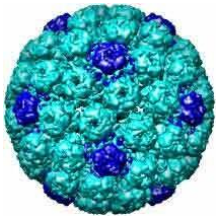
Using accelerators

Summary

Molecular Dynamics milestones

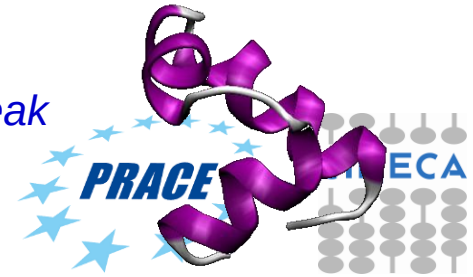


- **1959**: **First MD simulation (Alder and Wainwright)**
  - Hard spheres at constant velocity. 500 particles on IBM-704. Simulation time >2 weeks
- **1964: First MD of a continuous potential (A. Rahman)**
  - Lennard-Jones spheres (Argon), 864 particles on a CDC3600. 50,000 timesteps > 3 weeks
- **1977**: **First large biomolecule (McCammon, Gelin and Karplus).**
  - Bovine Pancreatic Trypsine inhibitor. 500 atoms, 9.2ps
- **1998**: First *μ*s 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 *μ*s of Villin Headpiece protein (34 residues).

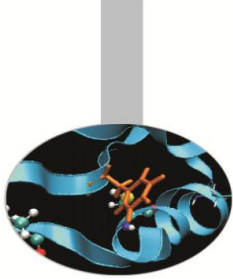


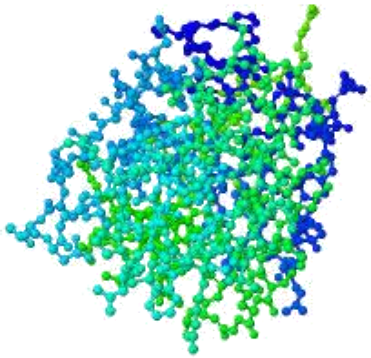
Folding@home distributed computing

folding@home equivalent to peak ~40 Pflops (Wikipedia)

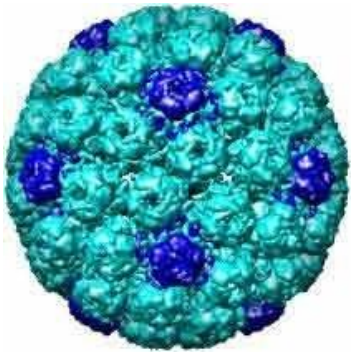


Biomolecular MD Simulation – system sizes



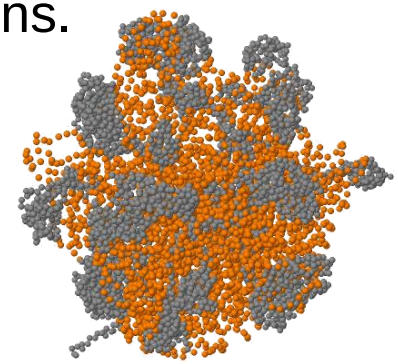


early 1990s. Lysozyme, 40k atoms

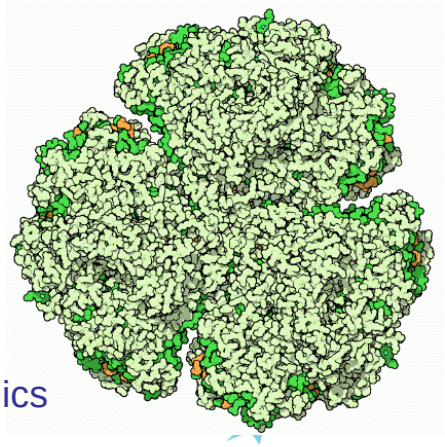


2006. Satellite tobacco mosaic virus (STMV). 1M atoms, 50ns

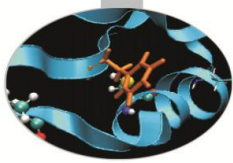
2008. Ribosome. 3.2M atoms, 230ns.



2011. Chromatophore, 100M atoms (SC 2011)



High Performance Molecular Dynamics



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

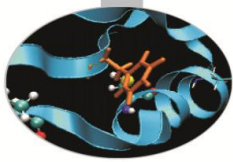
There are usually two types of non-bonded interactions:

1. Dispersion-type particle-particle interactions
2. Electrostatic interactions.

The dispersion interactions are normally solved with Lennard Jones (LJ) type potentials which can be truncated at short inter-particle separations.

Electrostatic interactions are commonly solve with the Particle Mesh Ewald (PME) Method or similar.

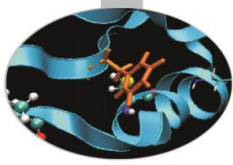
GROMACS timings



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

.....

Simple Molecular Dynamics program for neutral atoms

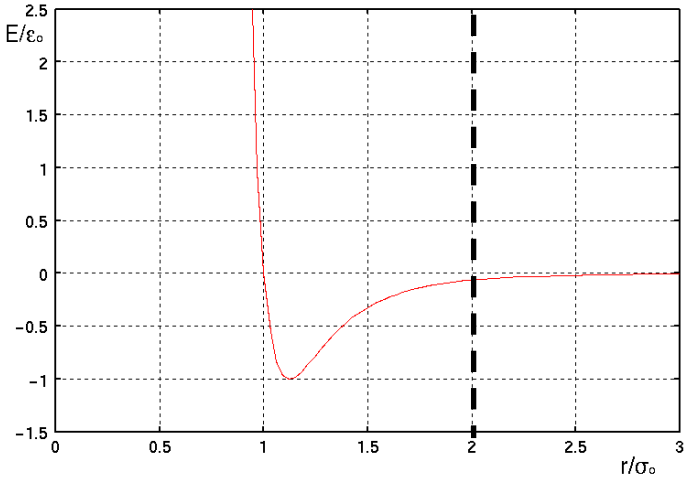


```
call init
T=0
do while (T.lt.Tmax)
   call compute_forces()
   call integrate_motion()
   call save_crds()
   call sample_averages()
   T = T + DT
enddo
call save_state()
stop
end
```

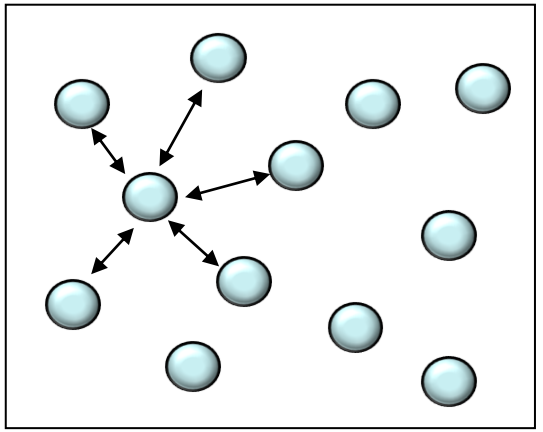
```
subroutine compute_energy_forces

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
```

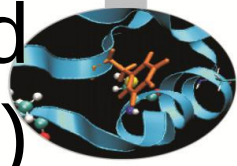
```
subroutine integrate_motion
do i=1,N
  r(i)=r(i)+verlet(F(i))
  v(i)=v(i)+verlet(F(i))
enddo
```



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



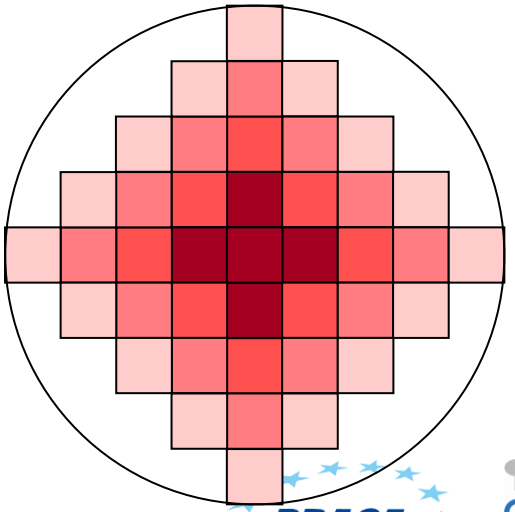
Electrostatic Interactions –Ewald Sum (1921)



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:

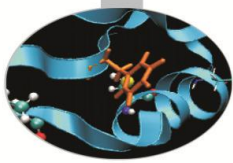
$$V = \frac{1}{2} \sum_{\mathbf{n}} \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{q_i q_j}{|\mathbf{r}_{ij} + \mathbf{n}|}$$

$\mathbf{n}=(n_x L, n_y L, n_z L)$



For large n the cell distribution is spherical

Electrostatic interactions – Ewald Sum

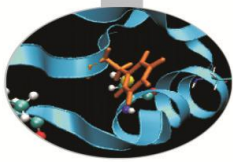


This pairwise summation converges slowly, but by assuming gaussian charge distributions around each charge it can be converted into faster converging real space (short range) and reciprocal space (long range) sums:

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).

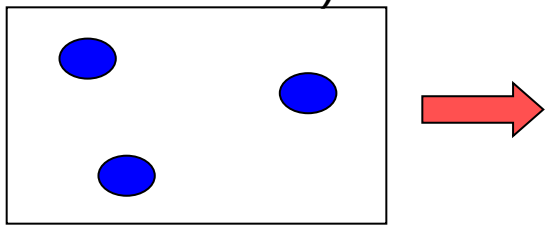
Particle Mesh Ewald



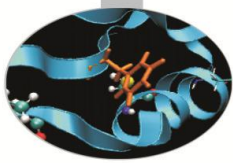
The second term converges quickly in reciprocal space but is computationally expensive:

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

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).



Parallelising the non-bonded, dispersion interactions

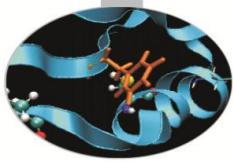


- In principle, molecular dynamics is a problem which is N^2 but fortunately it is also inherently parallel.
- Parallelisation schemes for the dispersion forces include:
  - Atom decomposition (aka replicated data)
  - Force decomposition
  - Domain decomposition
- These schemes are usually integrated with the serial optimisations such as neighbour lists or linked lists.

- Modern MD programs use the domain decomposition scheme in most cases (except for very small systems)

For the electrostatics with PME, parallelisation of the FFT is needed but this can be done with smaller code changes and dedicated libraries.

Atom decomposition algorithm

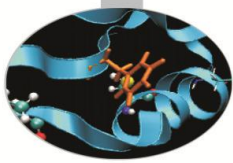


For every MD cycle:

- 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



force matrix

N

N

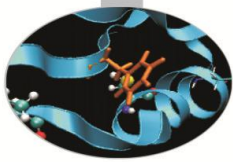
—	$-f_{21}$					
f_{21}	—					
		—				
			—			
				—		
					—	

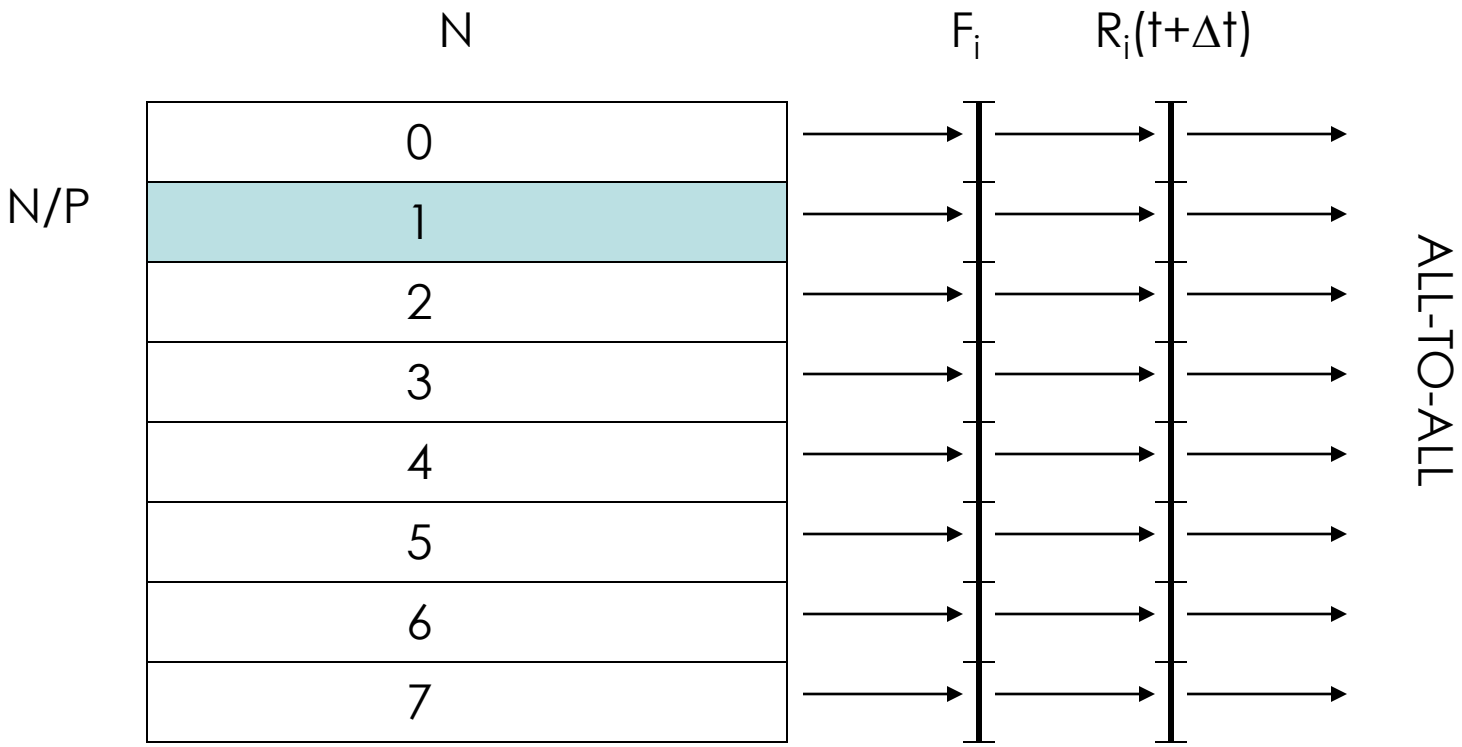
P_0

P_1

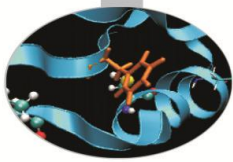
P_2

Atom decomposition





Atom decomposition - summary



Every processor must have all the positions from all the other processors such that they all have the same image in memory.

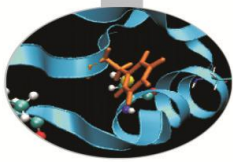
The MD force computation and integration is evenly shared across the processors -

- 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

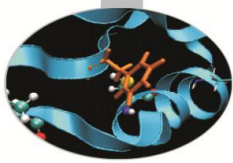


In this algorithm the parallelism is based on the block decomposition of the force matrix rather than on dividing up the atoms (which corresponds to the row-wise decomposition of the force matrix).

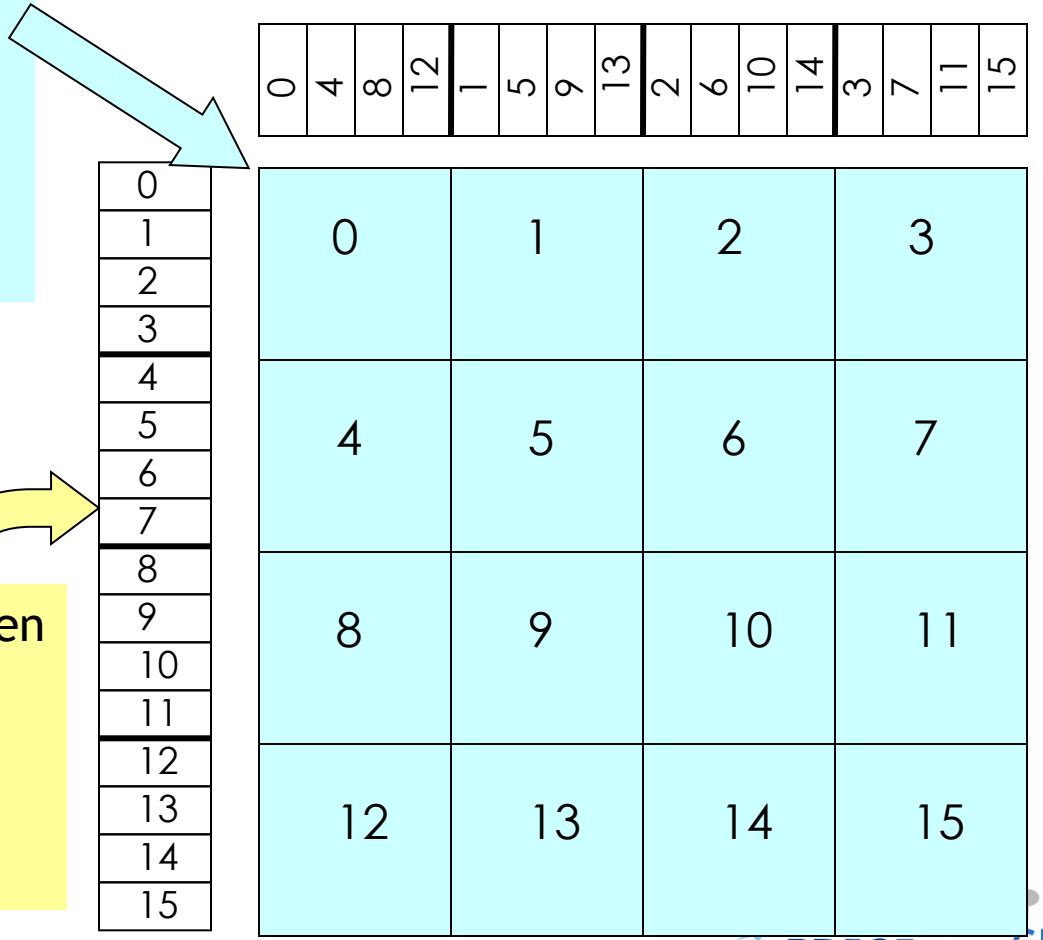
Processors are assigned to square areas of the force matrix, with size N^2/P. This reduces communication costs since communication is now between particular rows and columns, rather than over the whole matrix.

Communication costs for integrating the equations of motion can also be reduced by rearranging the columns in a particular way.

Force decomposition algorithm

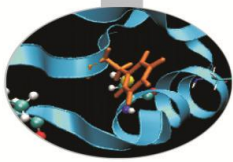


distribution of the work between 16 procs for calculating the forces



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

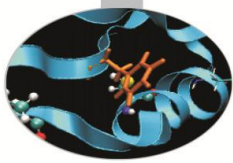
But less information is required by each processor O(N/√*P),* so lower memory and communication costs.

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

But both Atom and Force decomposition require significant 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

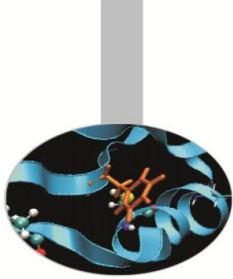


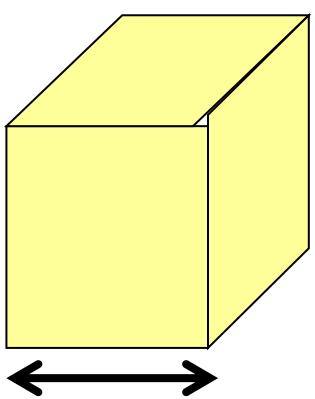
Here each processor is assigned to a spatial region of the simulation box (with side $r_d < r_c$) such that and stores only a portion of the whole system. This has two components:

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

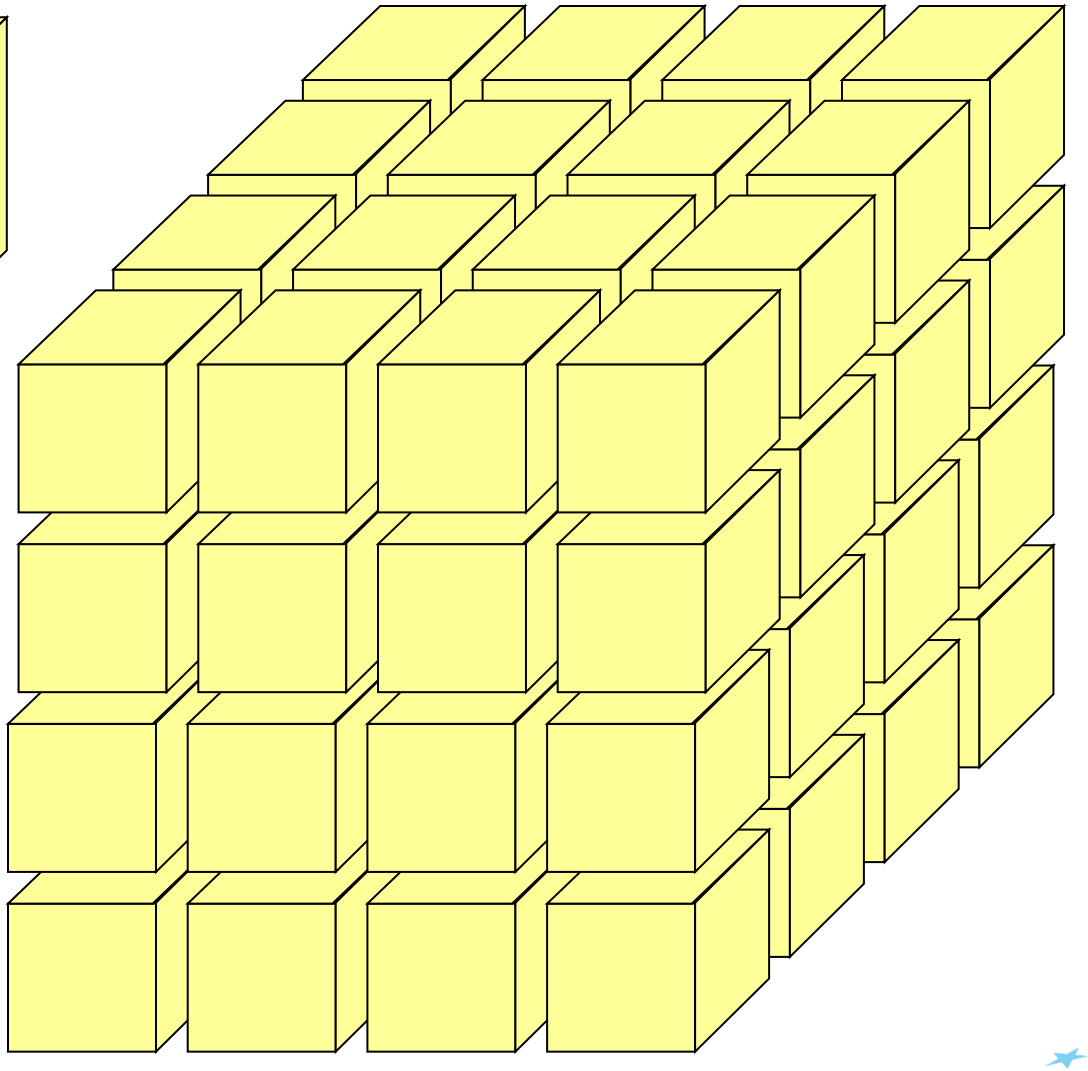
In order to minimise the surface with respect to the volume, and hence the communications, it is important to use regions that are as cubic as possible. In any case the communications are reduced since it is not necessary to update the whole system in local memory.

Domain decomposition



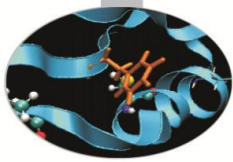


$r_d < r_c$

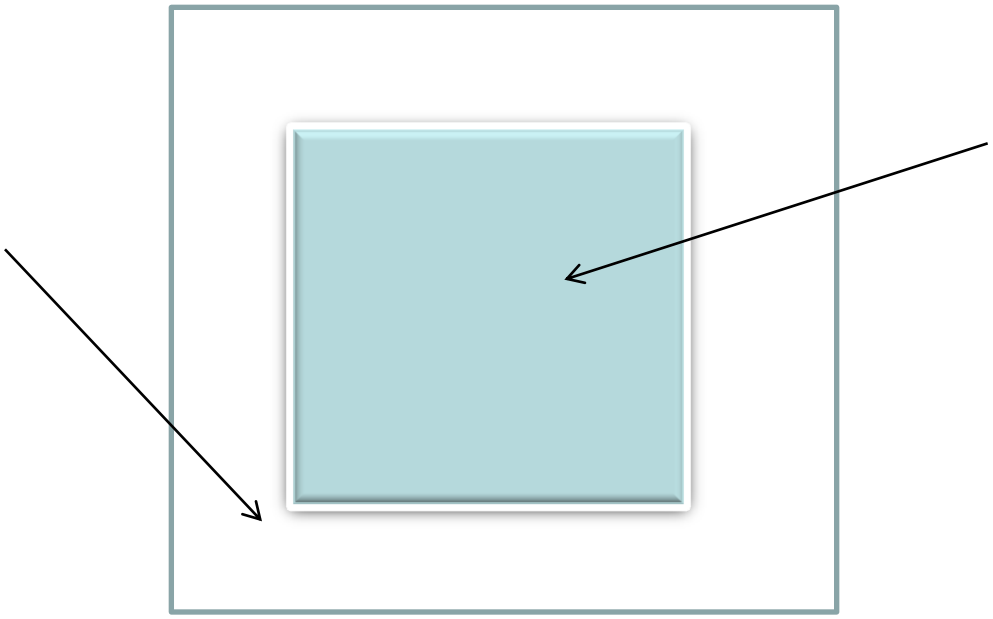


Must choose domain sides to be less than the cutoff

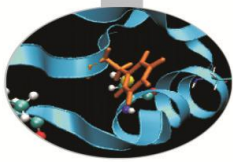
Domain decomposition



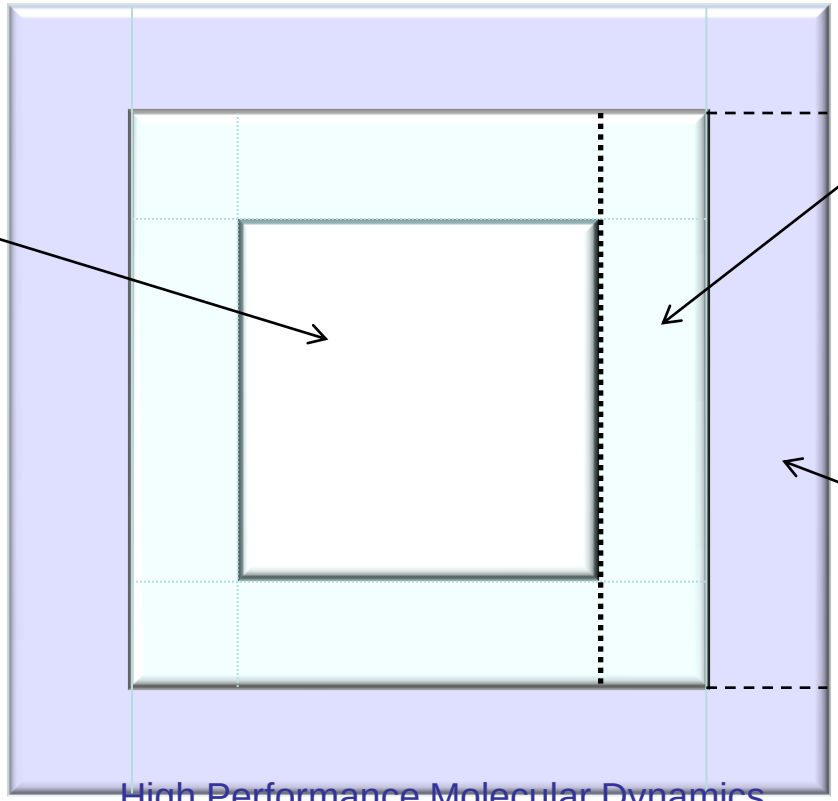
- Each domain will have 2 types of atoms:
  1. Those which can be entirely managed by the processor, i.e. all the interactions are within the domain
  2. Those for which the interactions *extend* outside the domain. Here data have to be sent/received to/from neighbouring domains.



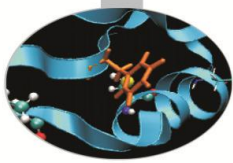
domain decomposition



- 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

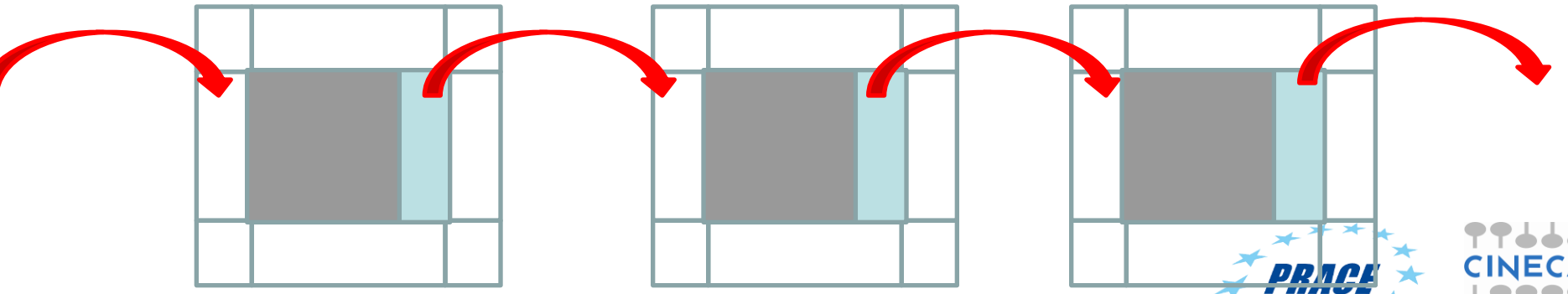


- neighbour coords in each dimension conveniently exchanged via **`mpi_cart_shift`** and **`mpi_sendrecv`** calls

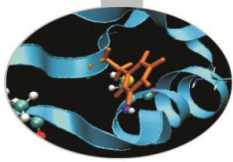
- First pass, x-direction, left to right

```
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
```

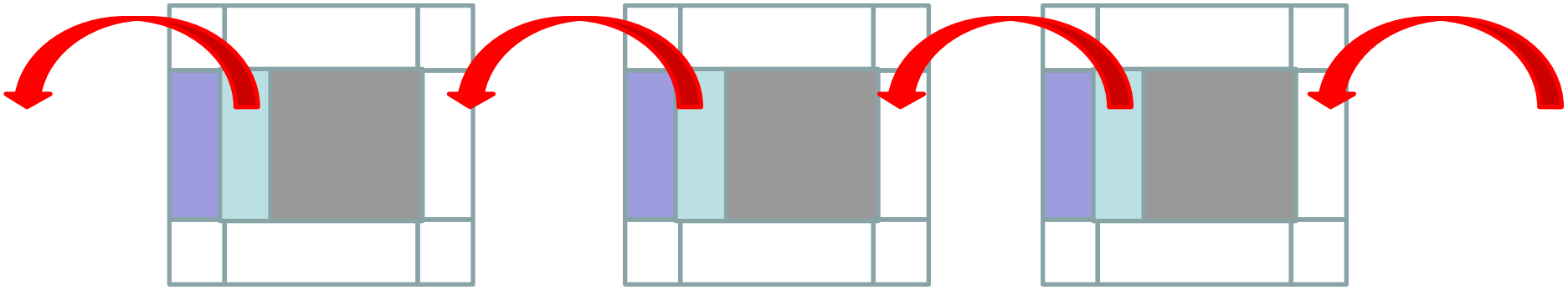


domain decomposition

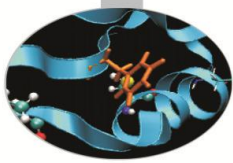


Then right to left

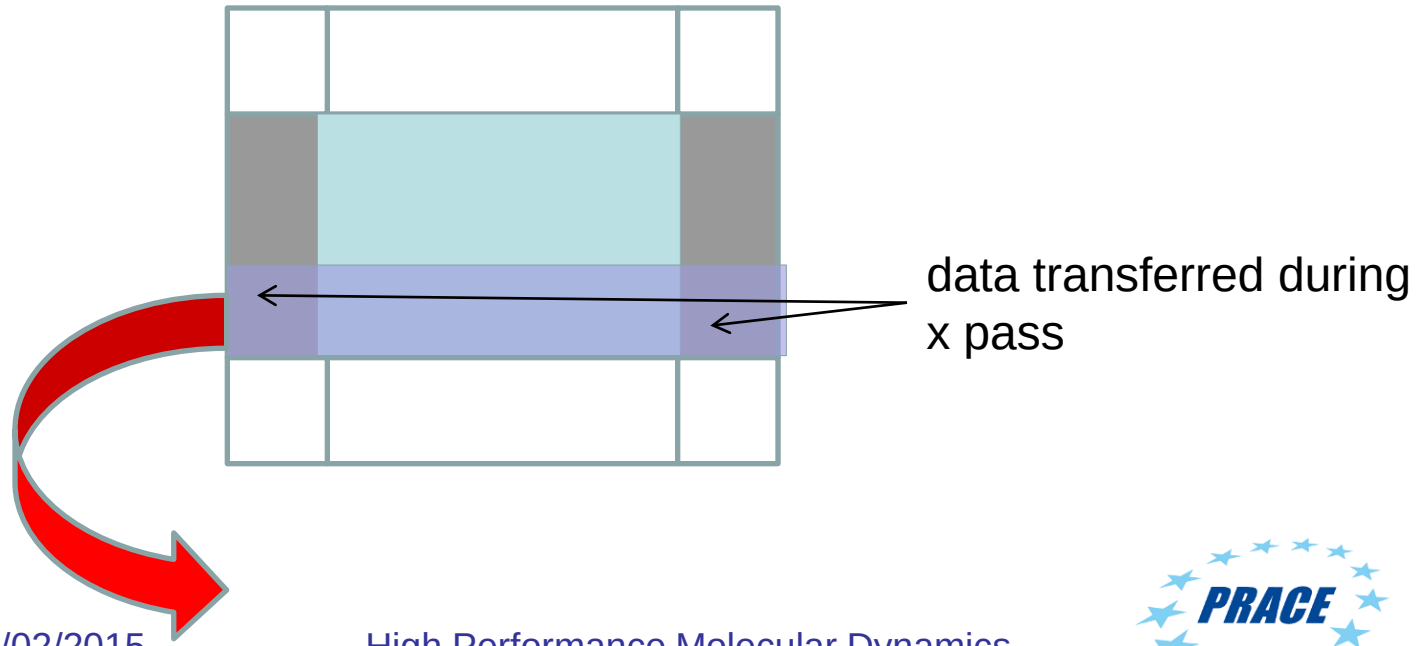
```
call mpi_sendrecv(left_side,nleft,MPI_REAL,proc_left,0,
halo_right,nright,MPI_REAL,proc_right,0,mpi_box,status,ierro
r)
```



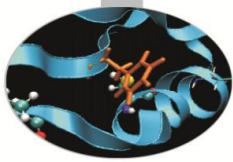
domain decomposition



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



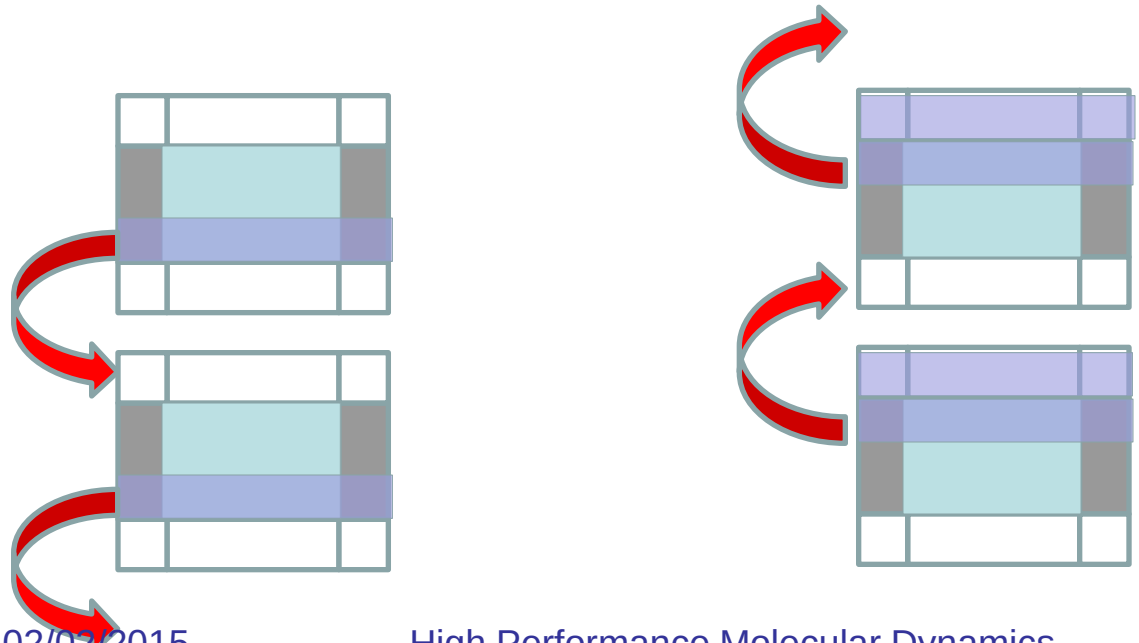
domain decomposition



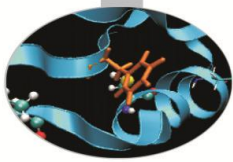
```
call mpi_cart_shift(mpi_box,0,1,proc_up,proc_down,ierror)

! 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

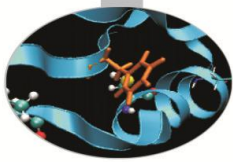


Similarly in the z direction, using data transferred in the previous y passes (which includes data transferred in x)

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



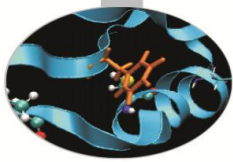
Advantages

- 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

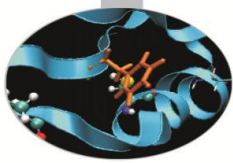
–Also relatively simple to code, although requires some pre-processing to avoid load balancing problems.

–Generally scales better than atom decomposition, but communication cost is still high $O(N/\sqrt{P})$

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

Novel domain-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

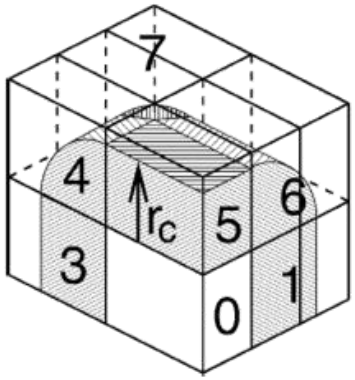
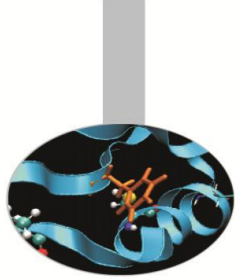


Figure 1. A nonstaggered domain decomposition grid of 3 × 2 × 2 cells. Coordinates in zones 1 to 7 are communicated to the corner cell that has its home particles in zone 0. r_c is the cutoff radius.

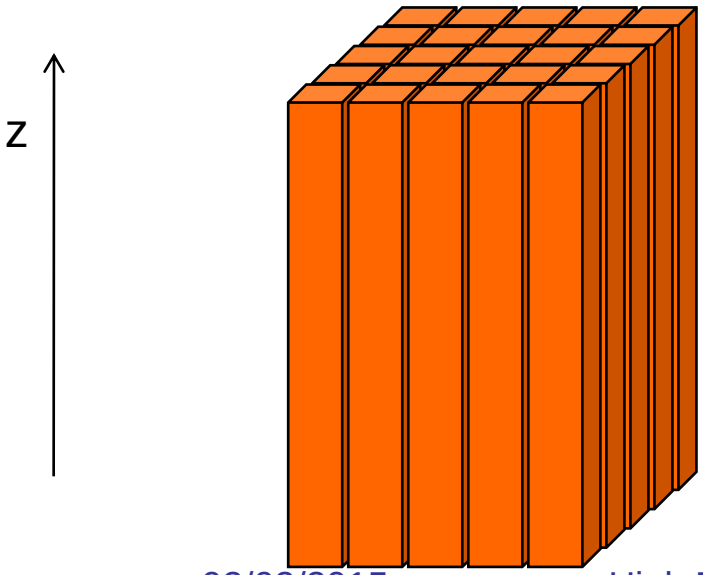
Hess et al., J. Chem, Theory Comput. C, 2007

Parallelisation of Electrostatics with Domain Decomposition and PME



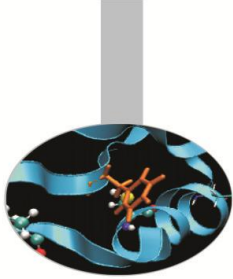
PME can be parallelised with a DD scheme but 3D FFT is very inefficient for many processors (or small N) because of all-to-all global communications (MPI_AlltoAll).

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?

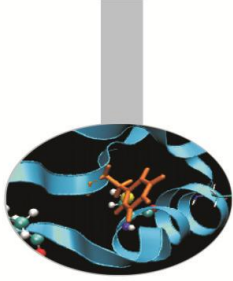


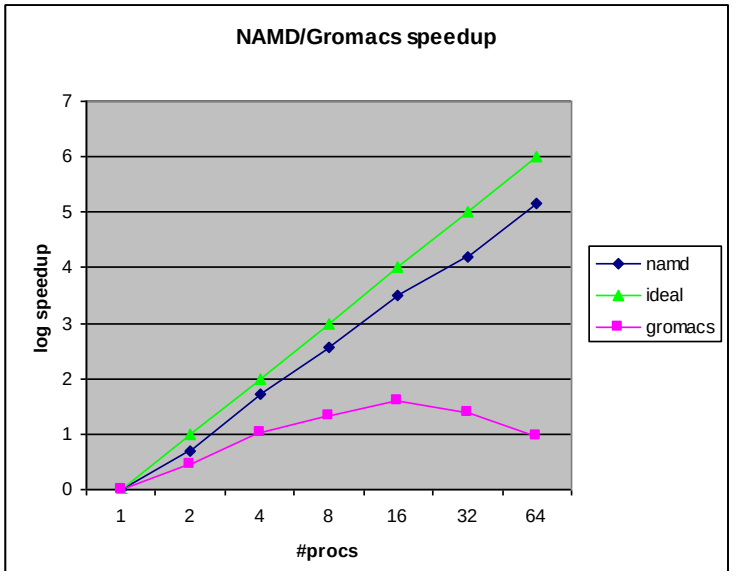
Compare

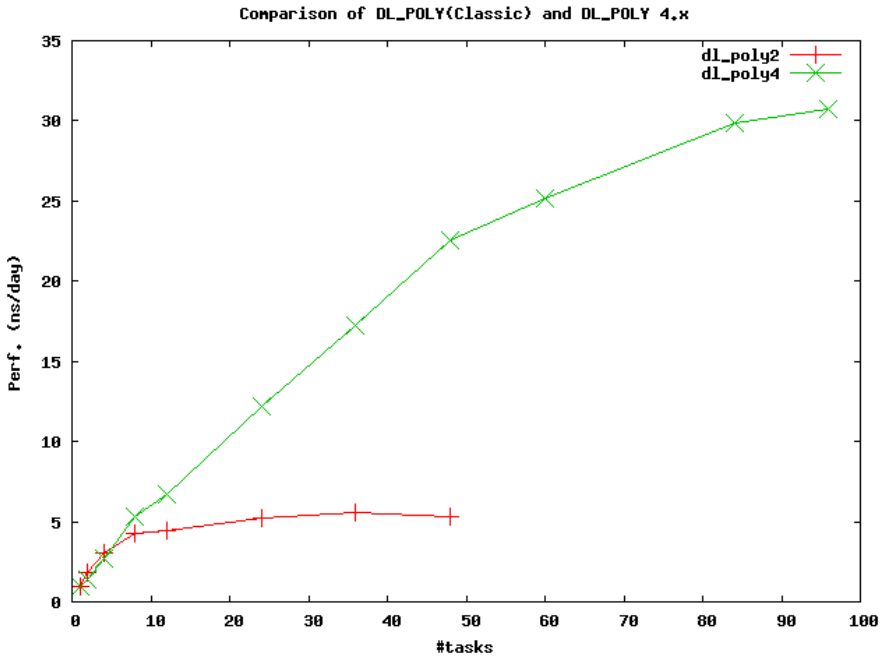
- GROMACS v 3.x and earlier with force-decomposition schemes
- GROMACS v 4.x with domain decomposition
- NAMD with domain decomposition

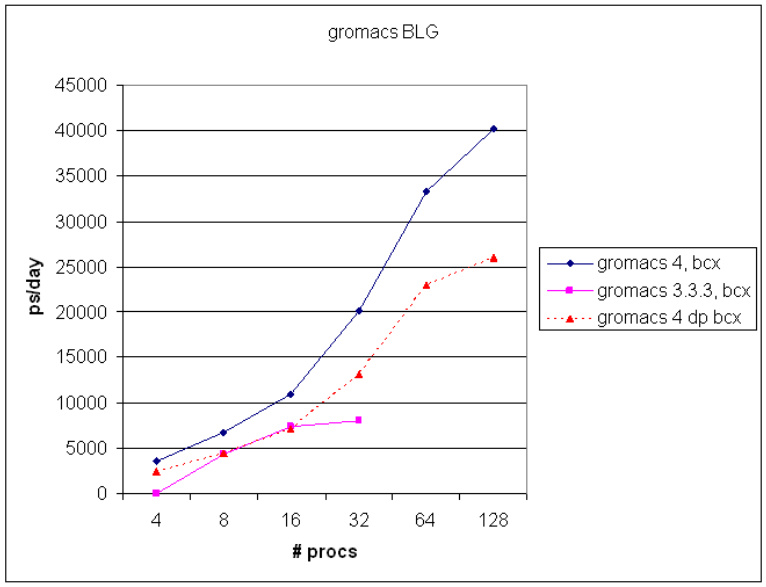
***Disclaimer**: There are many other differences between programs which could affect performance but parallel scaling is a good indicator of the parallelization scheme.*

Does domain decomposition work?





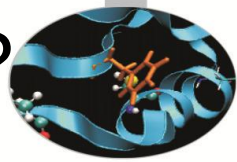




Simulation of 280K atoms of liquid argon with DL_POLY (Classic) and DL_POLY 4.03

rformance Molecular Dynamics

Why do MD (programs stop scaling ?



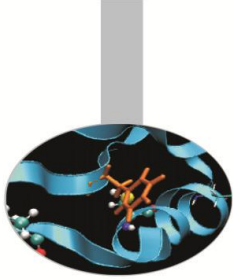
For most parallel programs the scaling levels out when the time of communications > time needed for calculations.

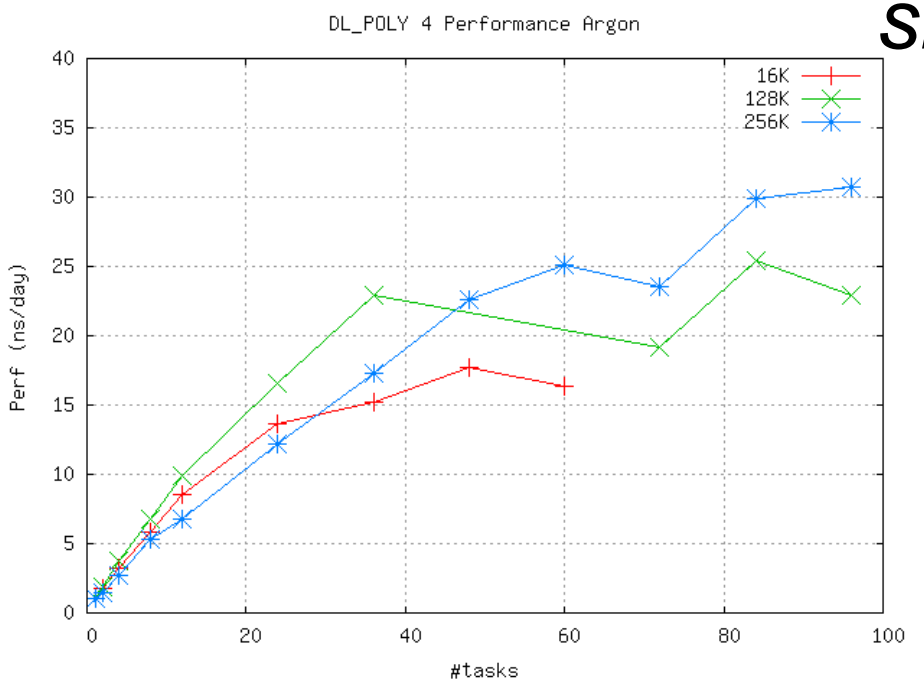
For modern molecular dynamics programs this can happen when system is too small compared to the number of cores -

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 .

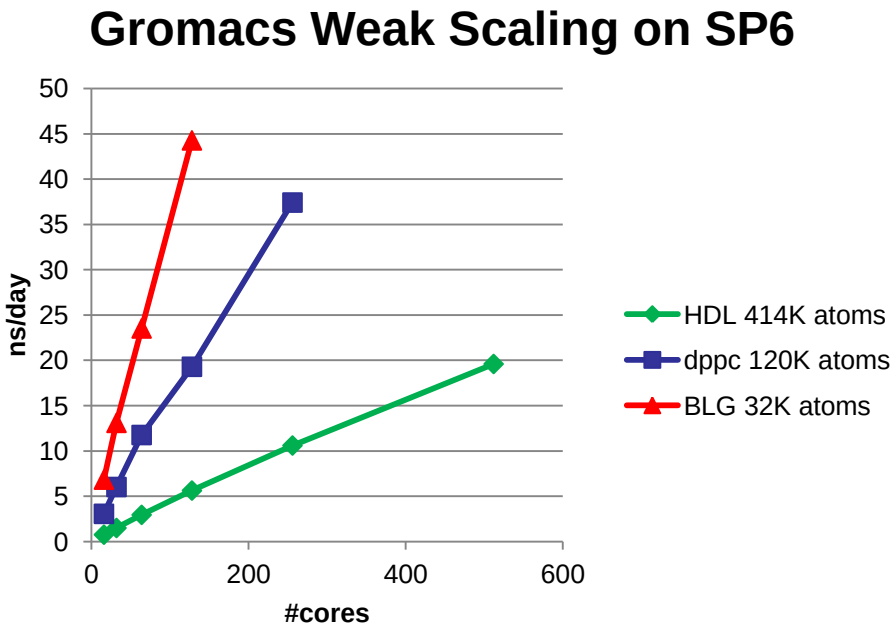
Scalability is generally higher in simulations without PME (for example with an electrostatic cutoff or implicit solvent) so this is usually the bottleneck to strong scaling.
As a rule of thumb, 100-200 atoms/core is going to be close to the scaling limit (i.e. max performance).

Scaling as a function of system *size ("weak scaling")*

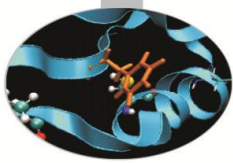




DL_POLY 4 simulation of liquid Argon as a function of system size (no electrostatics).



Implicit and Explicit solvents



The influence of PME on parallel scaling can be tested by using implicit solvent models which model the solvent as a continuous medium instead of interacting particles, but for many biological environments (interiors of proteins or membranes) it is considered too approximate.

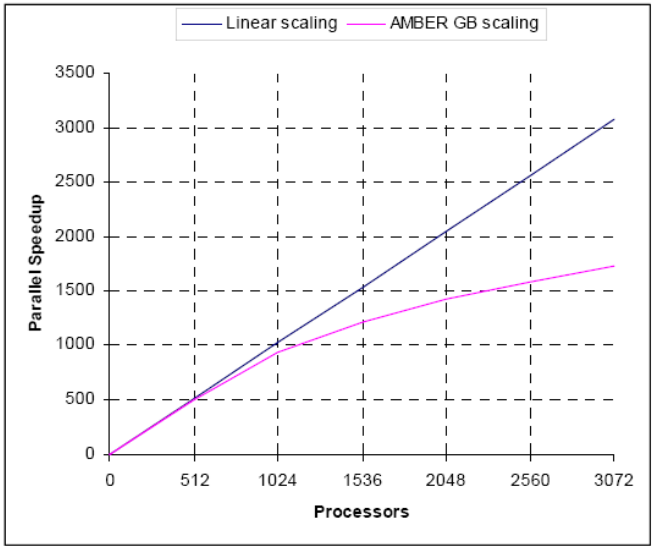


Figure 1. Parallel scaling of AMBER on Blue Gene. The experiment is with an implicit solvent (GB) model of 120,000 atoms (Aon benchmark).

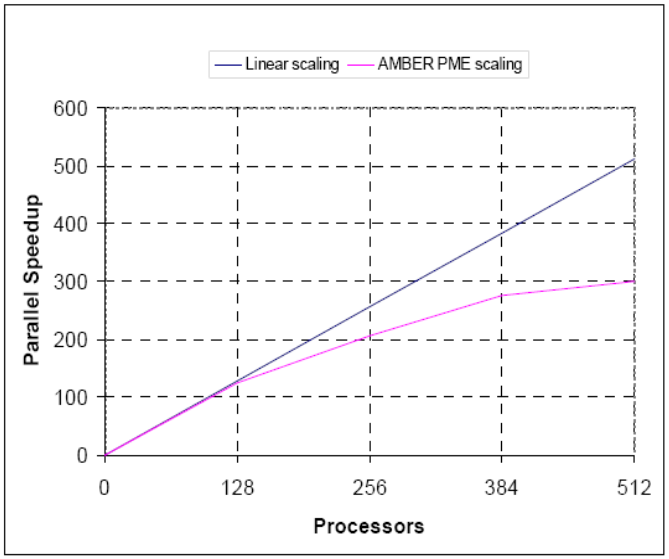
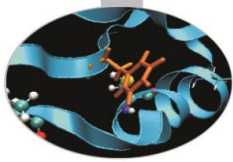
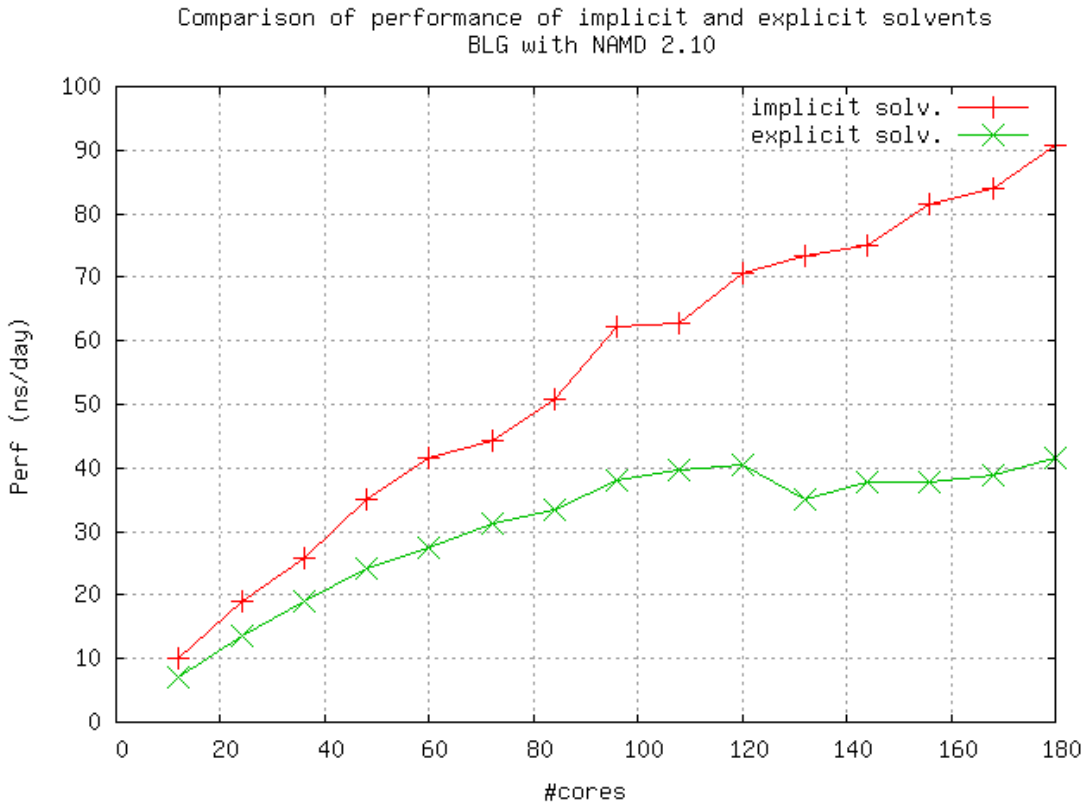


Figure 2. Parallel scaling of AMBER on Blue Gene. The experiment is with an explicit solvent (PME) model of 290,000 atoms (Rubisco).

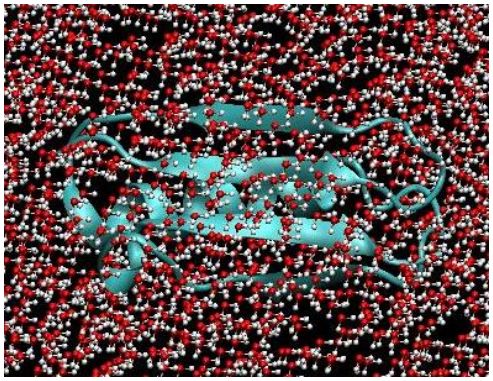
***Life Sciences Molecular Dynamics Applications on the IBM System Blue Gene Solution: Performance Overview*, http://www-03.ibm.com/systems/resources/systems_deepcomputing_pdf_lsmdabg.pdf**

Implicit and Explicit solvents





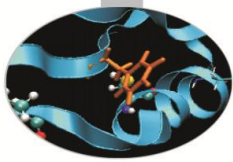
NAMD 2.10
Beta-lactoglobulin
in explicit and
implicit solvents



$$G_s = \frac{1}{8\pi}\left(\frac{1}{\varepsilon_0} - \frac{1}{\varepsilon}\right)\sum_{i,j}^{N}\frac{q_i q_j}{f_{GB}}$$

Generalized Born Equation

Scaling limits for HPC projects

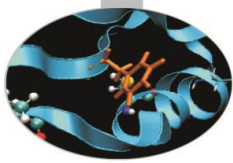


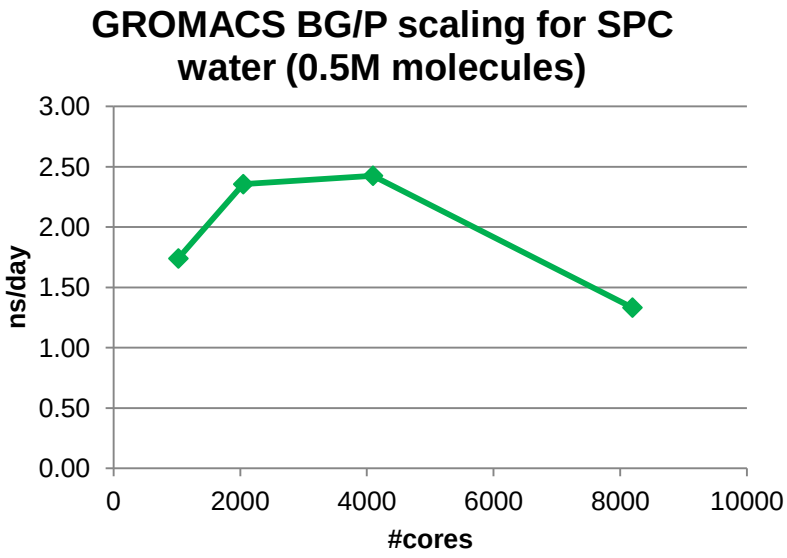
The Bluegene and other multi-thousand core architectures represent a challenge for projects based on molecular dynamics since often a minimum scaling is required.

Computer System	Minimum Parallel Scaling	Max memory/core (Gb)
Curie	Fat Nodes 128 Thin Nodes 512 Hybrid 32	4 4 3
Fermi	2048 (but typically >=4096)	1
SuperMUC	512 (typically >=2048)	*
Hornet	2048	*
Mare Nostrum	1024	2Gb

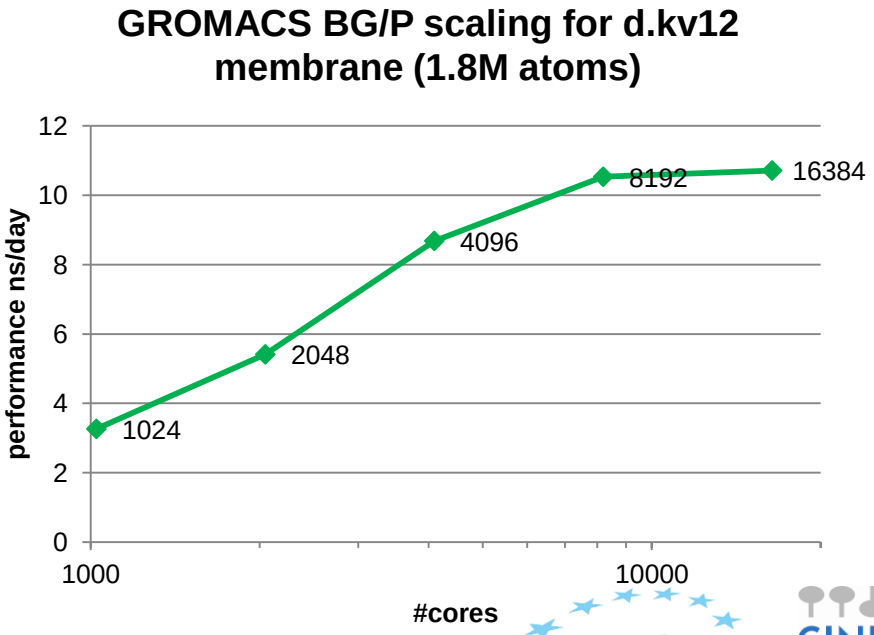
PRACE Tier-0 parallel scaling requirements in 2013

Why do MD programs stop scaling?

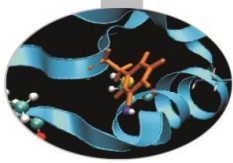




For this benchmark we had to duplicate the std GROMACS benchmark d.kv12 ion channel 16 times !

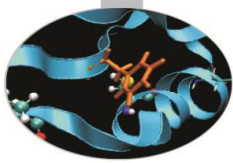


How can I increase the parallel scaling ?



1. Reduce the communications in the PME calculations. (e.g. GROMACS)
2. Try exploiting threads with hybrid MPI/OpenMP .
3. Increase the system size.
   - *But not always possible if your problem size is "fixed" (i.e.* because you are studying a particular molecule)

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



Reducing the PME cost - 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.

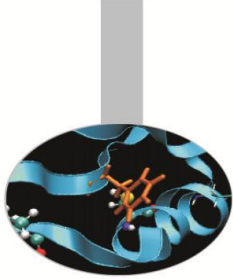
GROMACS 4.x allows separate nodes to be assigned to PME calculations:

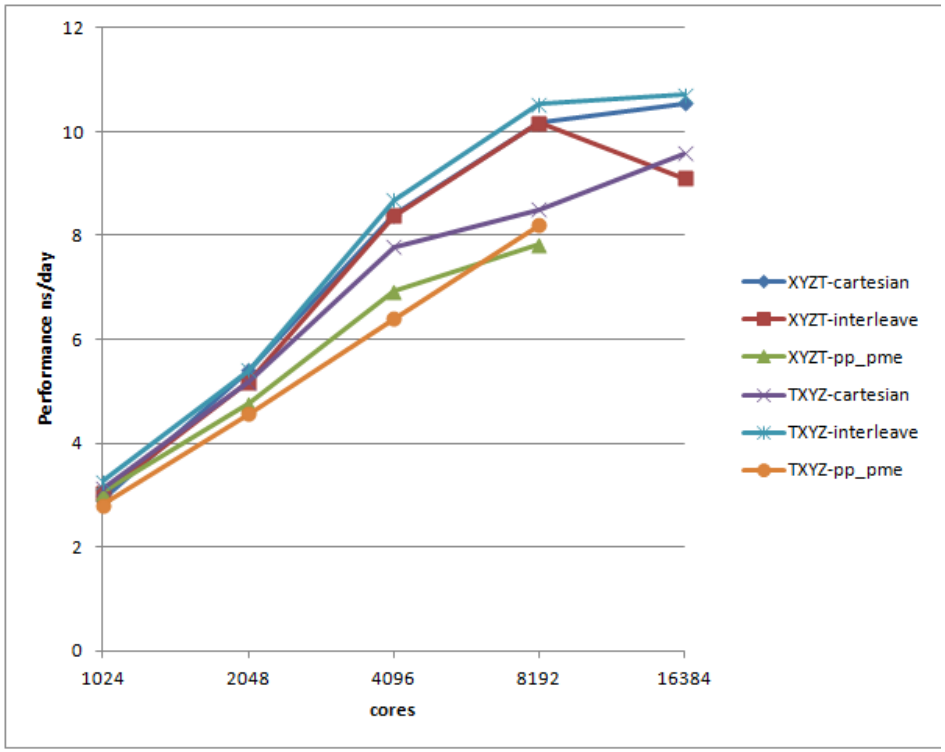
```
mpirun  mdrun -npme 4 md.conf
```

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

Also possible to change how the PME and PP nodes are partitioned with the –ddorder option of **mdrun**.

Reducing the PME cost: 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 BG/P.

http://www.prace-ri.eu/IMG/pdf/Performance_Analysis_and_Petascaling_Enabling_of_GROMACS.pdf

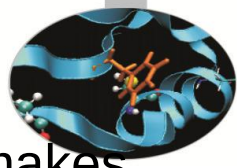
PP	PP	PP	PME
PP	PP	PP	PME

Cartesian, 2 PME nodes

PP	PME	PP	PME
PP	PME	PP	PME

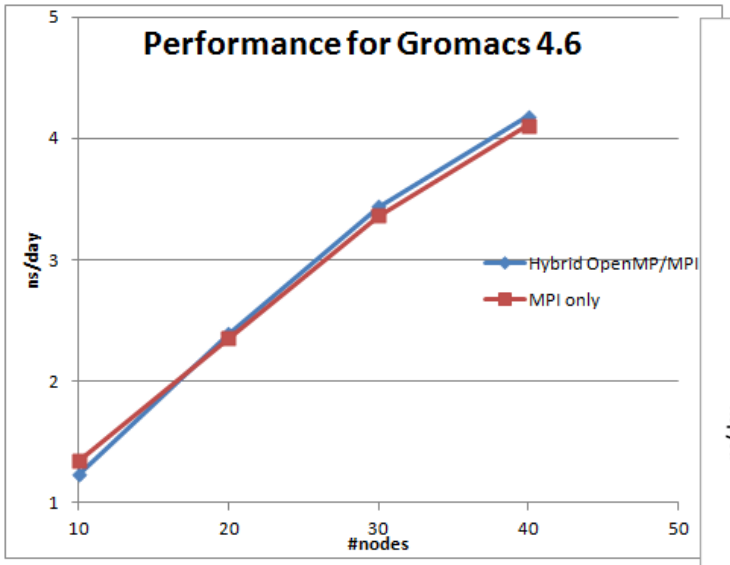
Interleave, 4 PME nodes

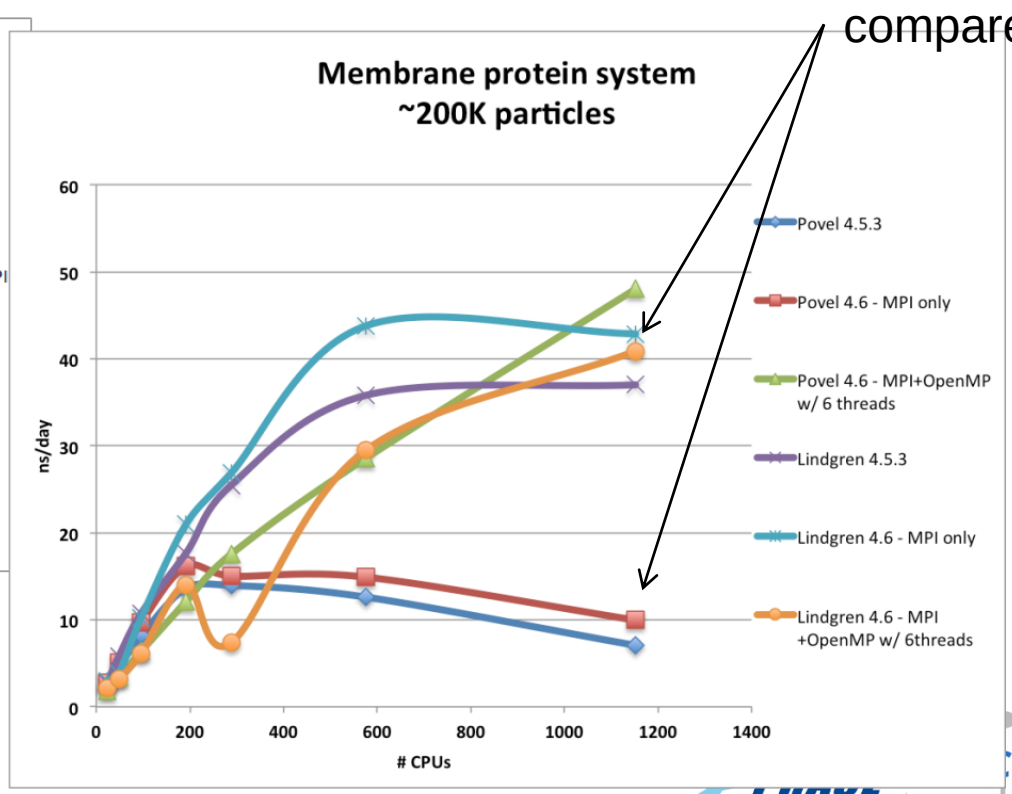
Hybrid MPI/OpenMP



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

http://www.prace-ri.eu/IMG/pdf/Performance_Analysis_and_Petascaling_Enabling_of_GROMACS.pdf

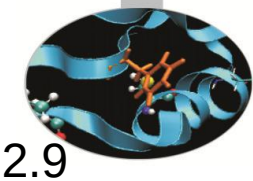




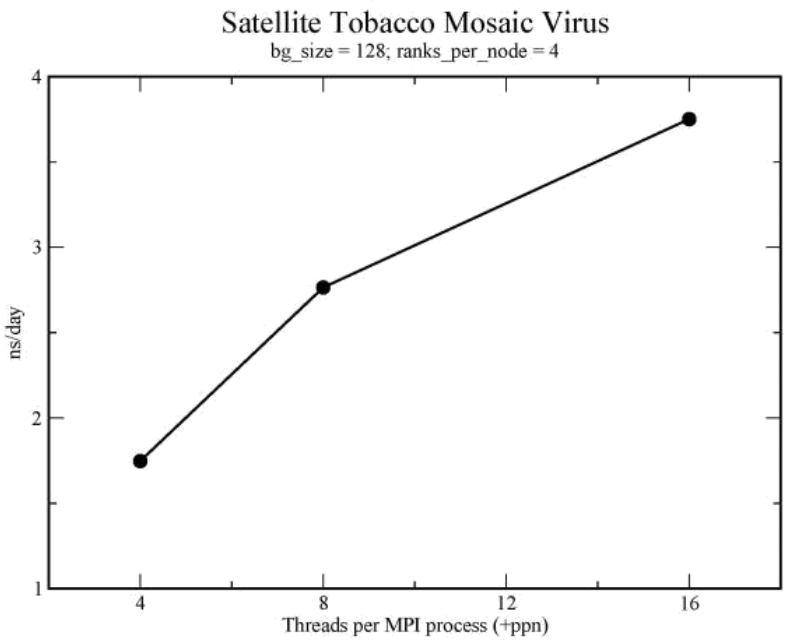
compare

PLX – fast network, few *nodes → no difference*

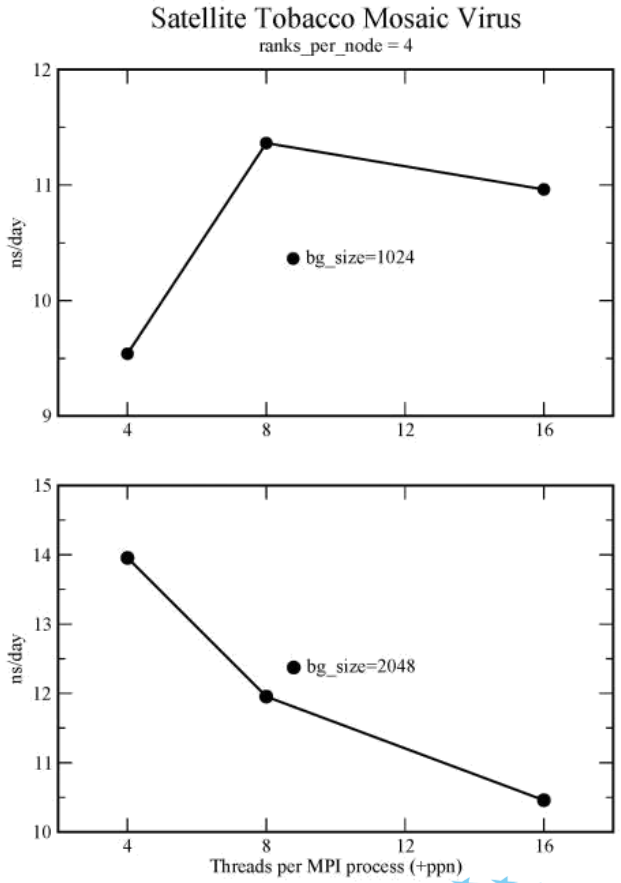
Hybrid MPI/OpenMP - NAMD



Small, but significant improvements obtained with threaded version of NAMD 2.9



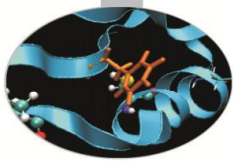
bg_size=128, ranks/node=4 (512 tasks)



http://www.hpc.cineca.it/content/namd-benchmark

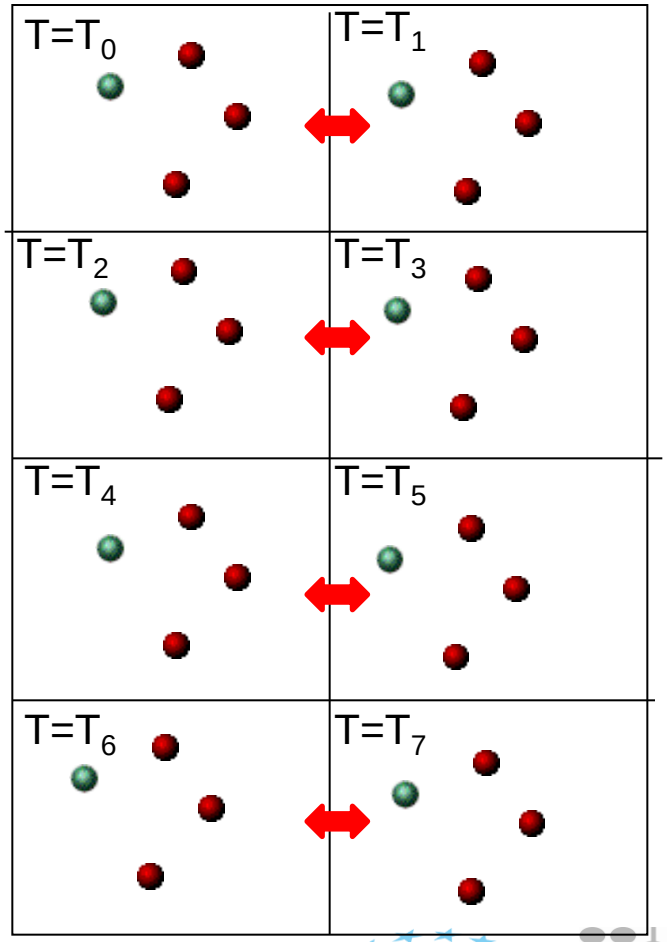
PRACE CINECA

Replica Exchange Molecular Dynamics

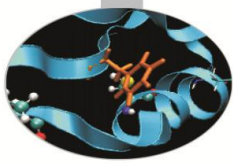


Replica Exchange Molecular Dynamics

- 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.

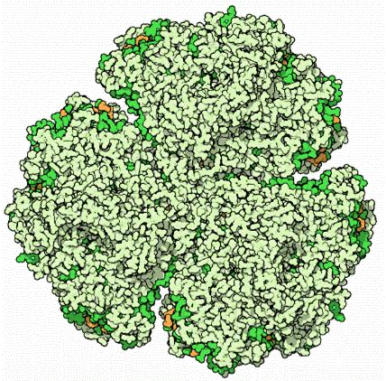


Million atom simulations- NAMD

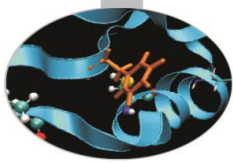


Very large systems (100 M atoms) pose challenges for conventional MD programs:

- 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

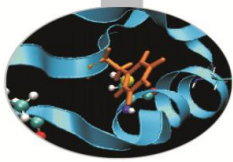


Million atom simulations- NAMD

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

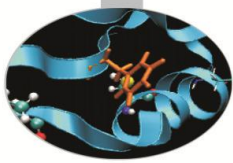
Million atom simulations- NAMD



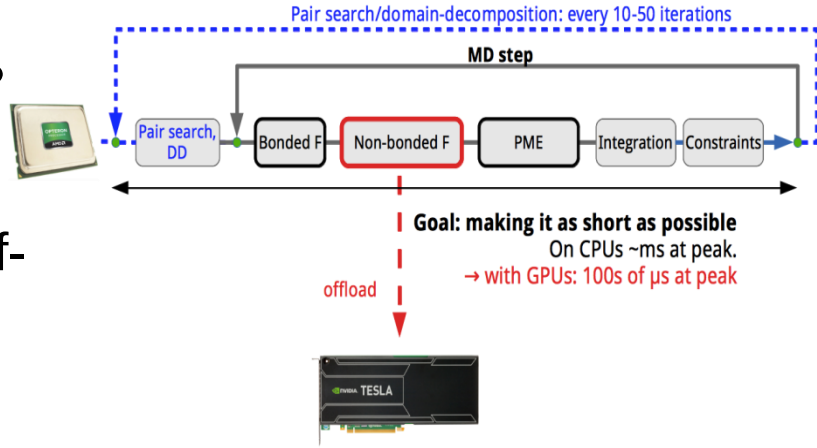
- Communication Optimization
  - Node-aware multicast – send message to communication thread on each node.
- Hierarchical Load Balancing
  - In previous load balancing schemes of NAMD, task 0 does load balancing for whole system. Instead divide total cores in load-balancing groups.

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

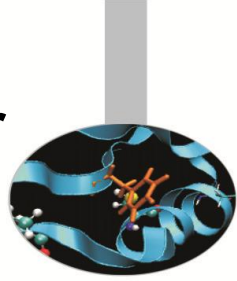
Molecular Dynamics and accelerators

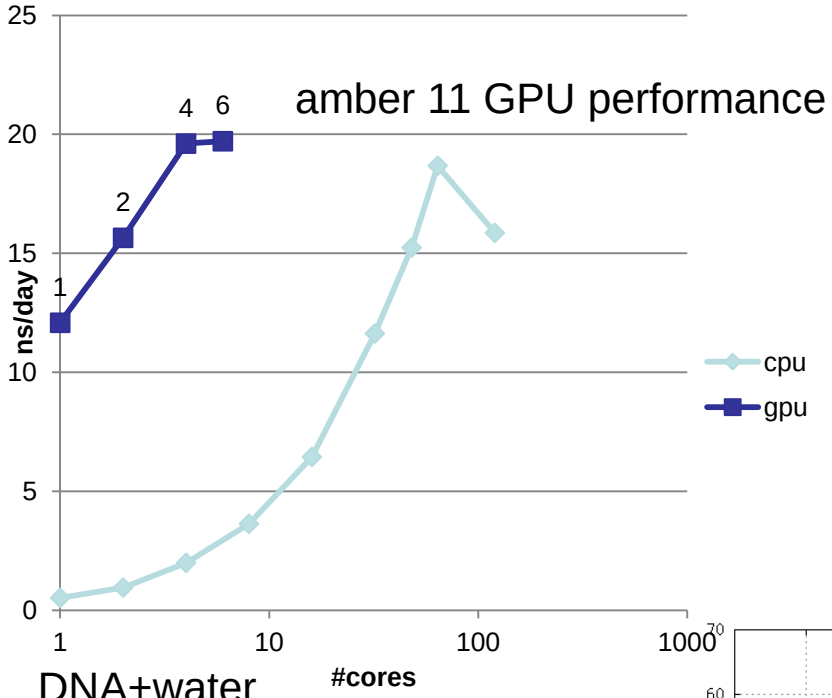


- If we cannot increase the parallelism, how can we increase performance *assuming Moore's law no longer valid?*
- Most of the common MD applications have GPU/CUDA-enabled versions which accelerate the calculations by off-loading the expensive, non-bonded calculations to the GPU.
- Particular effort with Amber with GPU-enabled port giving large speedups (tens of times in some cases) compared to non-accelerated codes.
- But reasonable speed-ups of 2-3x also for NAMD, GROMACS, etc.
- Sometimes maximum performance not affected significantly – main advantage is to obtain performance using fewer nodes.

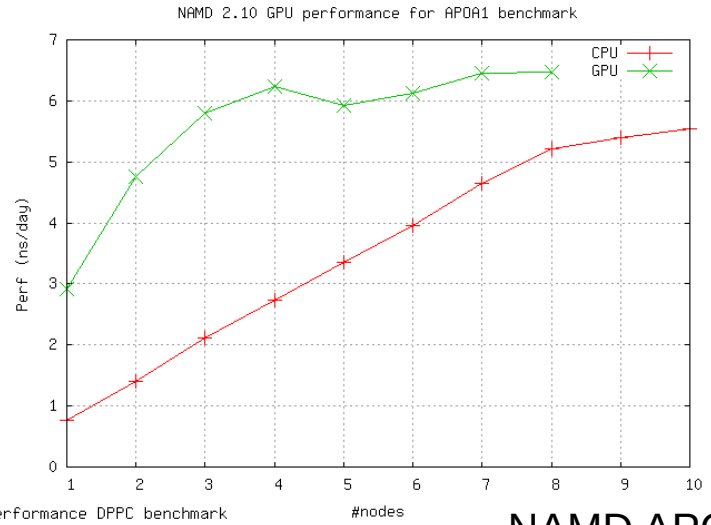


"Accelerated" Molecular Dynamics - GPU

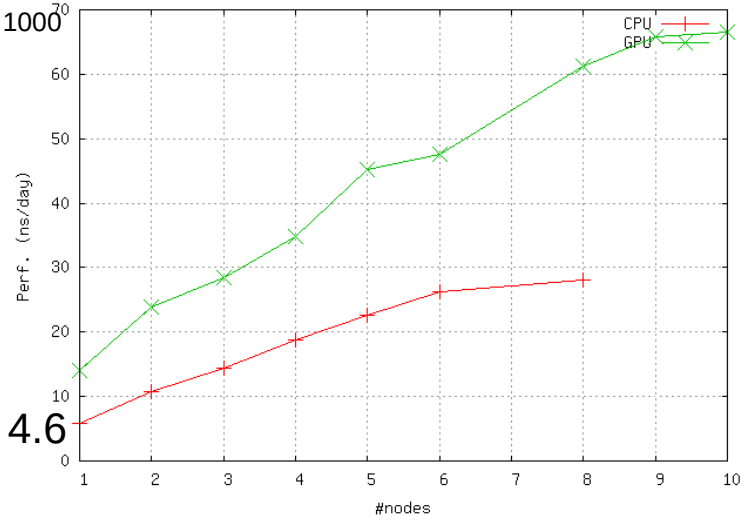




DNA+water
(40K atoms)



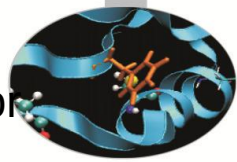
NAMD APOA1



GROMACS 4.6
DPPC

It is argued that poorly optimised un-accelerated codes give best speed-ups.

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 free energy calculations).
- Most MD codes offer GPU-versions which can get good performance for fewer resources, but do not increase by orders of magnitude the maximum performances. Xeon PHI code versions starting to appear but still at an early stage.
- Memory and I/O not normally problems but become important for million atom systems.
- No obvious candidate for beating the scalability barrier. Some interest in the use of Fast Multipole Methods for long-range forces but still very much in the research phase.