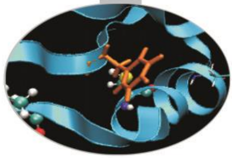
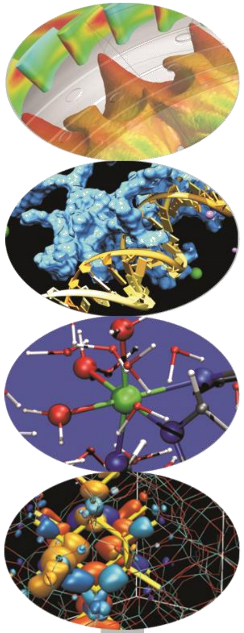
SuperComputing Applications and Innovation



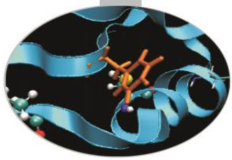
High Performance Molecular Dynamics



Parallelism and Parallel algorithms

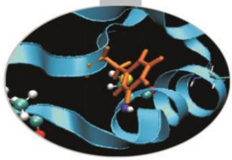
Andrew Emerson (a.emerson@cineca.it)

Agenda

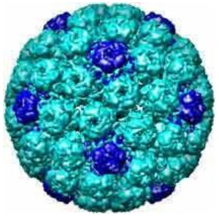


1. Molecular Dynamics milestones
2. Anatomy of a serial Molecular Dynamics program
3. Concepts of Parallelism
4. Parallel algorithms and scaling limits

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).

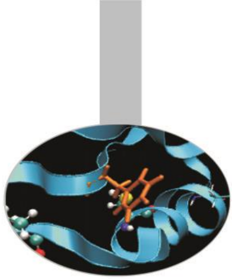


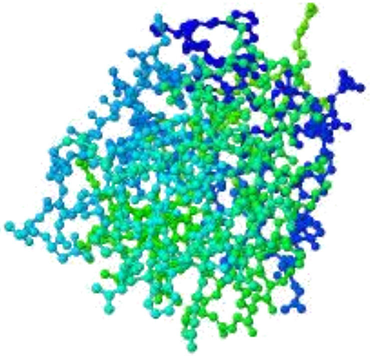
Folding@home distributed computing

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

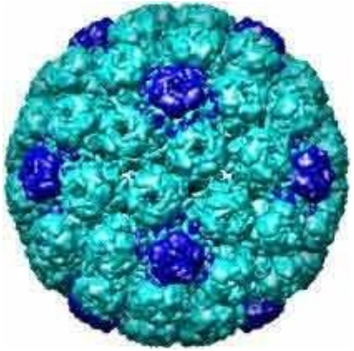
PRACE CINECA

Biomolecular MD Simulation – system sizes



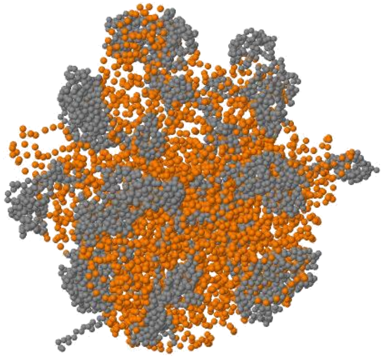


early 1990s. Lysozyme, 40k atoms

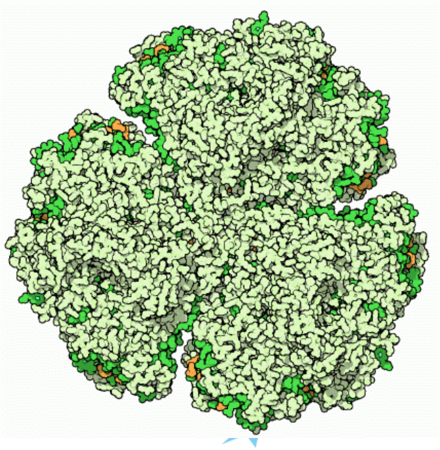


2006. Satellite tobacco mosaic virus (STMV). 1M atoms, 50ns

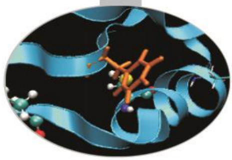
2008. Ribosome. 3.2M atoms, 230ns.

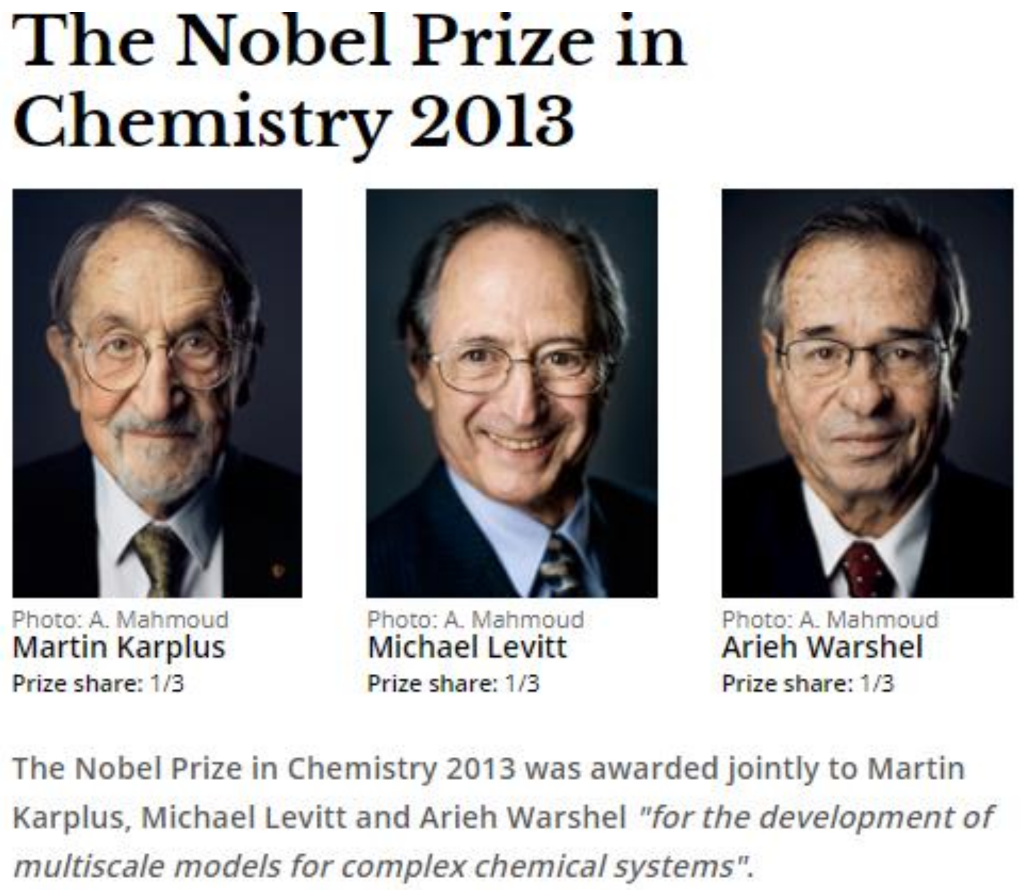


2011. Chromatophore, 100M atoms (SC 2011)



Nobel Prize for Karplus, Levitt and Warshel





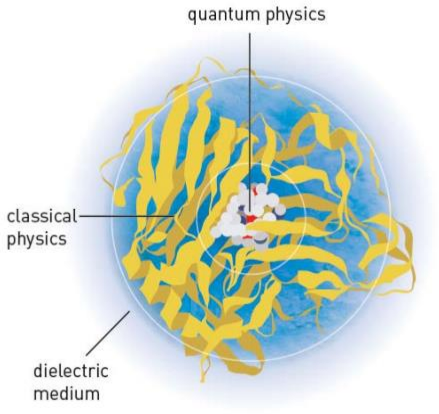
The Nobel Prize in Chemistry 2013

Photo: A. Mahmoud
Martin Karplus
Prize share: 1/3

Photo: A. Mahmoud
Michael Levitt
Prize share: 1/3

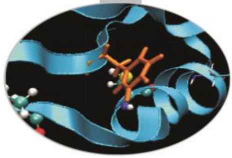
Photo: A. Mahmoud
Arieh Warshel
Prize share: 1/3

The Nobel Prize in Chemistry 2013 was awarded jointly to Martin Karplus, Michael Levitt and Arieh Warshel *"for the development of multiscale models for complex chemical systems"*.



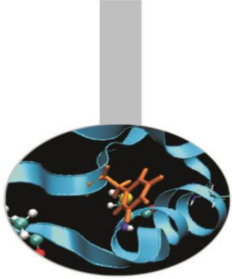
Taken from, “Scientific background on the Nobel Prize in Chemistry 2013”, *www.nobleprize.org*

Anatomy of a program



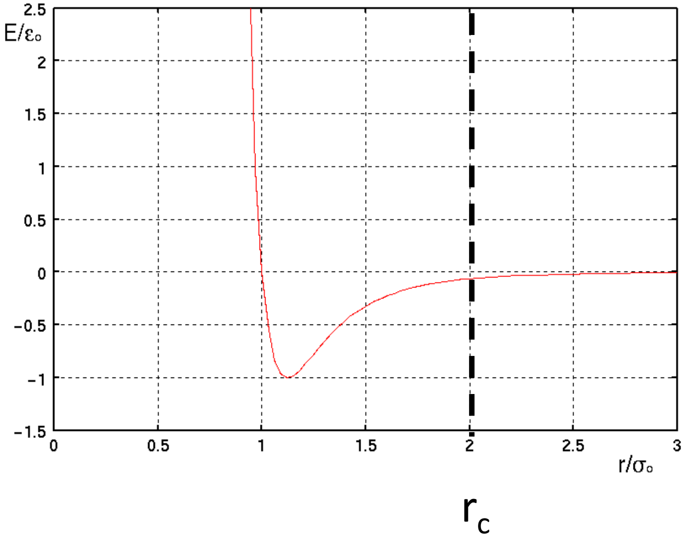
1. Read in parameters to control the simulation (e.g. run time, temperature, etc).
2. Generate or read in atomic coordinates and connectivity information. If starting from a previous run read in velocities, forces and other system data.
3. Start Main loop at time t.
    1. Compute forces between interacting atoms.
    2. Integrate forces to obtain velocities and positions at new time step t+Δt.
    3. Calculate thermodynamic properties (e.g. Temp, Pressure,etc).
    4. At intervals store configuration for trajectory and restart information.
    5. If t < required time loop back to step 1.
4. Output final configuration, thermodynamic and perhaps timing data.

Simple Molecular Dynamics program for neutral atoms

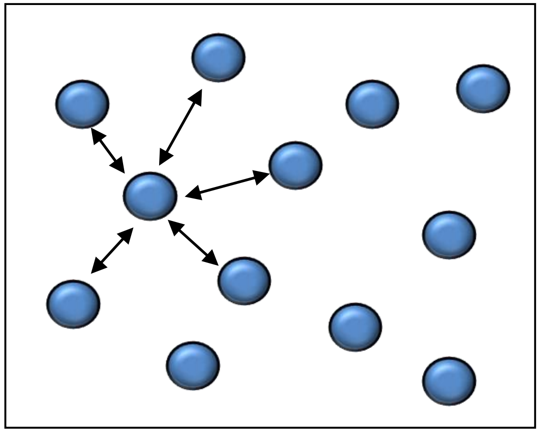


```
call init
T=0
do while (T.lt.Tmax)
    call compute_forces()
    call integrate_motion()
    call save_crds()
    call sample_averages()
    T = T + DT
enddo
call save_state()
stop
end
```

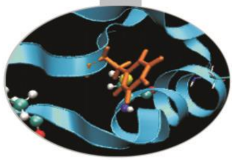
```
subroutine compute_energy_forces

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
```



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



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.

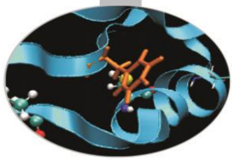
There are usually two types of long range non-bonded interactions:

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.

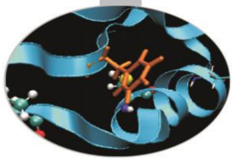
Electrostatic interactions are commonly solved with the Particle Mesh Ewald (PME) Method or similar. (electrostatic cutoffs are too approximate)

GROMACS timings



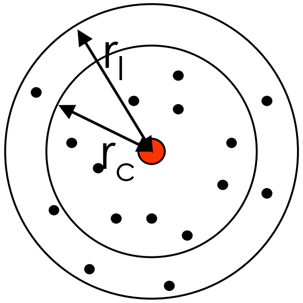
```
Computing:                           M-Number         M-Flops  % Flops
-----------------------------------------------------------------------------
 LJ                               66460.022385     2193180.739     2.8
 Coul(T)                          67295.126727     2826395.323     3.6
 Coul(T) [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
.....
```

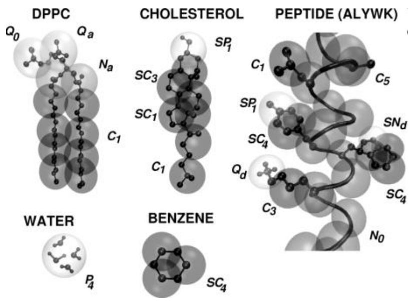
Optimising a serial program



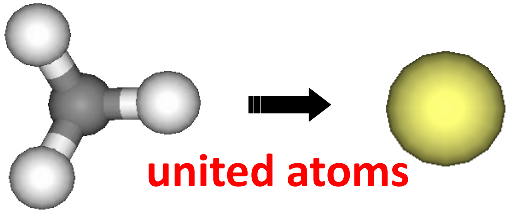
- To increase the performance of the program the number of interactions $O(N^2)$ to be calculated needs to be reduced.
- Common strategies include:
  - Potential cutoffs + Neighbour lists
  - United atoms (e.g. CH_4) or coarse grain approaches (e.g. Martini) to reduce the number of interacting sites
  - Holonomic constraints (e.g. SHAKE)
  - Multiple time steps (e.g. electrostatic time step in NAMD)
  - Implicit solvents as opposed to explicit solvents (but not recommended).

cut-off and neighbour list



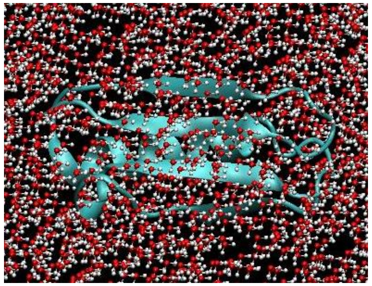


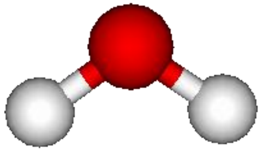
martini model



united atoms

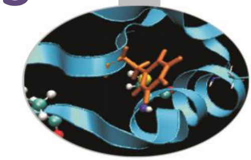
implicit and explicit solvents



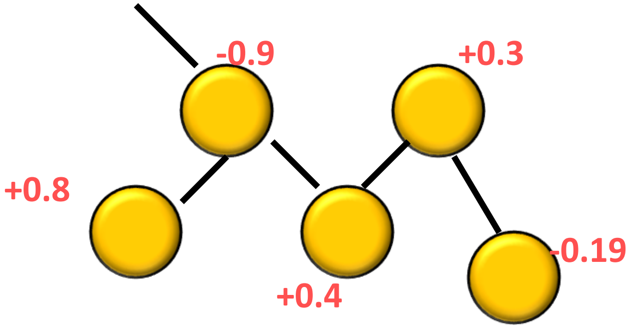


holonomic constraints (e.g. SHAKE) Δt=1fs → Δt=2fs

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):

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

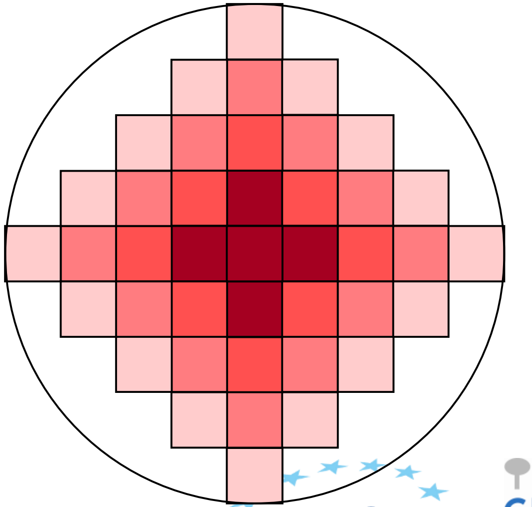
Problem: This is a long range interaction, varying with ~1/r. (c.f LJ, ~1/r^6) and so decays to zero slowly. The box cannot be made large enough without making the simulation impracticable. Electrostatic cutoffs on the other hand can give rise to artefacts.

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:

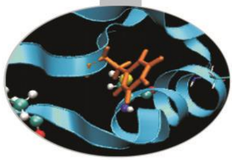
$$V = \frac{1}{2}\sum_{\mathbf{n}} \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{q_i q_j}{|\mathbf{r}_{ij} + \mathbf{n}|}$$

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



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

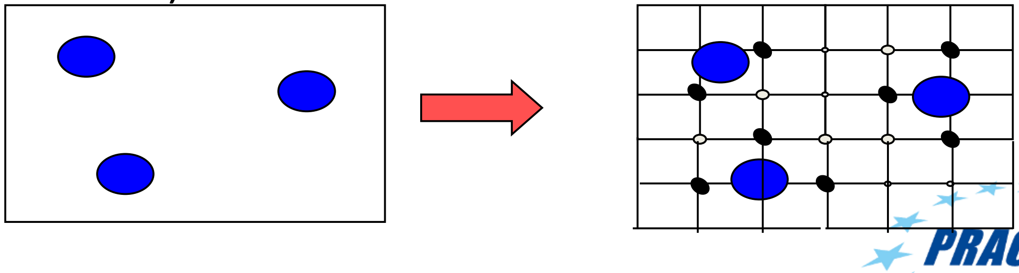
The real space term (which contains *erfc(x)*) can be calculated quite easily with standard libraries and usually a cutoff is applied (e.g. 9 Å).

Particle Mesh Ewald

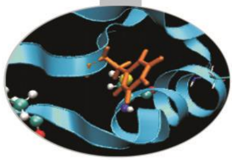
The second term converges quickly in reciprocal space but is computationally expensive:

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

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 a serial program



Do we need to parallelise MD ?

Galileo

gromacs 4.6.7 1 node (16 cores)

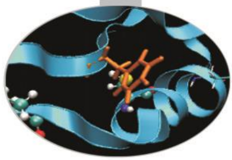
```
                Core t (s)   Wall t (s)
(%)
       Time:     8060.990      504.626
1597.4
                  (ns/day)    (hour/ns)
Performance:         3.425        7.008
Finished mdrun on node 0 Fri Nov  6 15:26:07
2015
```

PC:

```
                   (ns/day)    (hour/ns)
Performance:         0.075      319.699
```

Even using just one node of a cluster we can get speedups of 10X, 100X or more.

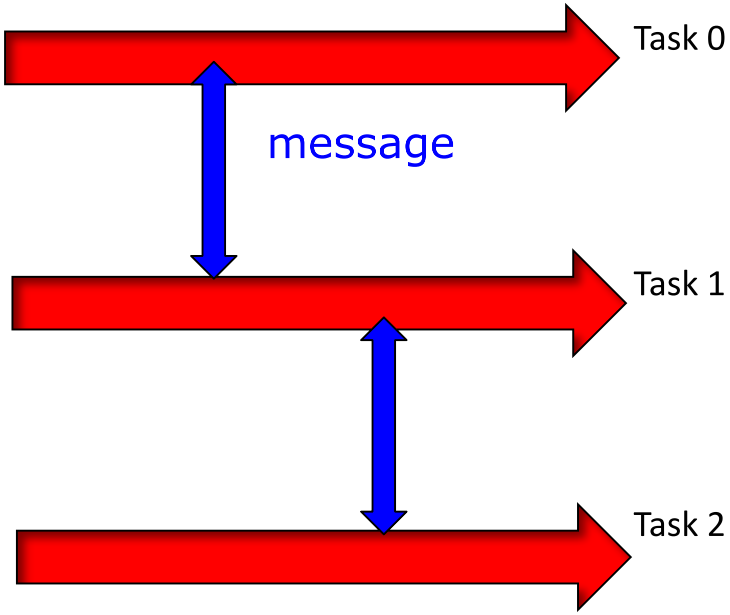
Concepts and practice of Parallelism



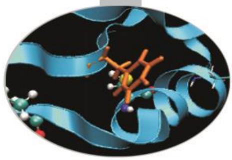
- Even if you do not intend to write a parallel program, just use one already present, it is important to understand some of the concepts and techniques used in the preparation and execution of a parallel project.
- Hardware is moving quite quickly so it is a challenge to understand everything but useful topics include:
  - MPI and message passing
  - OpenMP and threads
  - Accelerators such as GPUs
  - Measuring performance

Message Passing Interface (MPI)

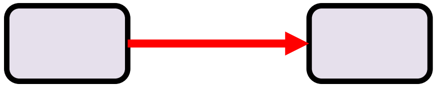
- MPI is a standard which implements parallelism via *message passing*, i.e. providing a mechanism for communication between parallel tasks.
- Usually SPMD (**S**ingle **P**rogram **M**ultiple **D**ata) model where multiple instances of the same program are launched. When necessary they communicate by MPI calls.
- Each instance is called a ***task*** and is identified by its ***rank*** (starting from 0). Normally all the tasks are created at the beginning of the parallel execution.

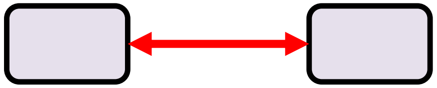


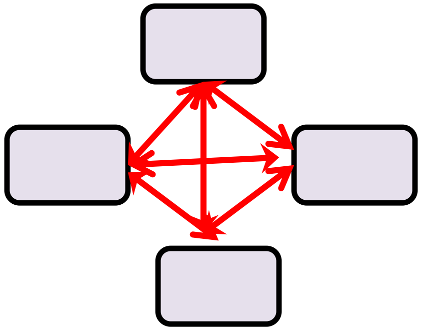
MPI Communications



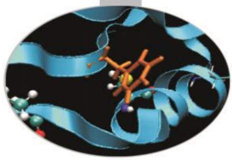
- MPI communications can be of various types:
  1. One-way communications.
  2. Point-to-point between two tasks.
  3. Collective calls between groups of tasks or even all of them.
- They can also be synchronous or asynchronous.
- Collective calls can be expensive, particularly when many tasks are involved.
- An efficient MPI program will minimise the time spent in communications as much as possible often by overlapping communications with calculations (*non-blocking communications).*







Using MPI



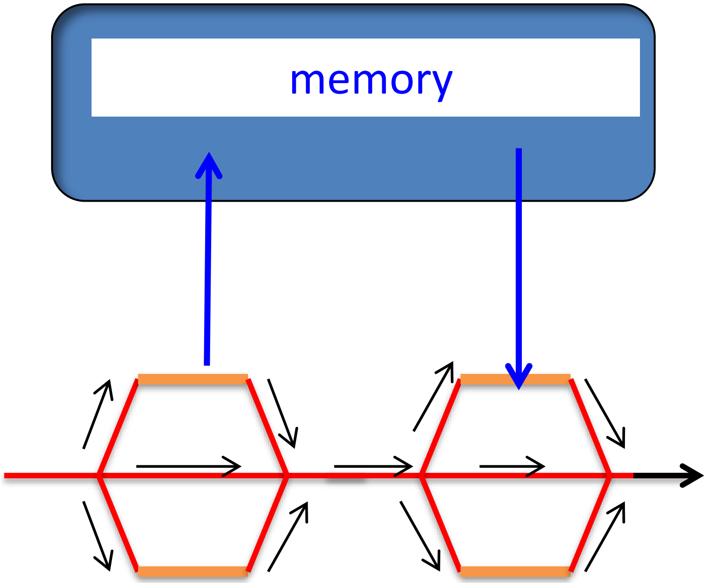
- MPI is implemented as a library which is used during compilation/linking and often also at execution.
- Different implementations may exist on a particular computer system (e.g. Intel MPI, OpenMPI, etc).
- Usually used within a launcher (e.g. mpirun, mpiexec, runjob, etc) which launches the required number of tasks.

```
module load intelmpi        # Intel MPI
mpirun –np 64 ./myprog.exe
```

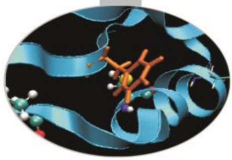
- Advantages:
  - Only standard model which allows cores over multiple nodes in a cluster to be used in a parallel program.
  - Highly optimised for current architectures
- Disadvantages:
  - Complex programming model and may require high memory (program instances + buffers)

OpenMP and threads

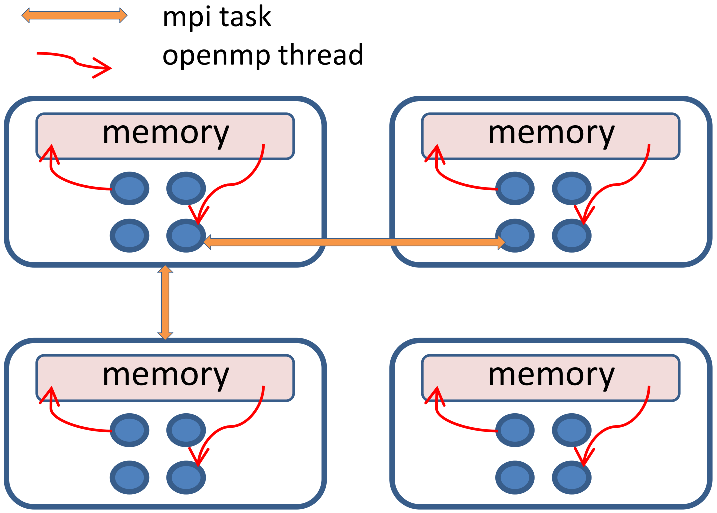
- The OpenMP standard implements parallel programming via *threads*.
- Threads are light-weight processes, requiring fewer resources than MPI tasks. Usually created and destroyed in a fork-join process.
- Often used for “work sharing” within loops but can be used to generate tasks.
- They communicate by reading and writing program variables in shared memory.
- Advantages:
  - Less memory and *may be* faster than MPI within a shared memory node. Simpler programming model than MPI.
- Disadvantages:
  - Cannot be used between separate nodes



Using OpenMP

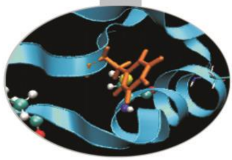


```
gfortran –fopenmp
myprog.c –o myprog
export OMP_NUM_THREADS=8
./myprog.exe
```



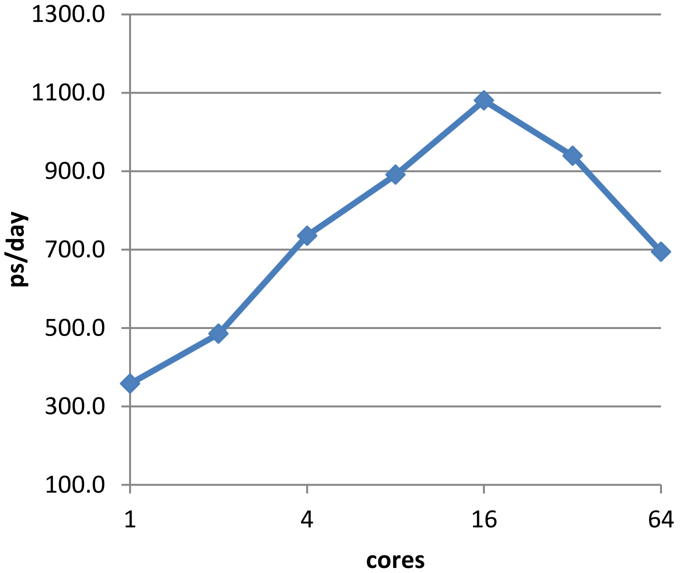
- But since openmp cannot be used between nodes common to use both – so called hybrid MPI/OpenMP programs (e.g. Gromacs)
- Typically use OpenMP thread within a node but MPI between nodes. Useful for minimising the number of MPI tasks. (see later)

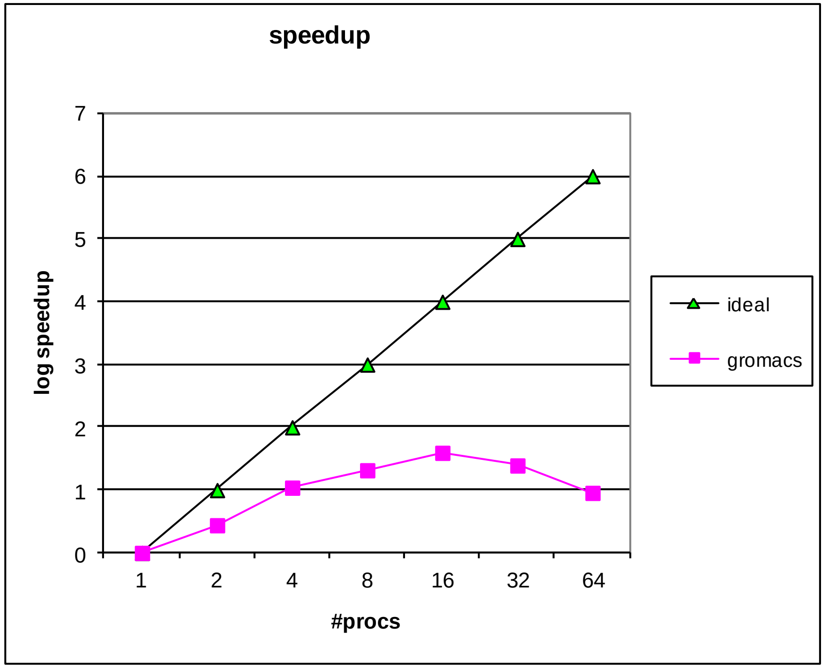
Strong and weak scaling and parallel efficiency



- For any parallel program important to measure the performance as a function of the parallel resources used (e.g. MPI tasks, threads, physical cores, etc).
- For MD usual to measure performance in terms of ns/day and this value is reported by most MD programs. Since computer grants are based on use of physical resources (e.g. cores) makes sense to plot performance against processor cores.
- This is called *strong* scaling and by comparison with the *ideal* case indicates how well parallelised your set up is. This can be emphasised by plotting the speedup with respect to the smallest number of cores used (e.g. 1 core).

Strong scaling examples





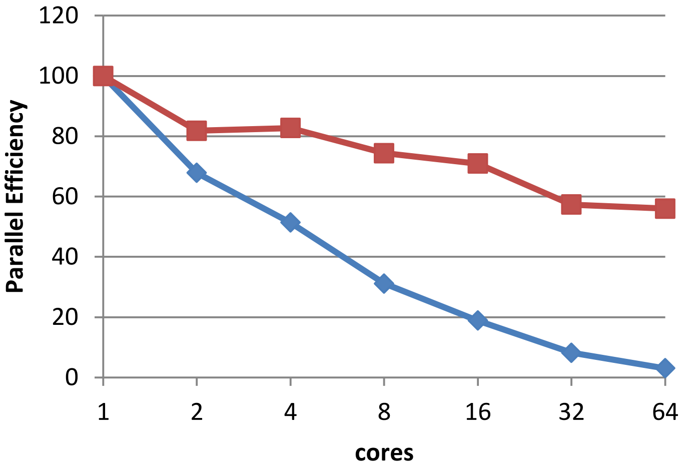
Speedup R

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

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

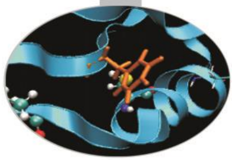
Strong scaling and parallel efficiency

- Computer scientists often prefer a metric called *the* *parallel efficiency*.
- Less interesting for MD researchers but worth quoting for grant applications (where the reviewers may be non MD users).
- Important to do strong scaling curves before embarking on production.

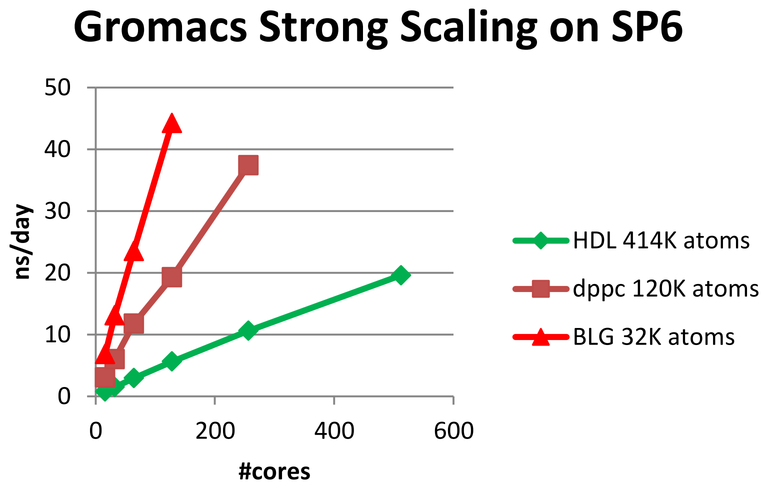


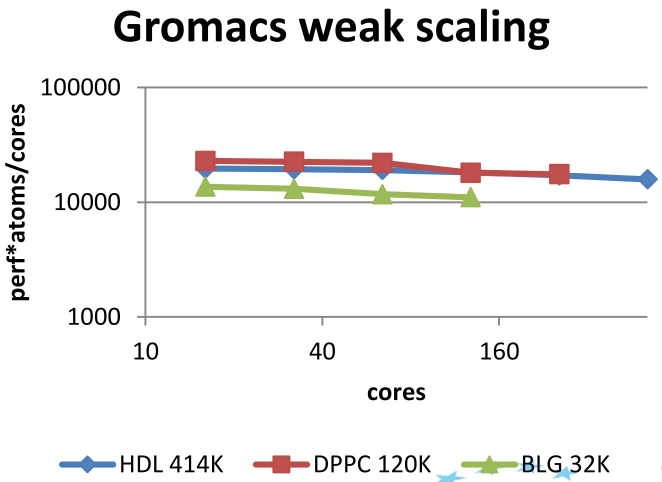
$$S = 100 \times \frac{P_N}{N \times P_1}$$

Weak scaling

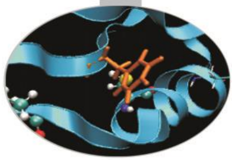


- This is formally as defined as “*how the solution time varies with the number of processors for a fixed problem size per processor.*”
- But usually used to know how the performance varies on increasing the input or problem size. Should be a horizontal line for perfect weak scaling.
- For MD this indicates how the performance varies with system size, i.e. number of atoms.
- Not often used in MD since researchers use one or only a few systems, probably with similar numbers of atoms.



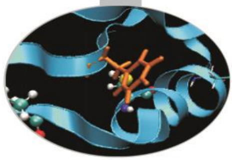


Other parallel concepts



- SIMD (Single Instruction Multiple Data) Vectorisation
  - Special hardware in the CPU (SIMD or Vector Unit) for optimising loops. For Intel known as SSE, AVX, etc (depending on processor version)
  - Most users do not need to know about this unless compiling or writing their own code.
- Load balancing
  - If parallel tasks in the program finish their calculations more or less at the same time, there is good "load balancing". If some processes have to wait for other processes then clearly the program will take longer.
- Parallel I/O
  - Often one task (e.g. rank 0) is given the job of reading and writing files since having many tasks accessing the same file is not safe. This task then sends the data to the other tasks.
  - For very large files and many processes may be more efficient to allow multiple access. Normally achieved by MPI-IO or specialist formats (HDF5).
  - In MD not normally used except for very large simulations (millions of atoms), e.g. in NAMD 2.10 or DL_POLY4.

Why do parallel programs stop scaling?



Regardless of algorithm, as the number of parallel tasks increase the relative time spent doing communications also increases, thus reducing the time for calculations.

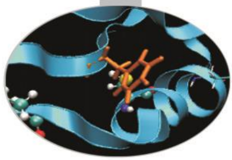
Very roughly, when

Time (communications) > Time (calculations)

increasing the number of processors will not lead to an increase in performance (in fact it may start decreasing).

Of particular importance are global or *collective communications* involving groups or even all the parallel tasks and programmers tend to minimise their use.
Other factors affecting scaling may include increased I/O or memory usage.

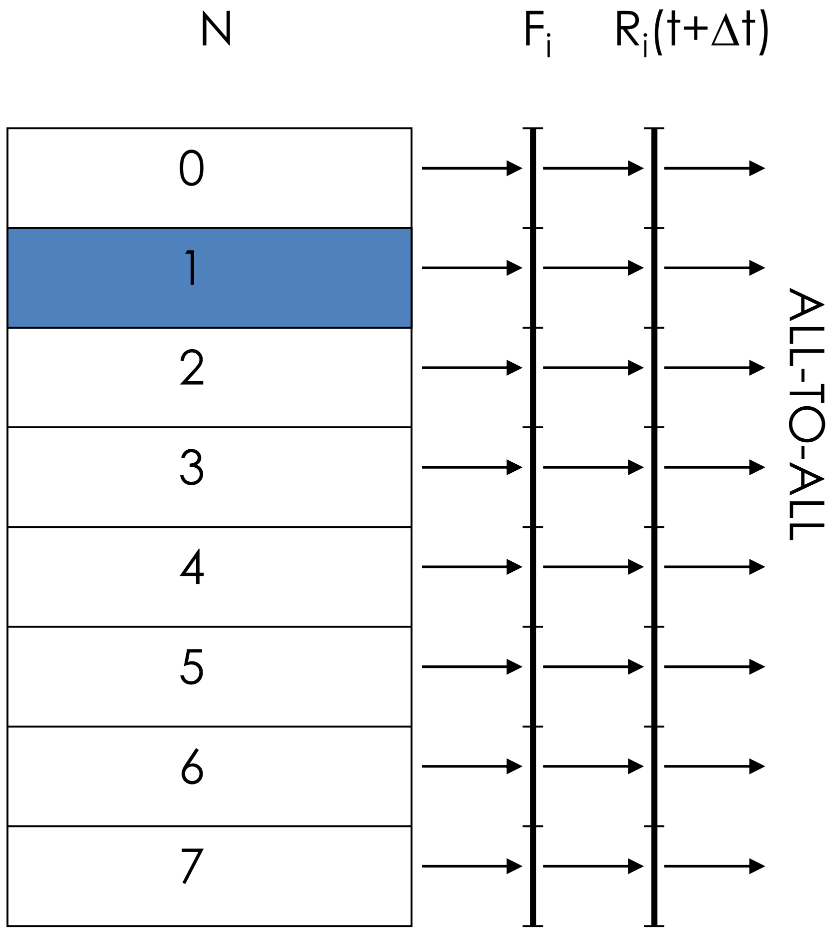
Parallelising Molecular Dynamics



- Now we have the tools how can we parallelise an MD program?
- Need an algorithm to accelerate the most timing consuming parts of the serial program, i.e the non-bonded long ranges forces calculation.
  - Dispersion forces
  - electrostatic forces with PME
- But must minimise communications between tasks.

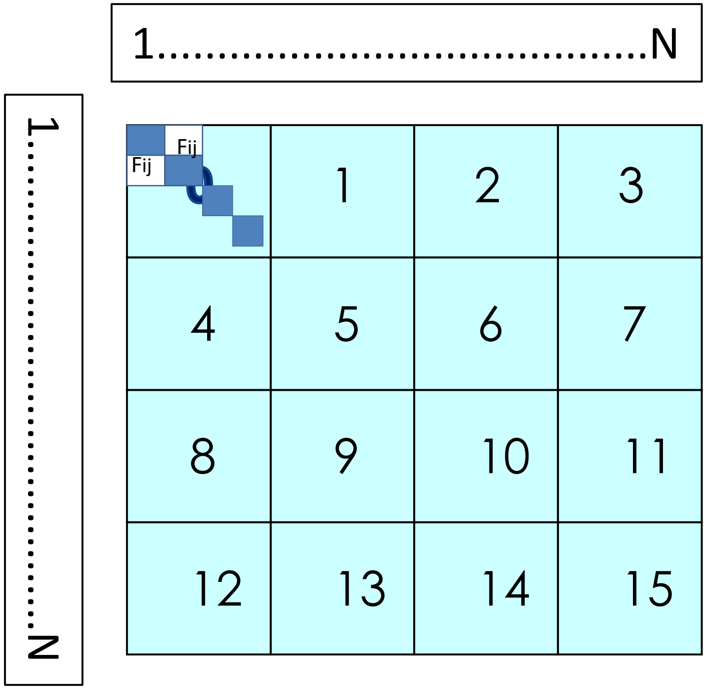
Atom (particle) decomposition

- One of the first algorithms implemented for parallel MD. Sometimes also called "Replicated Data" since each processor requires a copy of the entire system.
- Nowadays rarely used because of the high memory and global communications required.
- The particle decomposition option of Gromacs was removed in the latest release.



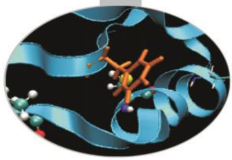
Force decomposition

- Improvement on particle decomposition, inspired by the parallel algorithms for matrices.
- Reduced memory and communication overheads but still relatively expensive at high core counts.


1...N
1..............................N
Fij
Fij
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

F_{ij} force matrix

Spatial (or domain) decomposition algorithm

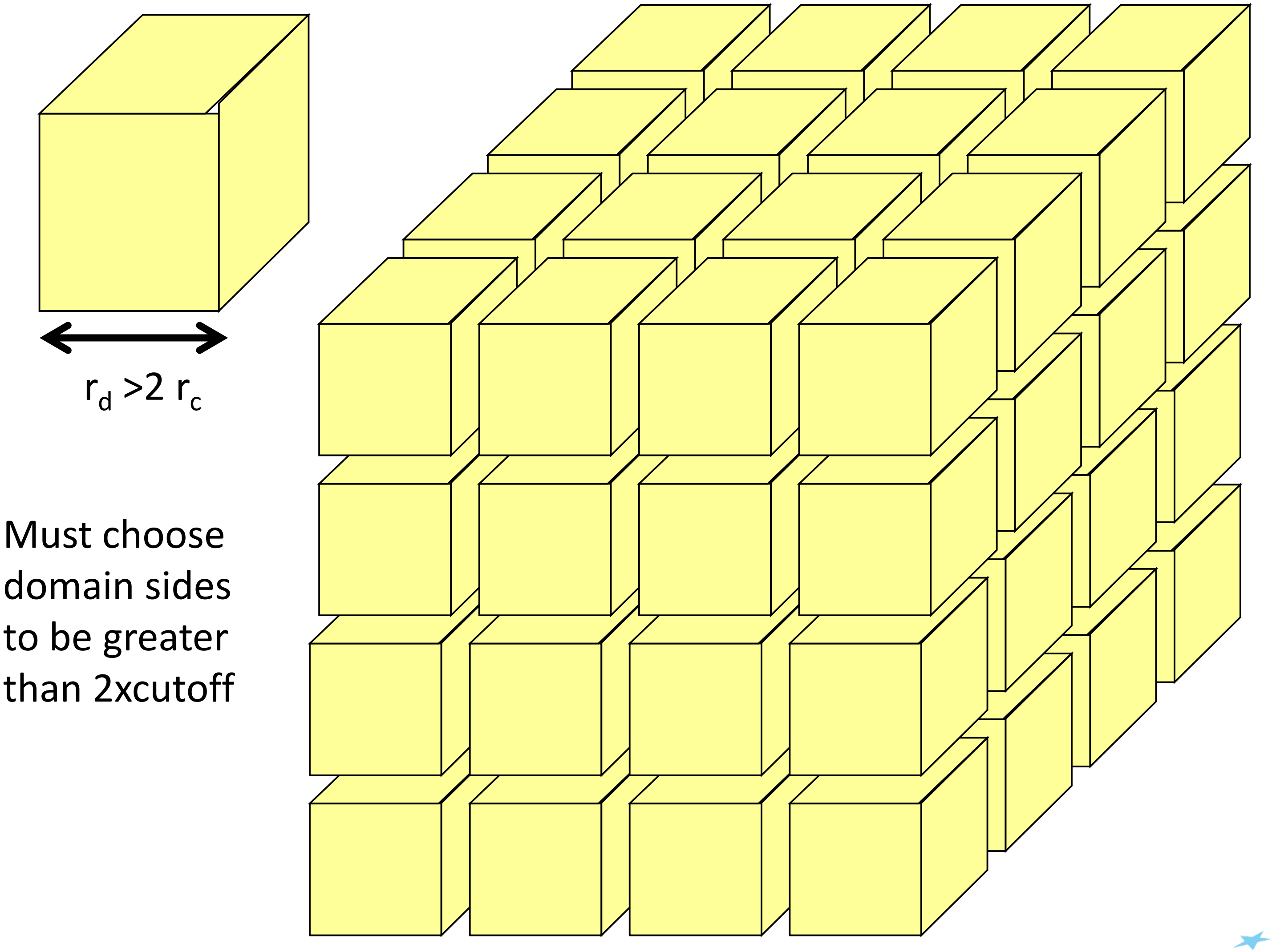


Here each processor is assigned to a spatial region of the simulation box (with side $r_d > 2*r_c$) such that it stores only a portion of the whole system. This has two components:

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

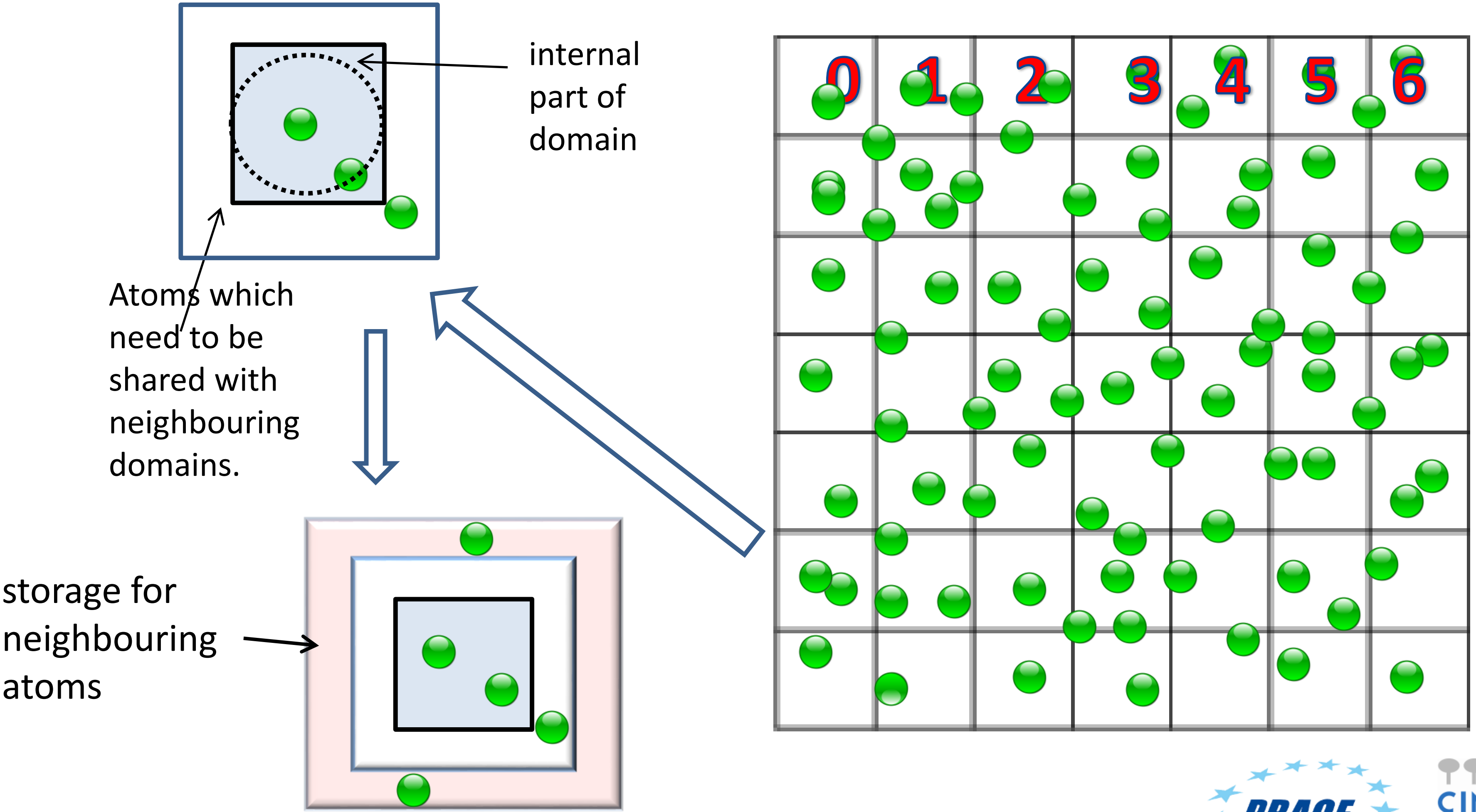
In order to minimise the surface with respect to the volume, and hence the communications, it is important to use regions that are as cubic as possible. In any case the communications are reduced since it is not necessary to update the whole system in local memory.

Domain decomposition

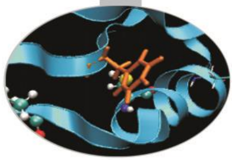


Must choose domain sides to be greater than 2xcutoff

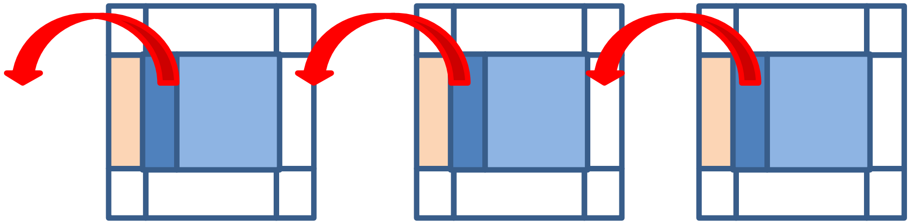
Domain Decomposition



Simple Domain Decomposition - algorithm

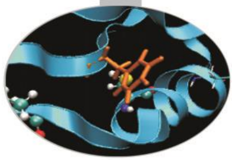


1. Read in atomic coordinates (and velocities)
2. Assign atoms to domains (processors) according to x,y,z position.
3. For each domain (processor):
    1. identify interacting atoms in neighbouring domains and copy coords.
    2. calculate forces.
    3. copy partial forces of neighbour atoms back to their home domains
    4. with the forces calculate new velocities and positions.
4. Calculate thermodynamic averages (T, P,E, etc)
5. Loop back to 2 if not finished.



MPI has many commands for transferring data efficiently in a cartesian topology (such as a simulation box.)

Domain decomposition



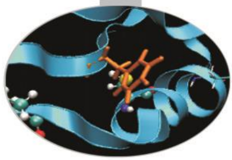
Advantages

- 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.

Novel domain-decomposition schemes

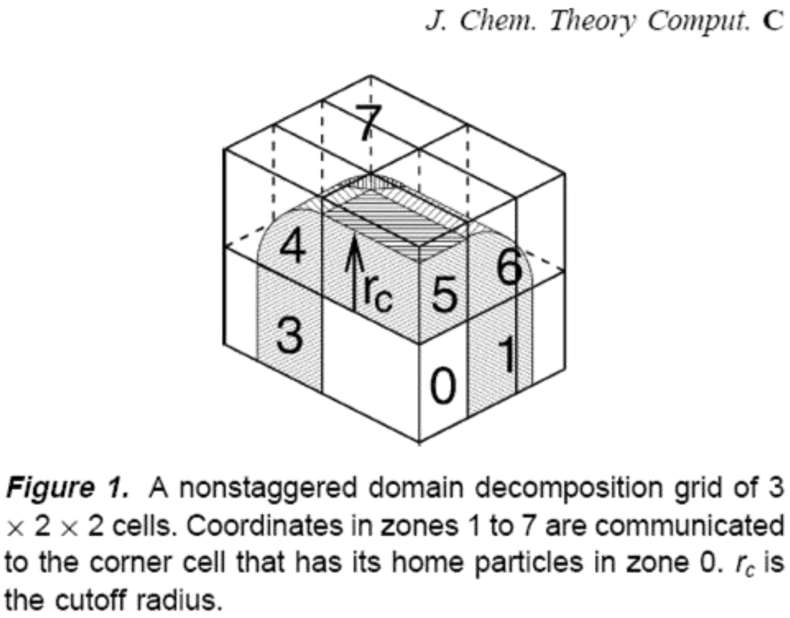


Problem with domain decomposition occurs when density of particles is uneven or fluctuates.

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.x now has ***Dynamic Load Balancing*** which adjusts dynamically particle-processor assignment.



J. Chem. Theory Comput. C

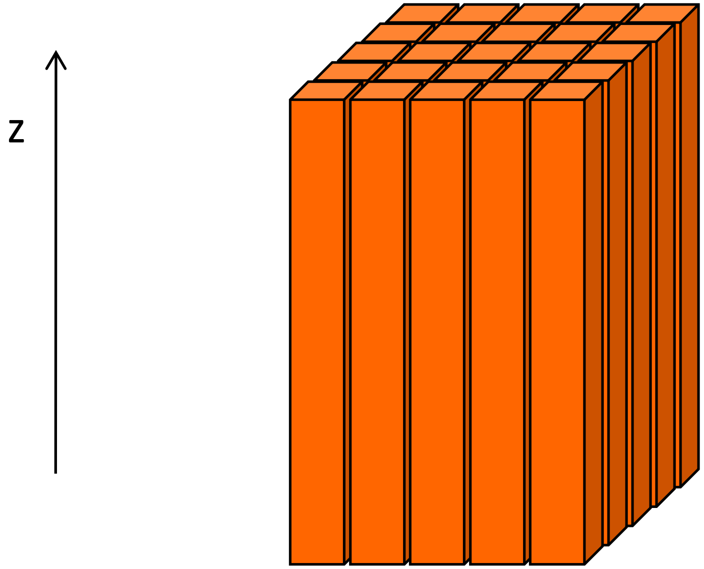
Figure 1. A nonstaggered domain decomposition grid of 3 × 2 × 2 cells. Coordinates in zones 1 to 7 are communicated to the corner cell that has its home particles in zone 0. r_c is the cutoff radius.

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

Parallelisation of Electrostatics with Domain Decomposition and PME

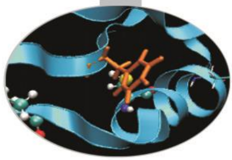
PME can be parallelised with a DD scheme but 3D FFT is very inefficient for many processors (or small N) because of all-to-all global communications (e.g MPI_AlltoAll).

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?

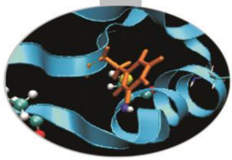


Compare

- 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?

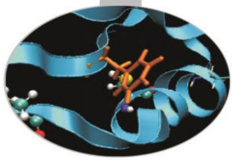


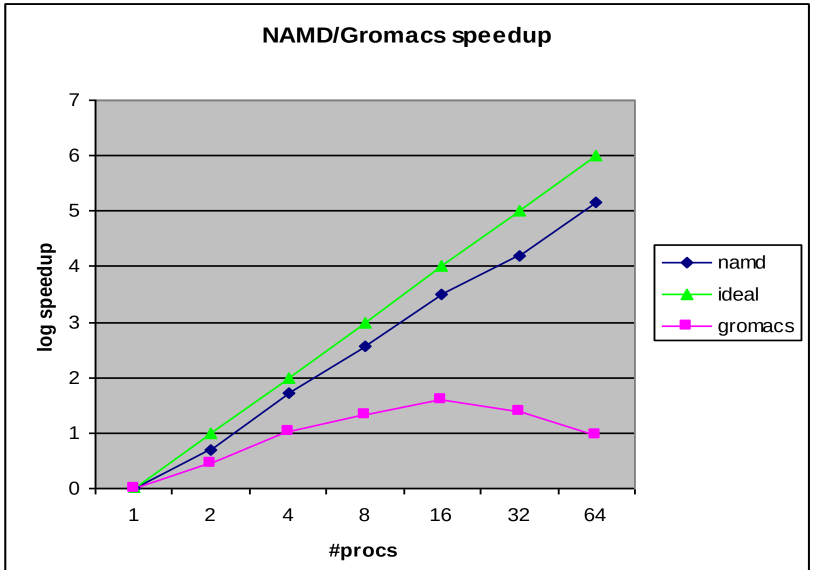
Compare

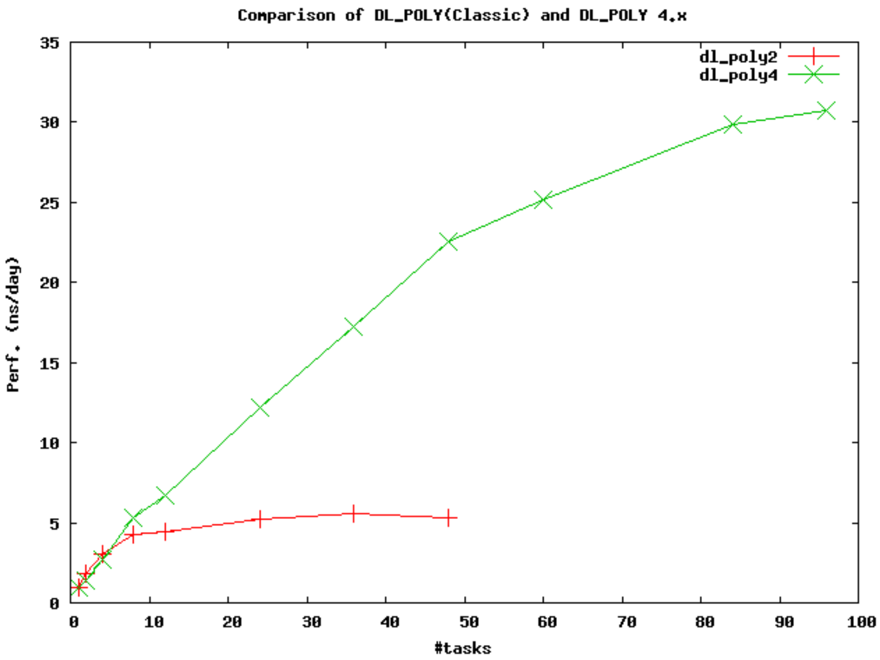
- GROMACS v 3.x and earlier with force-decomposition schemes
- GROMACS v 4.x with domain decomposition
- NAMD with domain decomposition

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

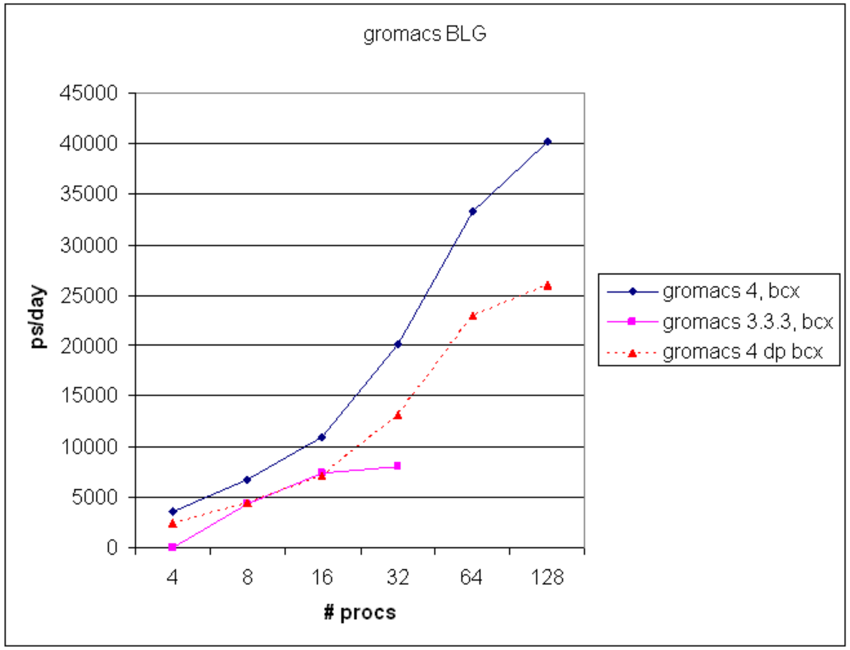
Does domain decomposition work?



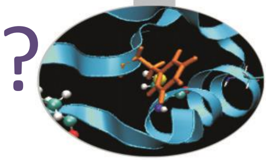




Simulation of 280K atoms of liquid argon with DL_POLY (Classic) and DL_POLY 4.03



Why do MD (programs stop scaling ?



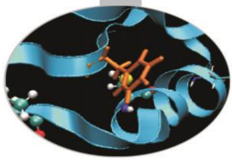
For most parallel programs the scaling levels out when the time of communications > time needed for calculations.

For modern molecular dynamics programs this can happen when the system is too small (i.e. number of atoms too low) 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 communications (in the 3D FFT) and this cost varies as N^2 .

As a rule-of-thumb many MD simulations reach a scaling limit when there are ca. 100-200 atoms/core.

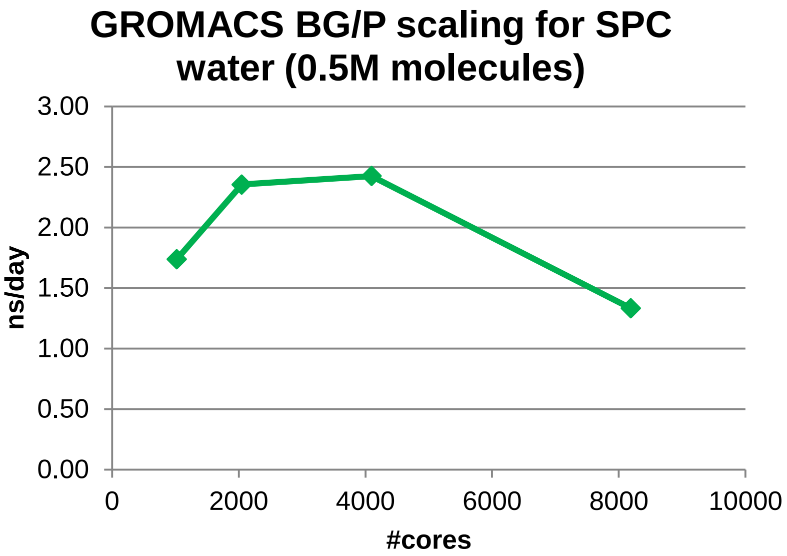
Why do MD programs stop scaling?



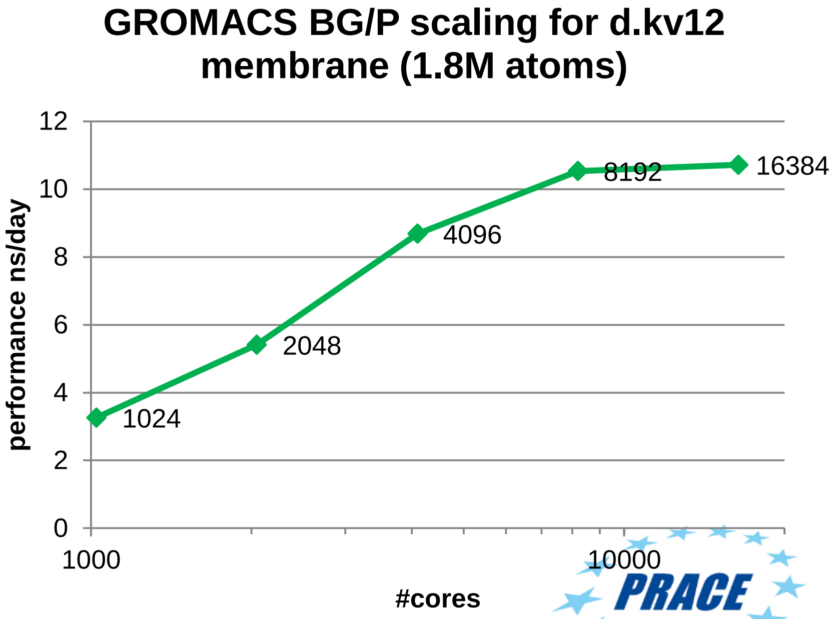
At the scaling limit communication time presumably > calculations, but which algorithm features cause this?

Candidate features:

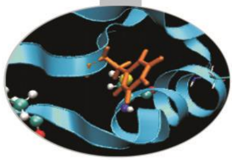
1. Non-bonded dispersion with DD or
2. PME for electrostatics.



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



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.

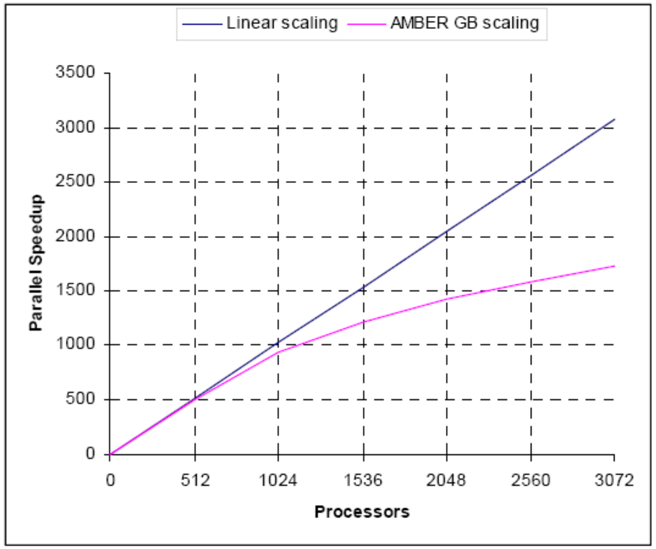


Figure 1. Parallel scaling of AMBER on Blue Gene. The experiment is with an implicit solvent (GB) model of 120,000 atoms (Aon benchmark).

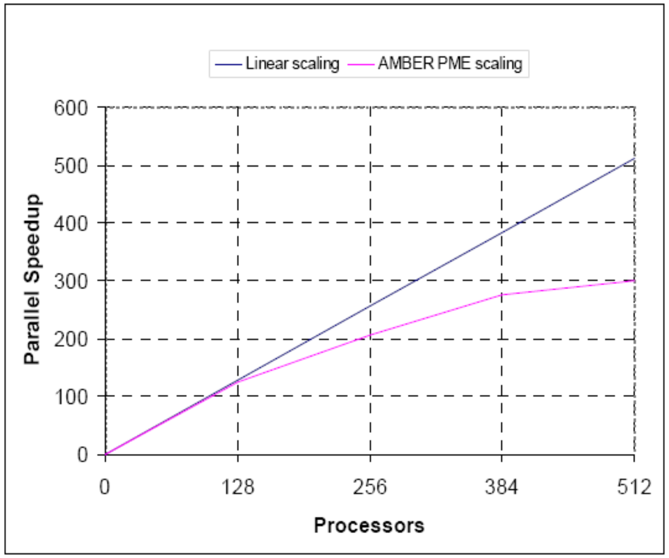
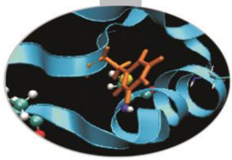
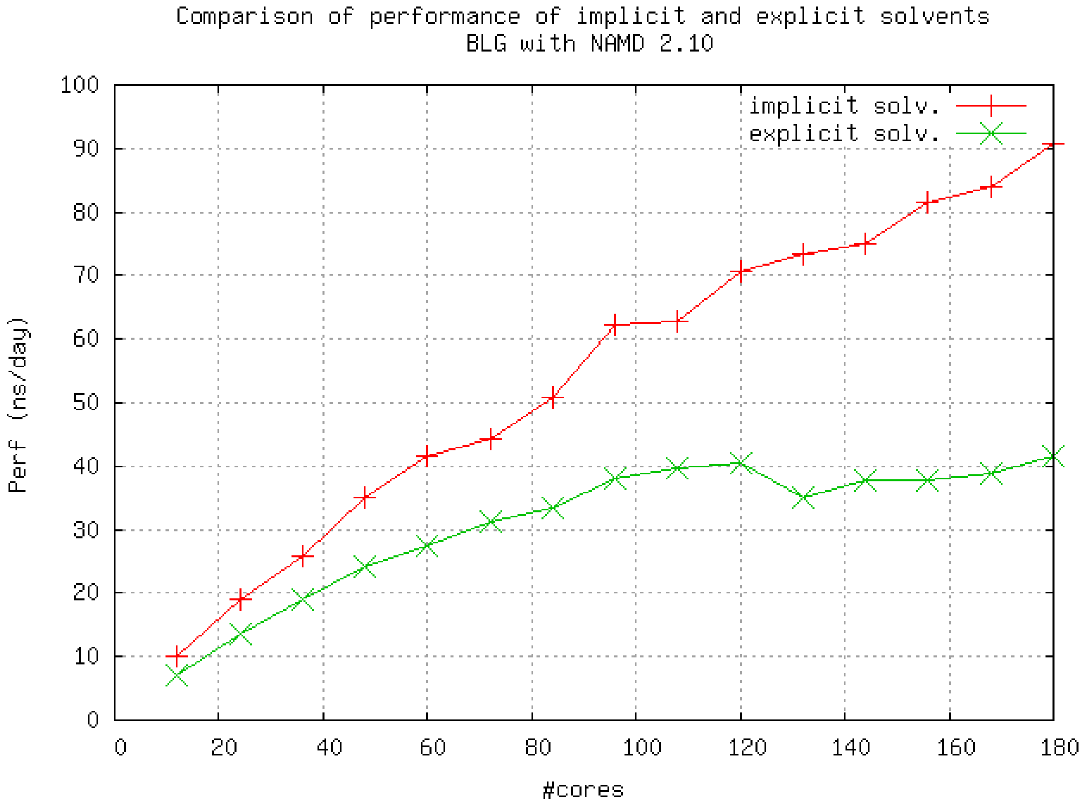


Figure 2. Parallel scaling of AMBER on Blue Gene. The experiment is with an explicit solvent (PME) model of 290,000 atoms (Rubisco).

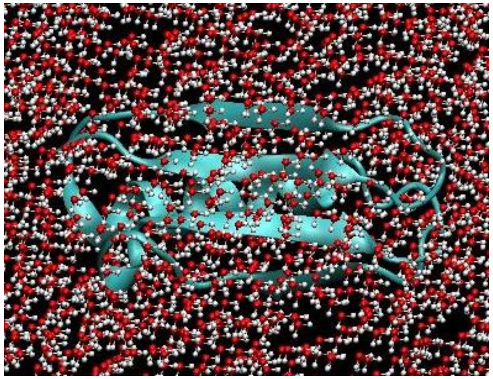
***Life Sciences Molecular Dynamics Applications on the IBM System Blue Gene Solution: Performance Overview*, http://www-03.ibm.com/systems/resources/systems_deepcomputing_pdf_lsmdabg.pdf**

Implicit and Explicit solvents





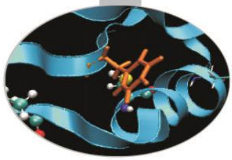
NAMD 2.10
Beta-lactoglobulin
in explicit and
implicit solvents



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

Generalized Born Equation

Why is parallel scaling important ?

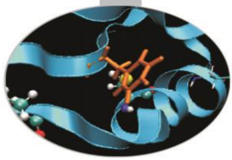


The Bluegene and other multi-thousand core architectures represent a challenge for projects based on molecular dynamics since often a minimum scaling is required.

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

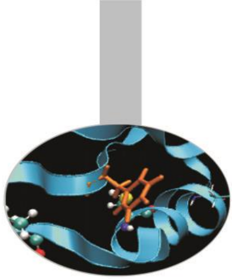
How can I increase the parallel scaling ?



It is generally accepted that the PME method has the most influence on parallel scaling due to the global communications in the FFT but even without PME the simulations reach a performance limit. How can we mitigate this ?

1. Reduce the communications in the PME calculations. (e.g. –npme option of GROMACS)
2. Try exploiting threads with hybrid MPI/OpenMP .
   - OpenMP allows a finer-grain parallelism. With fewer MPI processes we can have larger domain sizes.
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,..

Each system is different so important to benchmark your simulations to find the best results.



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.

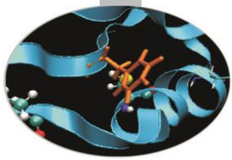
GROMACS 4.x allows separate nodes to be assigned to PME calculations:

```
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`**.

PME nodes



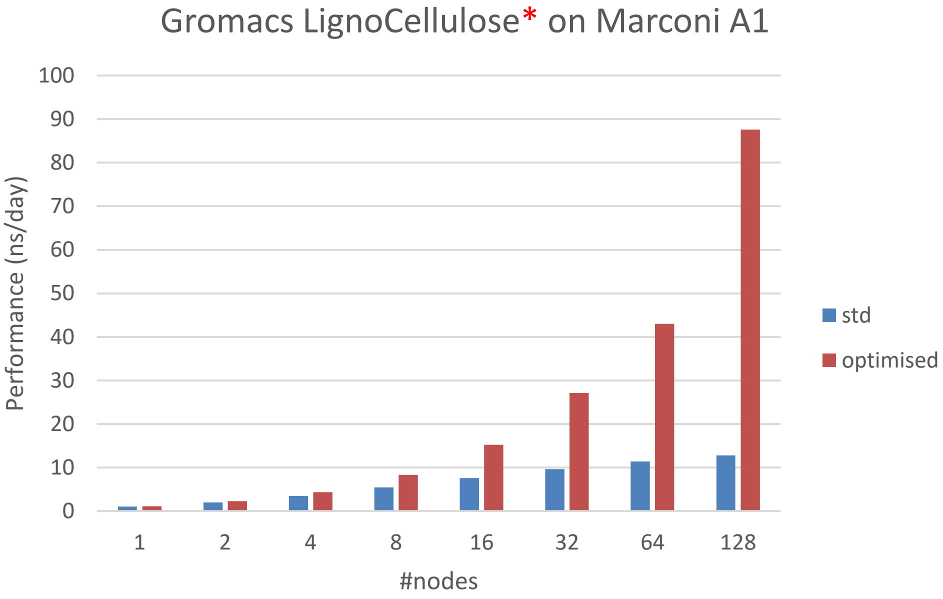
```
 Average load imbalance: 21.3 %
 Part of the total run time spent waiting due to load imbalance: 6.7 %
 Average PME mesh/force load: 1.277
 Part of the total run time spent waiting due to PP/PME imbalance: 15.4 %

NOTE: 6.7 % of the available CPU time was lost due to load imbalance
      in the domain decomposition.
      You might want to use dynamic load balancing (option -dlb.)

NOTE: 15.4 % performance was lost because the PME ranks
      had more work to do than the PP ranks.
      You might want to increase the number of PME ranks
      or increase the cut-off and the grid spacing.
```

Always checkout the output of your program for hints on performance.

PP nodes/ PME nodes	Perfromance ns/day
40 / 8	9.142
36 / 12	10.798

DPPC benchmark on 1 node Marconi Skylake

Very large Gromacs simulations



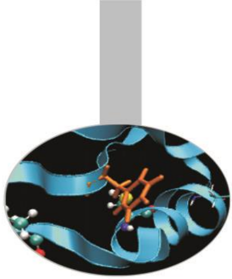
mpirun gmx_mpi mdrun -s topol.tpr -resethway -noconfout -gcom 20 -nstlist 20

-resethway	Resets perf counters
-noconfout	Do not write out final configuration
-gcom	Controls global communications freq.
-nstlist	Neighbour list

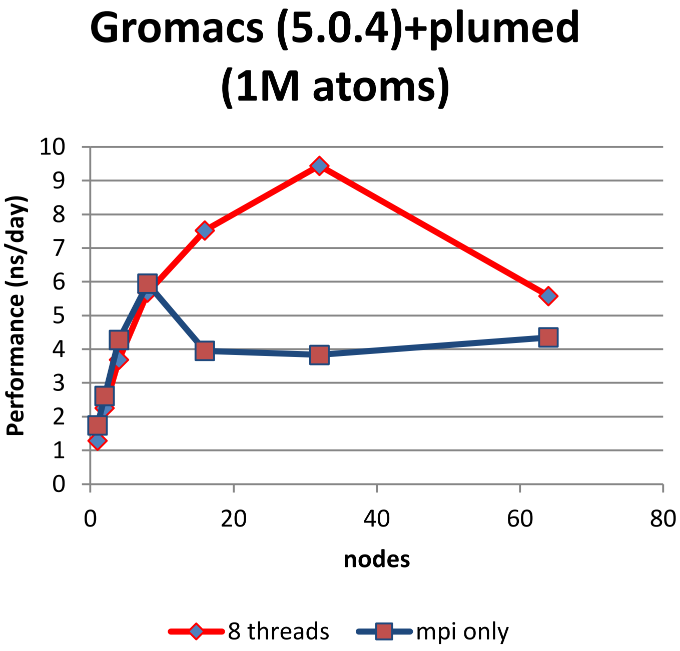
For very large simulations with Gromacs use runtime options to improve performance. In particular, -gcom to reduce the global communication (e.g. energy) frequency (at 128 nodes 48% time consumed in MPI_Bcast).

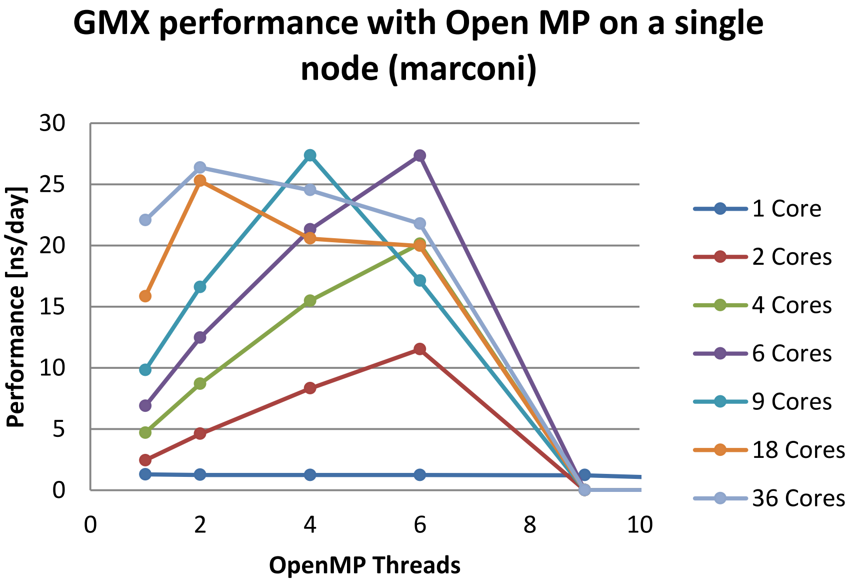
**Ligno cellulose model, using Reaction Field instead of PME (i.e. no FFT),4M atoms.*

Hybrid MPI/OpenMP - Gromacs



- GROMACS v 4.6 and upwards can use OpenMP parallelization for the PME.
- OpenMP threads use less memory than MPI tasks, and by replacing MPI tasks, reduce communications.
- For Gromacs ensure that **no. of threads x no. mpi tasks = no. of physical cores**
- Because of the overheads may give improvements only at high core counts or with slow networks.

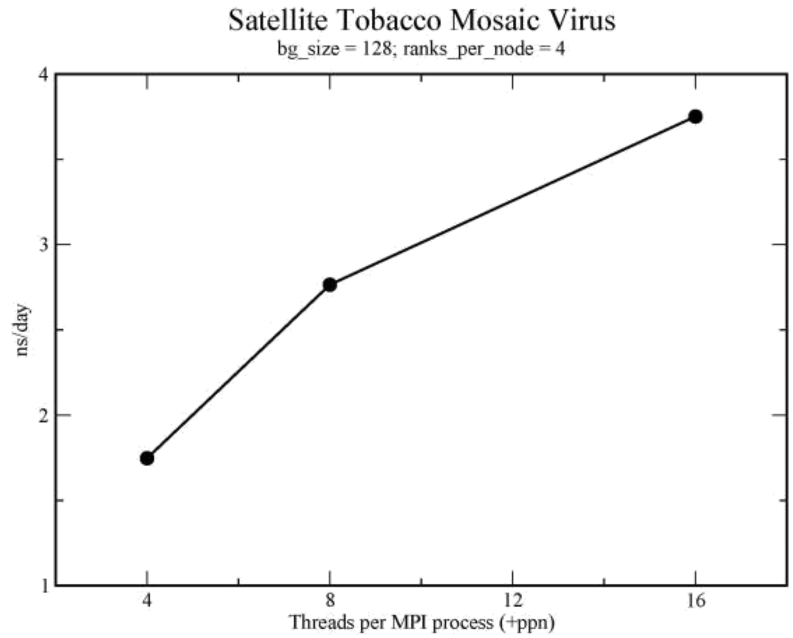




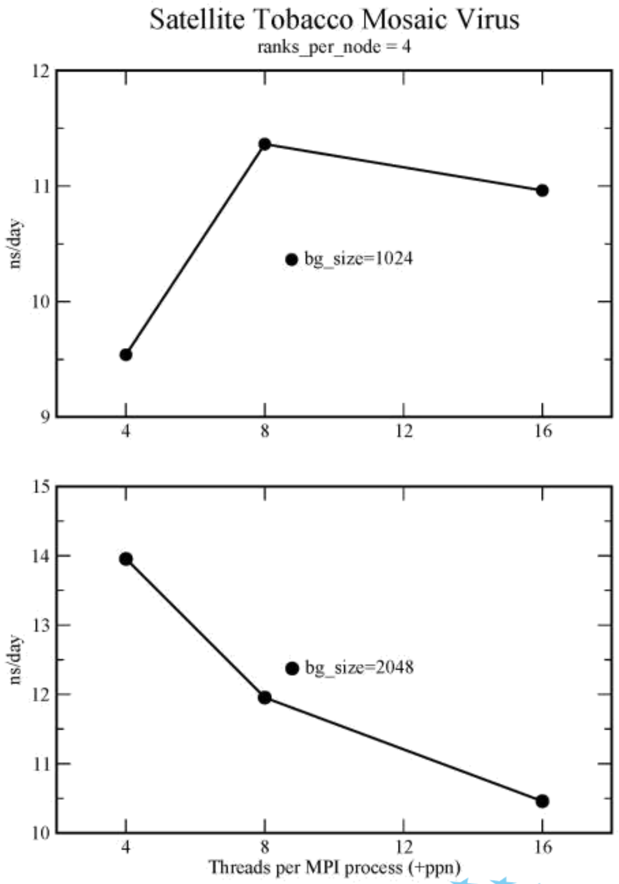
thanks to M. Alberghini

Hybrid MPI/OpenMP - NAMD

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

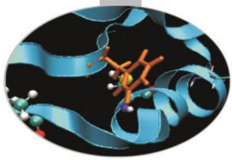


bg_size=128, ranks/node=4 (512 tasks)



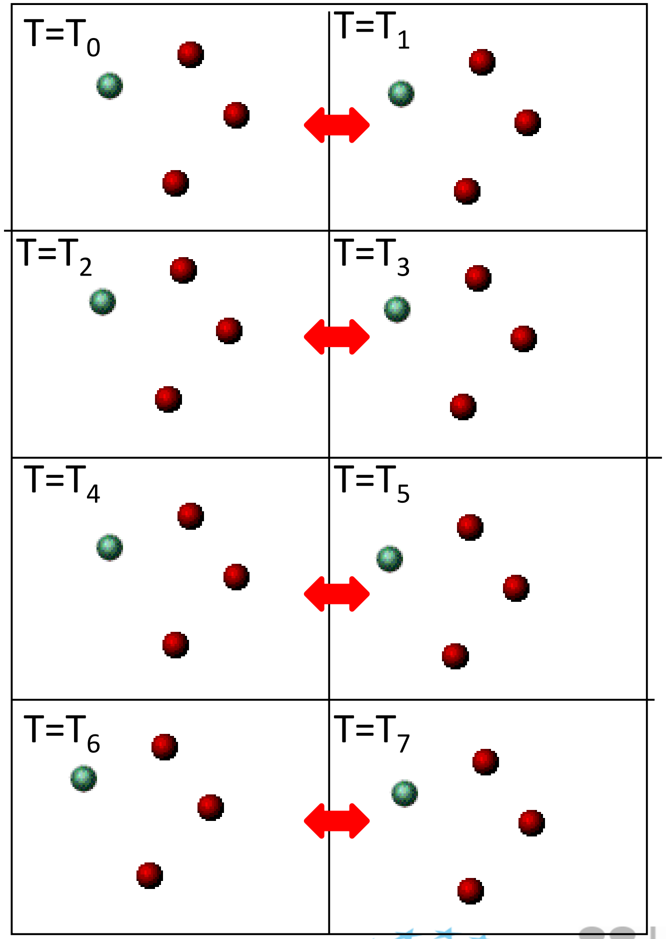
http://www.hpc.cineca.it/content/namd-benchmark

Replica Exchange Molecular Dynamics



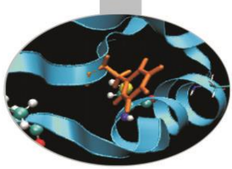
Replica Exchange Molecular Dynamics

- 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 *N* cores the individual replicas need only scale up to *N/n* cores for efficient performance.

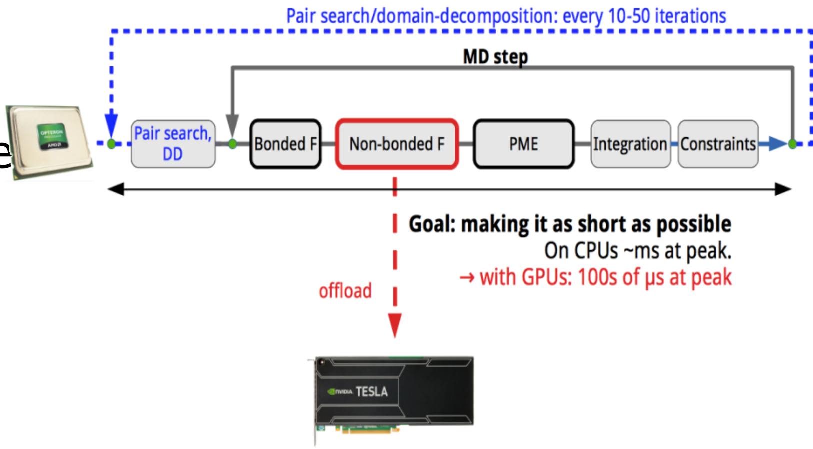


Other examples include metadynamics with multiple walkers (e.g. PLUMED), various other free energy algorithms, etc..

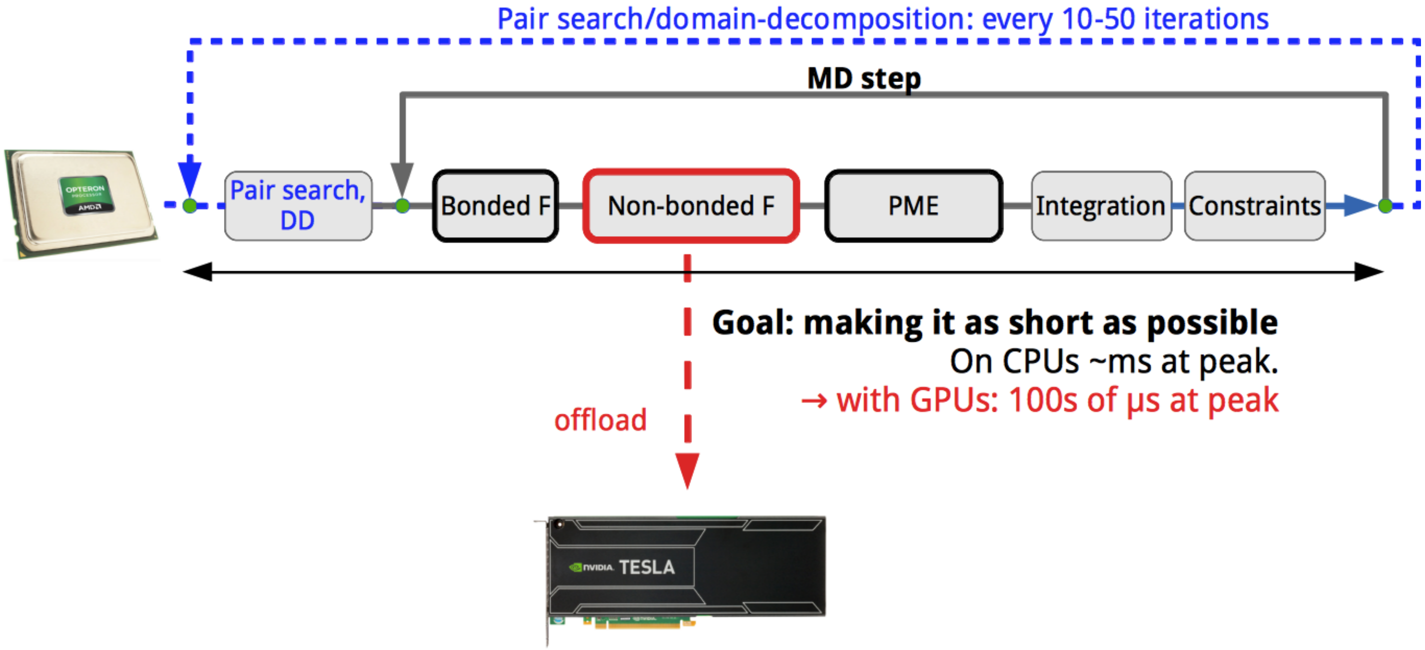
Molecular Dynamics and accelerators



- 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 the same performance but using fewer nodes.


Pair search/domain-decomposition: every 10-50 iterations
MD step
Pair search, DD
Bonded F
Non-bonded F
PME
Integration
Constraints
Goal: making it as short as possible
On CPUs ~ms at peak.
→ with GPUs: 100s of μs at peak
offload
TESLA

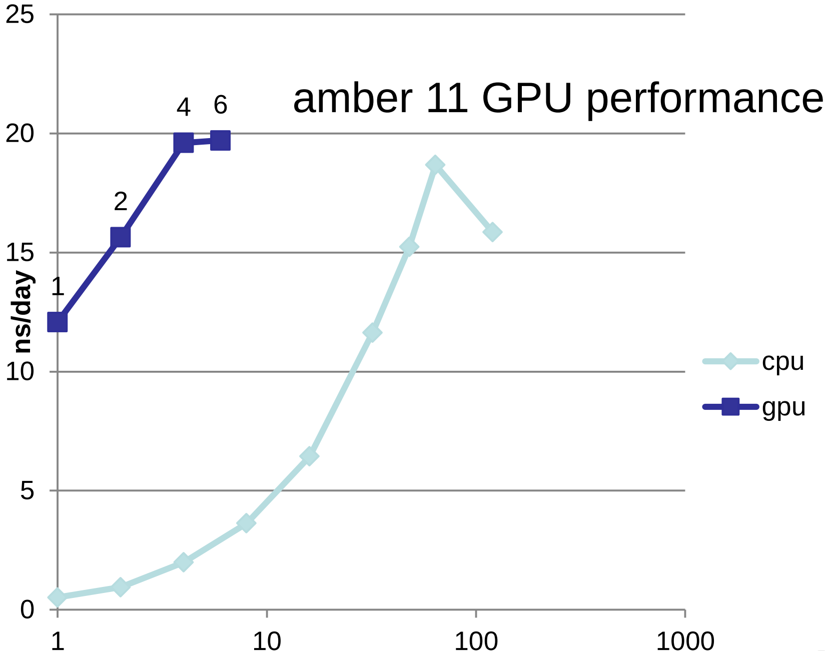
Molecular Dynamics and Acceleration - GROMACS



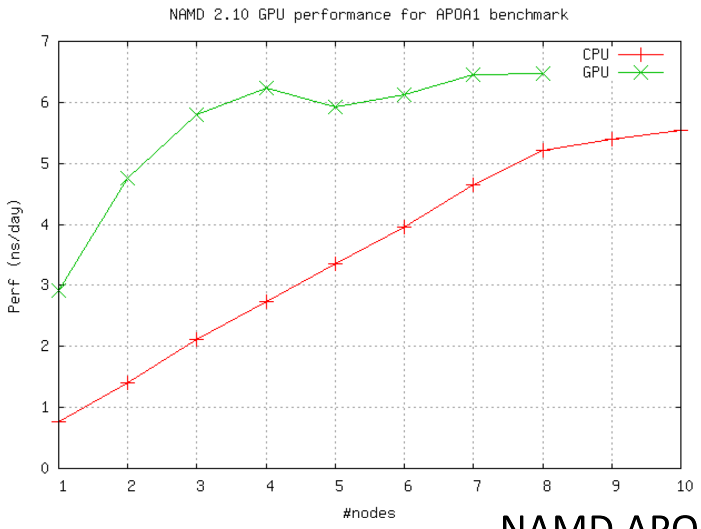
Goal: making it as short as possible
On CPUs ~ms at peak.
→ with GPUs: 100s of μs at peak

Gromacs offloads non-bonded (non PME) calculation to GPU while the main CPU does PME and bonded force calculations.
NAMD uses a similar strategy (I think)

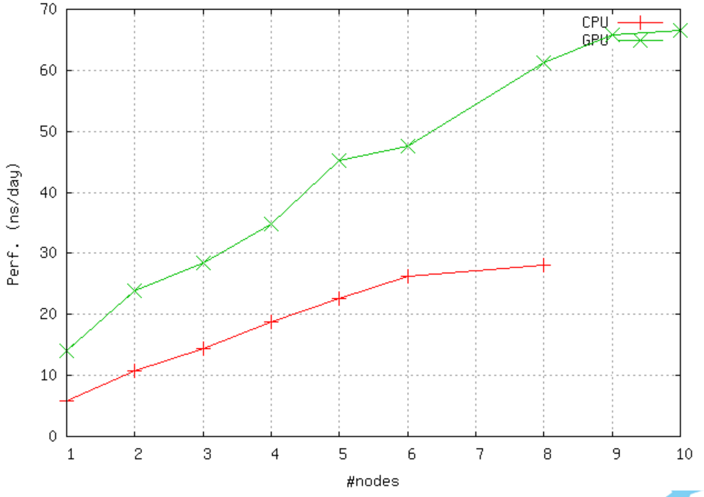
“Accelerated” Molecular Dynamics - results



DNA+water (40K atoms)



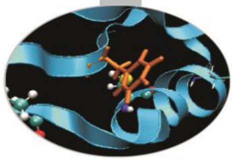
NAMD APOA1



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

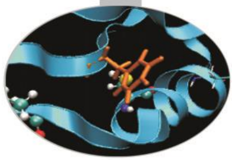
GROMACS 4.6
DPPC

Intel Knight’s Landing (KNL)

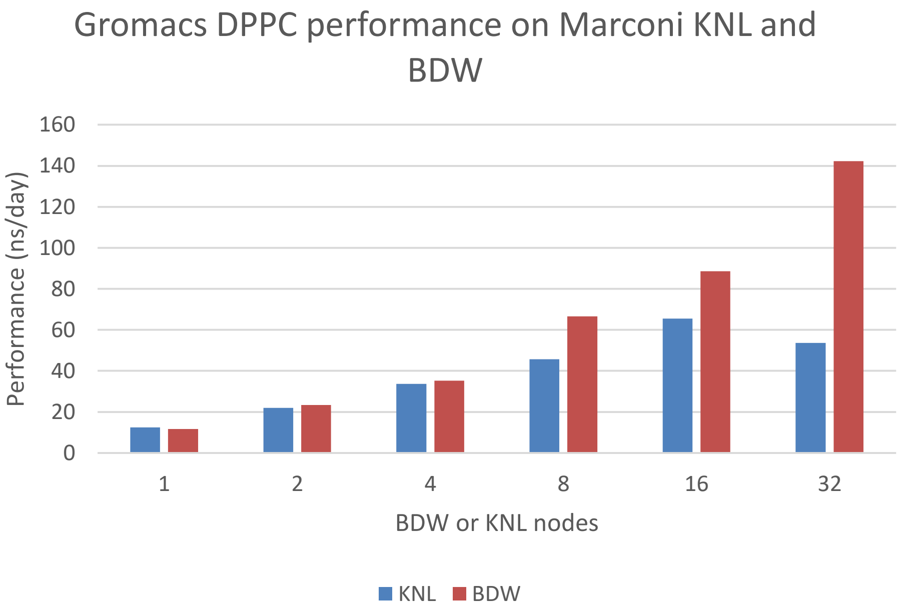


- The KNL version of the Intel Xeon PHI processor is not an accelerator so for most MD programs should work similarly to standard Intel CPUs.
- Although not essential, for performance recommended to recompile for KNL.
- Installation
  - GROMACS.
    - Recompile with -DGMX_SIMD=AVX_512_KNL to exploit the KNL vector processor.
  - NAMD
    - Recompile in SMP mode (Linux-KNL-multicore) according to the Intel website: *https://software.intel.com/en-us/articles/building-namd-on-intel-xeon-and-intel-xeon-phi-processor.*
- Running
  - Generally identical to usual CPUs but OpenMP threads might be needed to get reasonable performance (particularly NAMD).

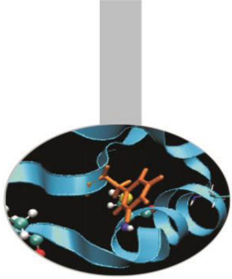
KNL and GROMACS



Similar performance to Intel Broadwell, although the DPPC benchmark performs less well with higher cores (don't know why).



KNL and NAMD



- Important to use the MPI-SMP version of CHARM++/NAMD on KNL.
- Unfortunately this version has a complicated syntax. Best to follow published recipes and see which works best (see Intel page).
- Small performance increase for very large systems, but otherwise use standard Intel processors.

The following gives about 5ns/day for the STMV virus benchmark (1M atoms):

```
# 17 MPI processes (for communication) * 7 threads/MPI (ppn) + 17 =  136
per node.
node=16
mpirun -perhost 17 -n $(($node*17)) $exe +ppn 7 +pemap 0-67,68-135:4.3
+commap 71-135:4 namd2  stmv.namd > stmv16.log
```

Process mapping

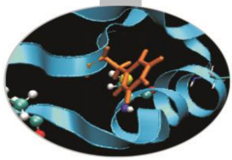
Threads/task

Total tasks

Communication thread mapping

tasks/node

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 PME/FFT for electrostatics

  The FFT is the more serious limitation.
- Many “normal” systems do not scale up to 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.
- Memory and I/O not normally problems but become important for very large systems (e.g. >1M atoms)
- No obvious candidate for beating the scalability barrier. Some interest in the use of Fast Multipole Methods or Multi-Level Summation Method (MSM) instead of PME but still very much in the research phase.