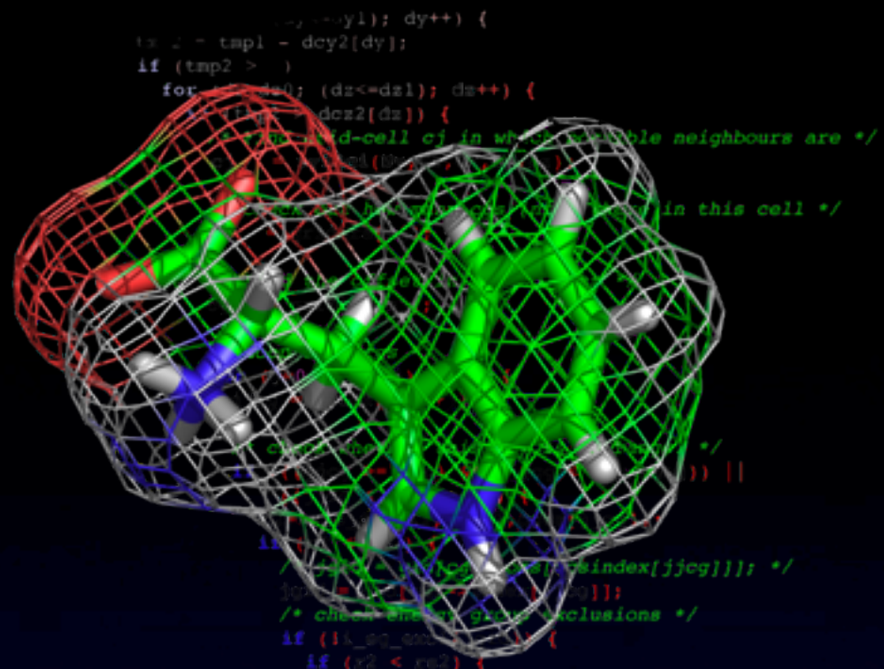
Braga 2010-06-18

Streaming Molecular Simulation: Gromacs, OpenMM & CUDA

Erik Lindahl

lindahl@cbr.su.se

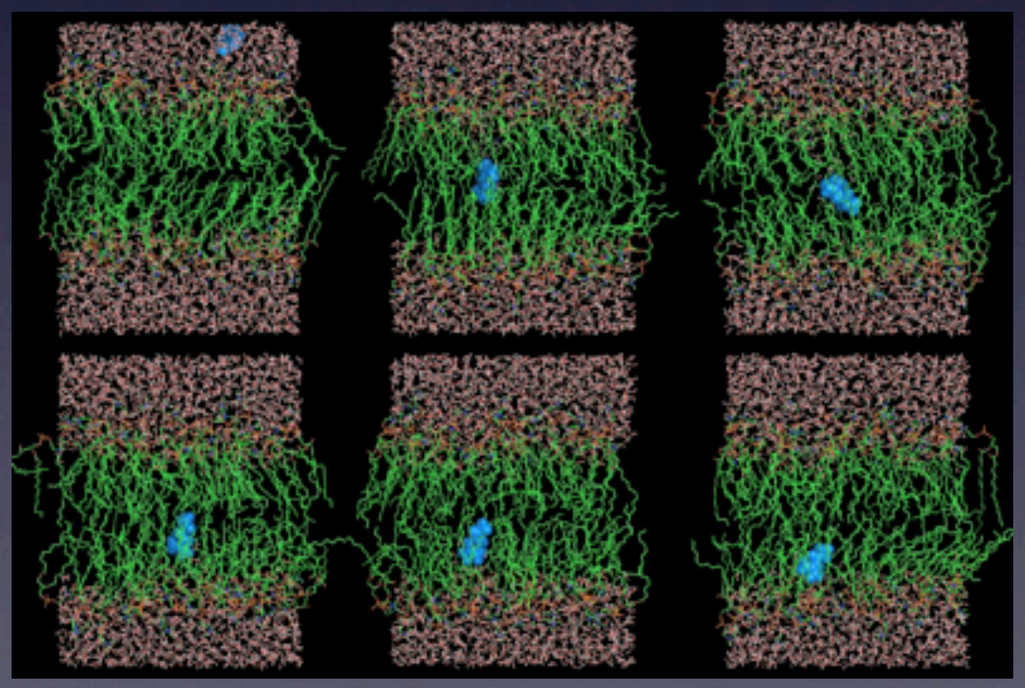
Center for Biomembrane Research
Stockholm University



Molecular Dynamics on CPUs

MD on GPUs
The good, the bad, and the ugly

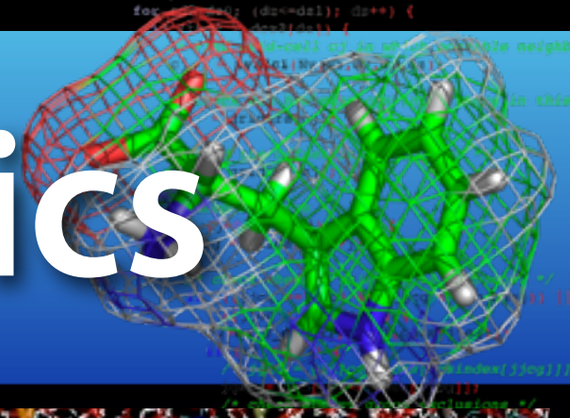
Should we change the way we do HPC?

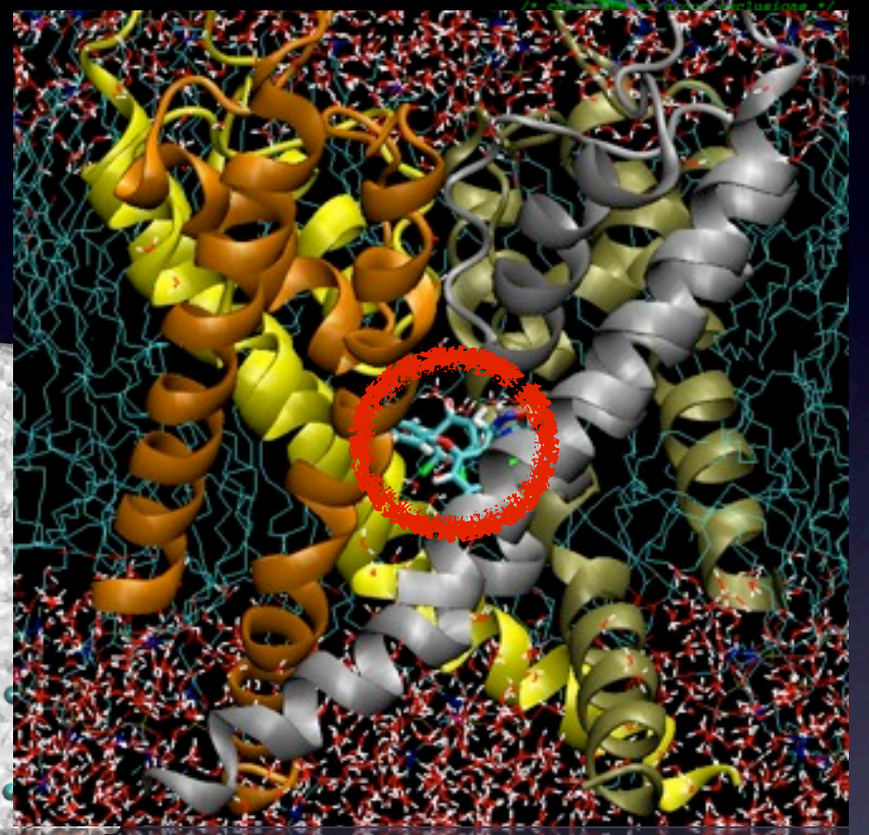


Ensemble simulations

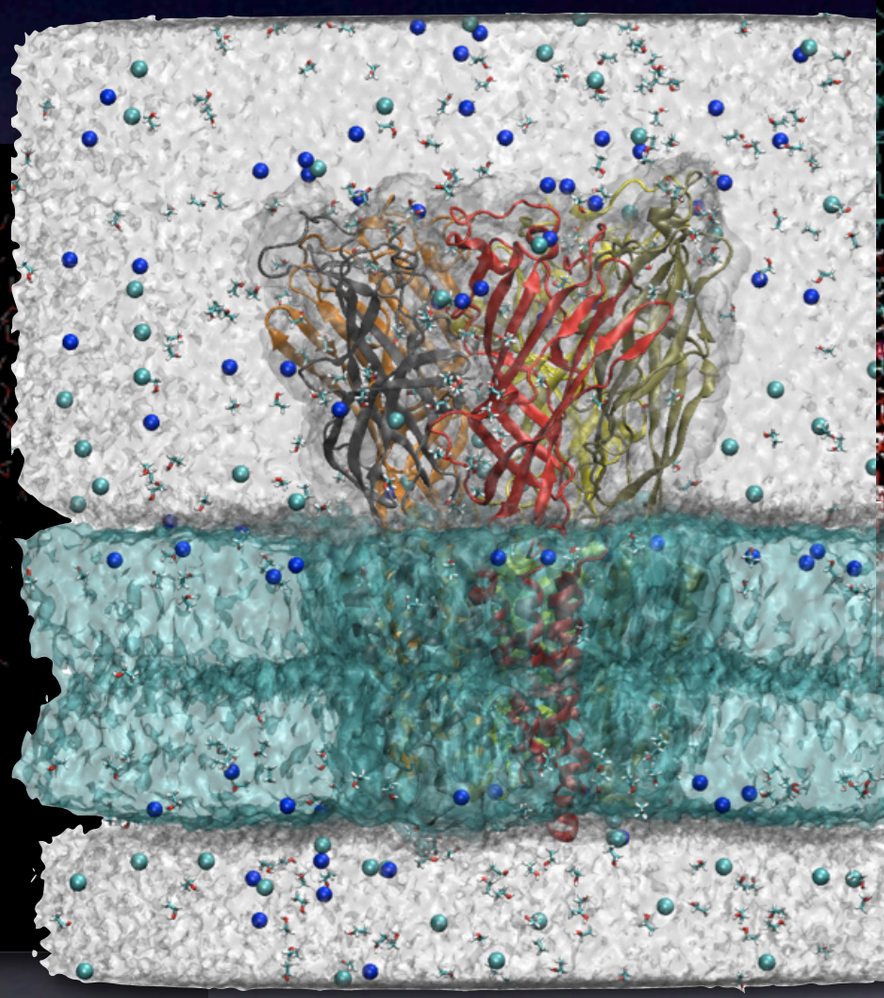
Molecular Dynamics on traditional CPUs

Biomolecular Dynamics

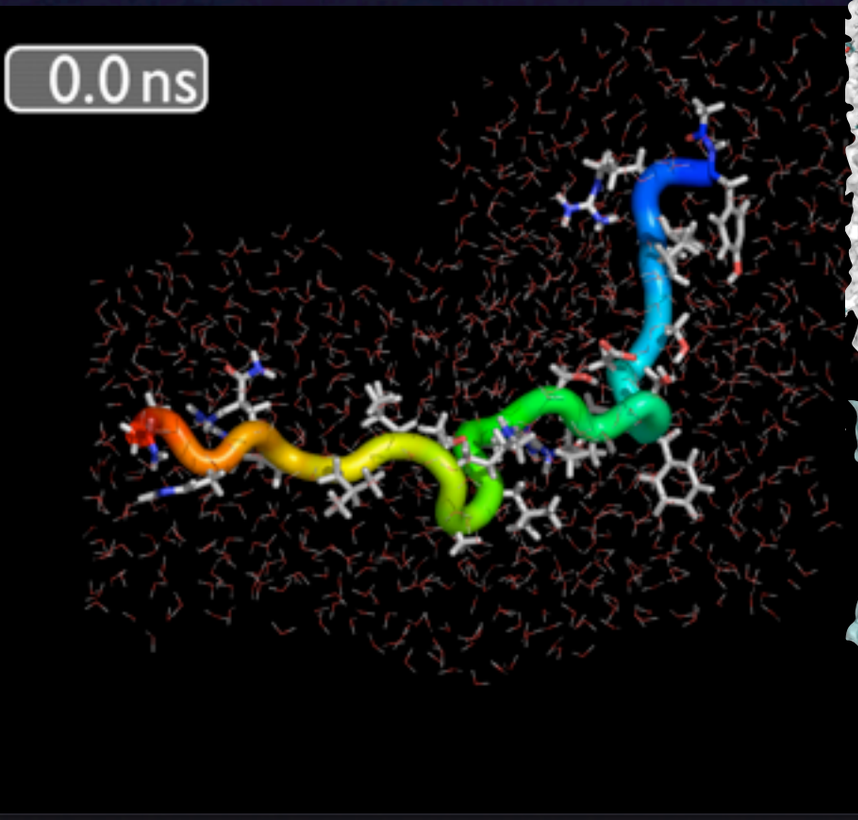




Free Energy &
Drug Design

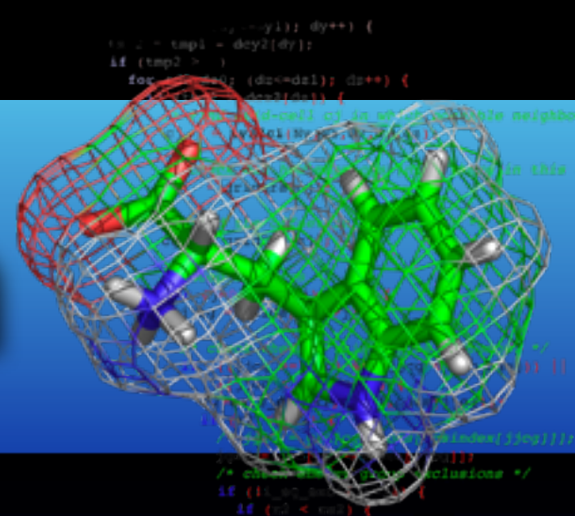


Understand biology



Protein Folding

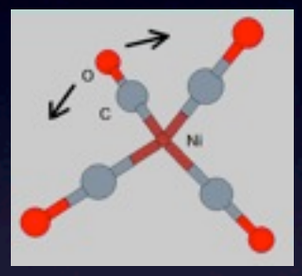
Timescales of Motion

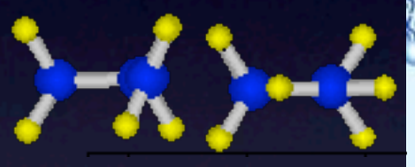


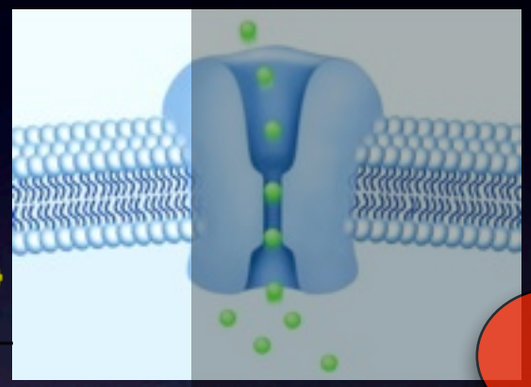
Experiments

Efficient averaging

Less detail







Where we need to be

Physics

Chemistry

Biology

10^{-15}s 10^{-12}s 10^{-9}s 10^{-6}s 10^{-3}s 10^{0}s 10^{3}s

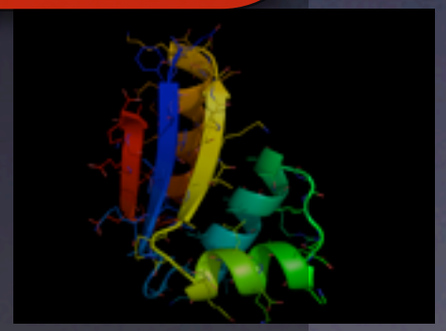
Simulations

Extreme detail

Sampling issues?

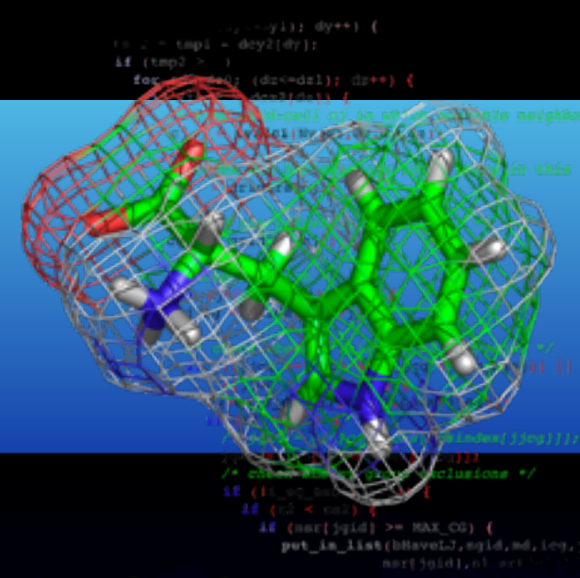
Parameter quality?

Where we are



Where we want to be

Modeling levels



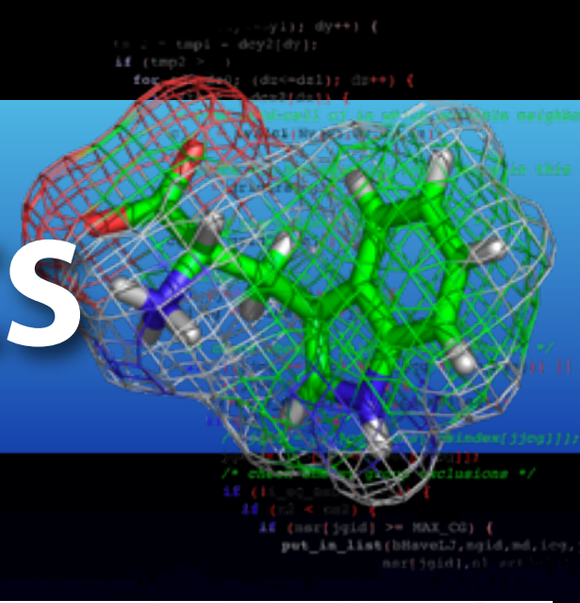
- QM
- Car-Parinello dynamics
- Molecular mechanics simulations
- Mesoscopic models

Detailed

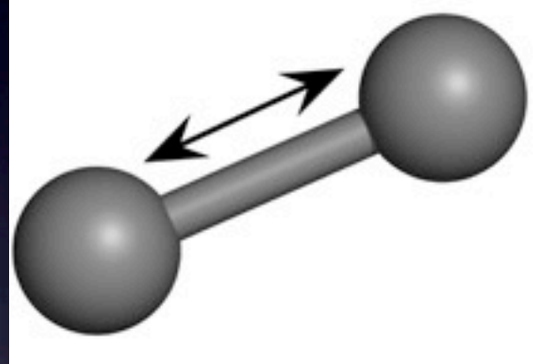
Better sampling

- Which is more important?
  - More detailed model
  - Better phase-space sampling

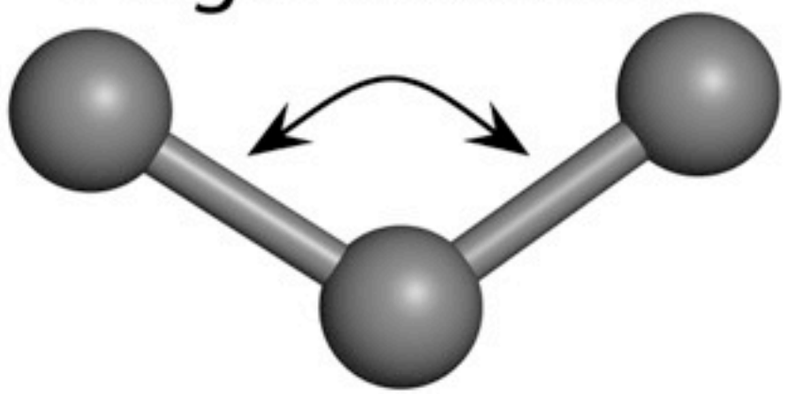
Molecular Mechanics



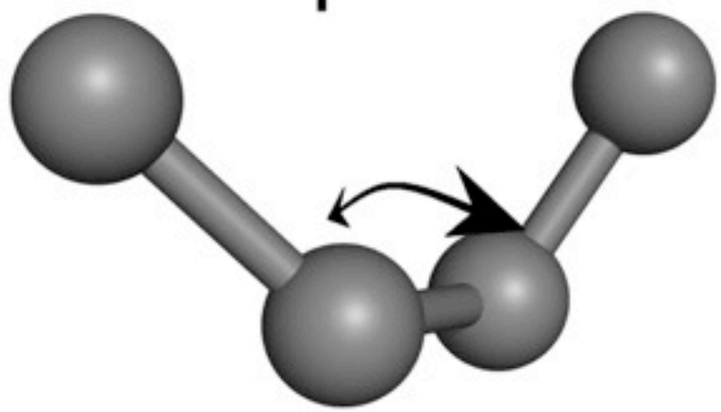
Bond vibration



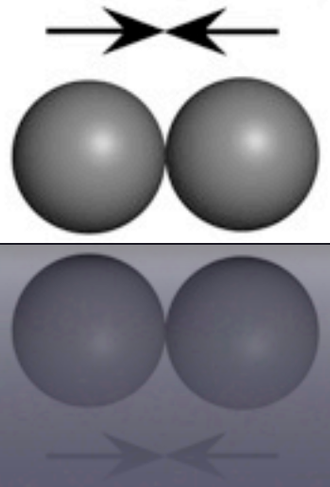
Angle vibration



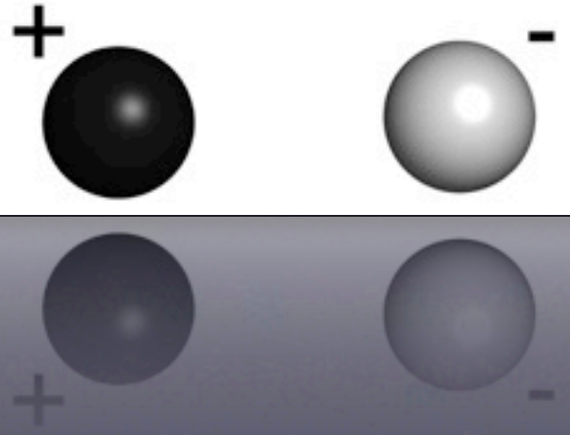
Torsion potentials



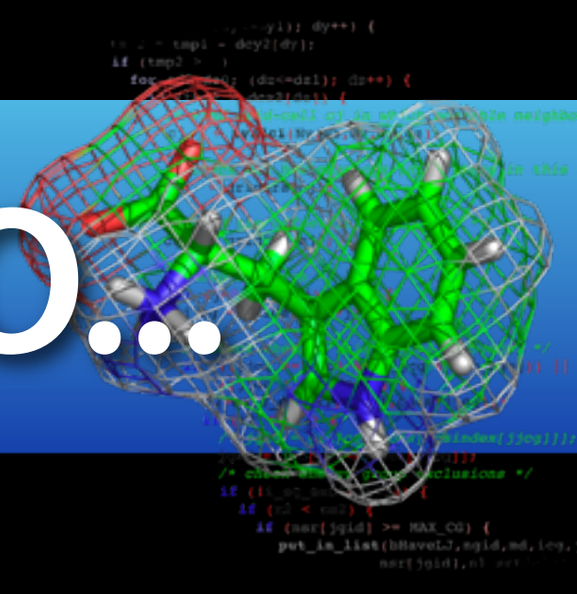
van der Waals interactions



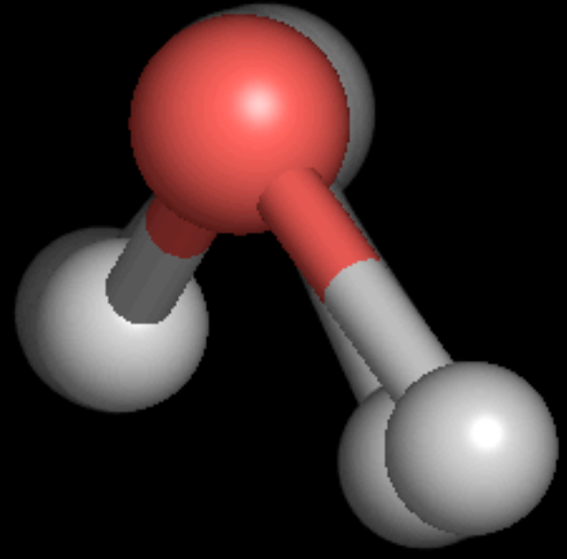
Electrostatics



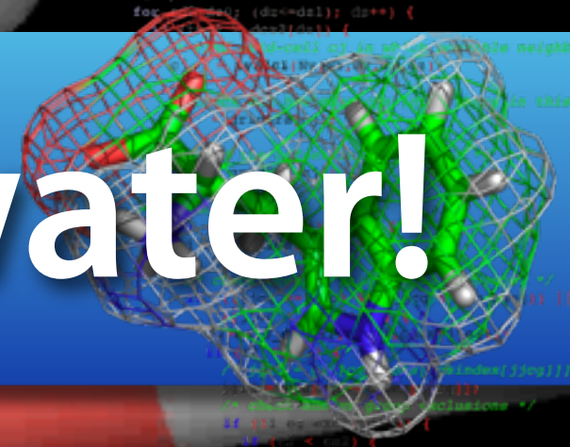
One small step for H_2O...



8 fs

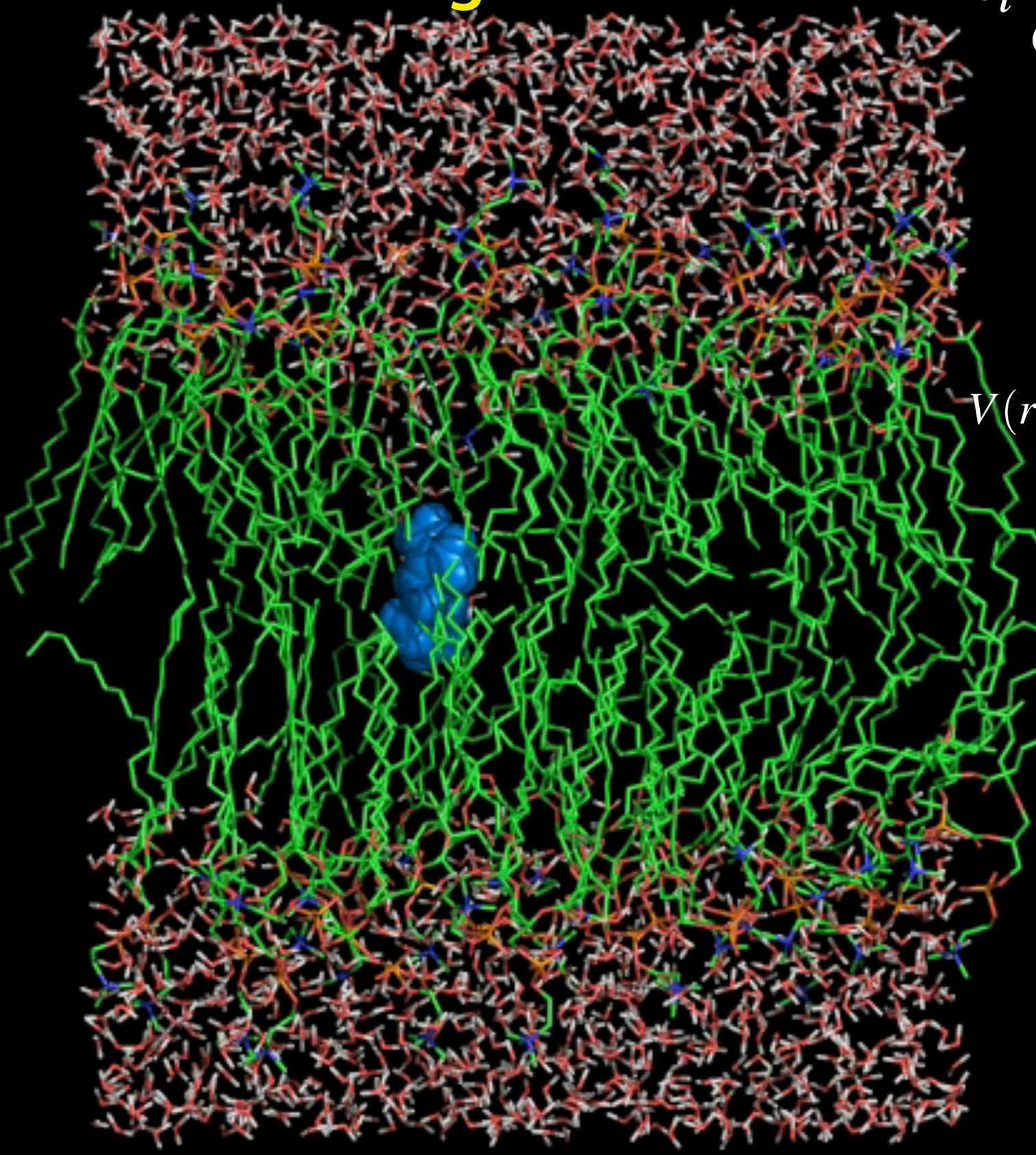


...one giant leap for water!



10 ps!

Most interesting systems are not homogenous...



$$m_i \frac{\partial^2 r_i}{\partial t^2} = F_i \qquad i = 1..N$$

$$F_i = -\frac{\partial V(r)}{\partial r_i}$$

$$V(r) = \sum_{bonds} \frac{1}{2} k_{ij}^{b} \left(r_{ij} - r_{ij}^{0}\right)^2$$

$$+ \sum_{angles} \frac{1}{2} k_{ijk}^{\theta} \left(\theta_{ijk} - \theta_{ijk}^{0}\right)^2$$

$$+ \sum_{torsions} \left\{ \sum_{n} k_{\theta} \left[1 + \cos\left(n\phi - \phi_0\right)\right] \right\}$$

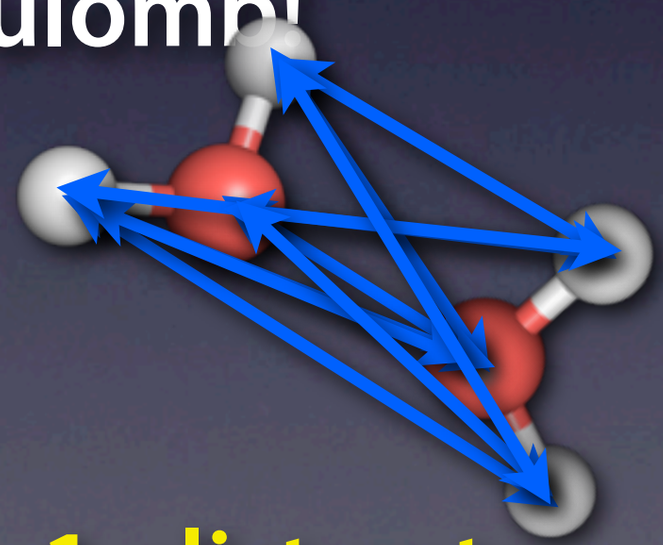
$$+ \sum_{impropers} k_{\xi} \left(\xi_{ijkl} - \xi_{ijkl}^{0}\right)$$

$$+ \sum_{i,j} \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}}$$

$$+ \sum_{i,j} \left[\frac{C_{12}}{r_{ij}^{12}} - \frac{C_6}{r_{ij}^{6}} \right]$$

20 years of CPU Optimization

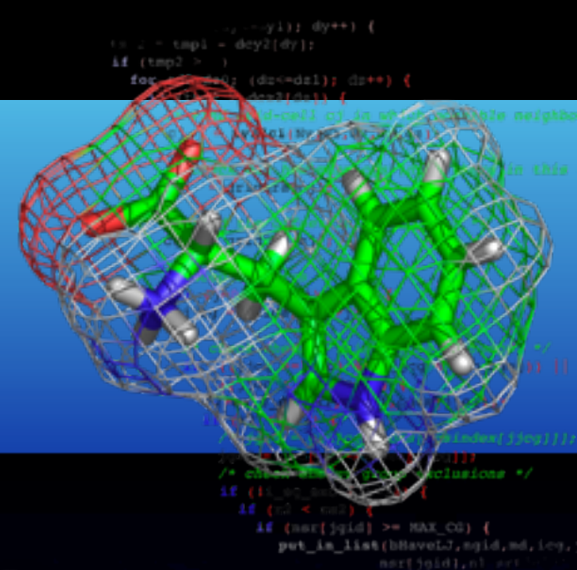
- Single precision when possible *"Save FLOPS"*
- Handtuned 1/sqrt(x) instructions
- Single-instruction Multiple-Data
- Don't calculate zeros
  - No charge on an atom? Don't do coulomb!
  - Interaction specific kernels
  - Complex neighbor lists
- Fancy algorithms to extend timestep
  - Bond constraints, virtual sites, etc.



1 nlist entry
9 interactions

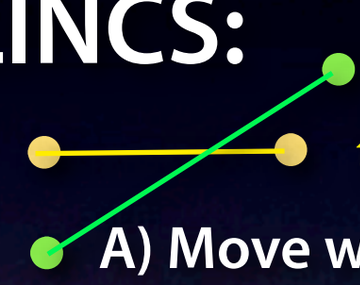
Stuff that might be hard to port to Cuda (1):

Constraints

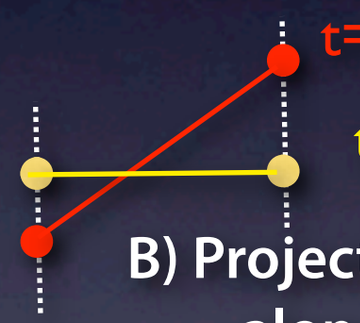


- **Δt limited by fast motions - 1fs**
  - **Remove bond vibrations**
- **SHAKE (iterative, slow) - 2fs**
  - **Problematic in parallel (won't work)**
  - **Compromise: constrain h-bonds only - 1.4fs**
- **GROMACS (LINCS):**
  - **LINear Constraint Solver**
  - ***Approximate* matrix inversion expansion**
  - **Fast & stable - much better than SHAKE**
  - **Non-iterative**
  - **Enables 2-3 fs timesteps**
  - **Parallelizes (in theory at least)**

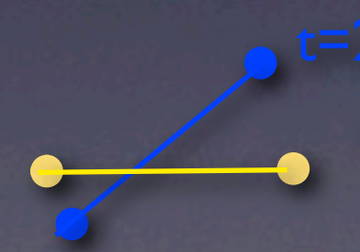
LINCS:



A) Move w/o constraint



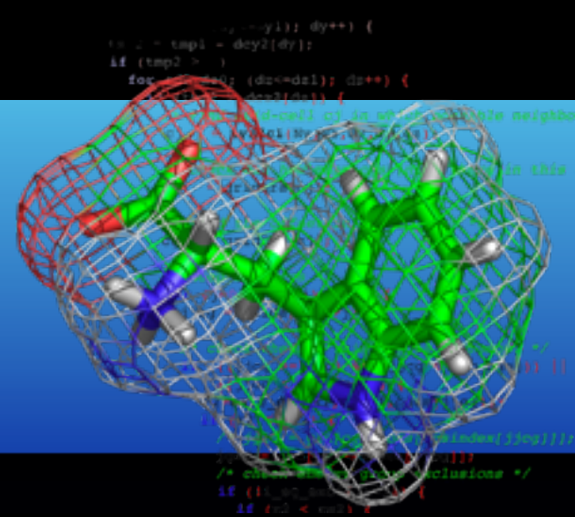
B) Project out motion along bonds



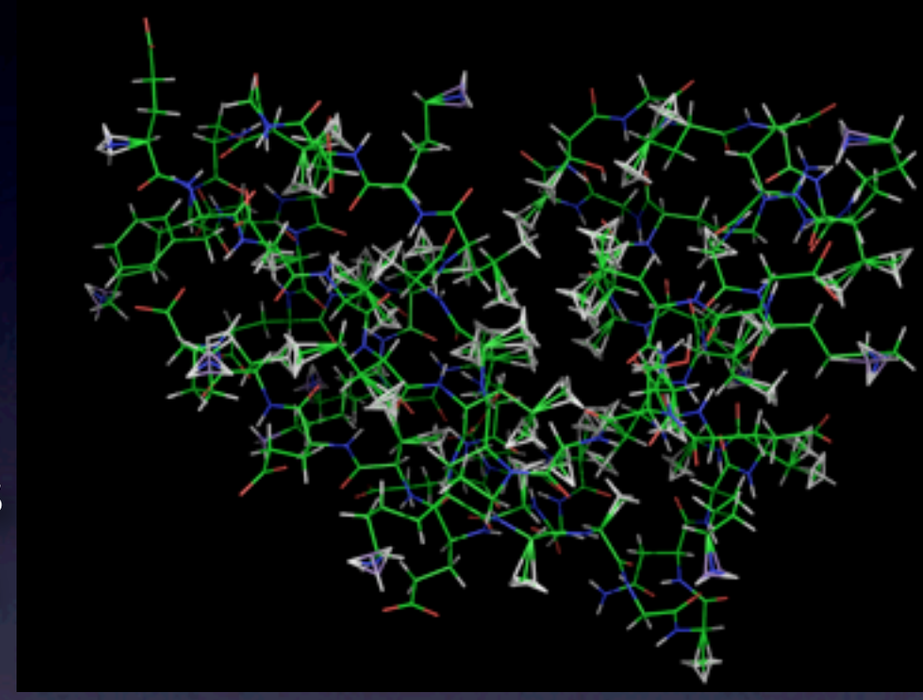
C) Correct for rotational extension of bond

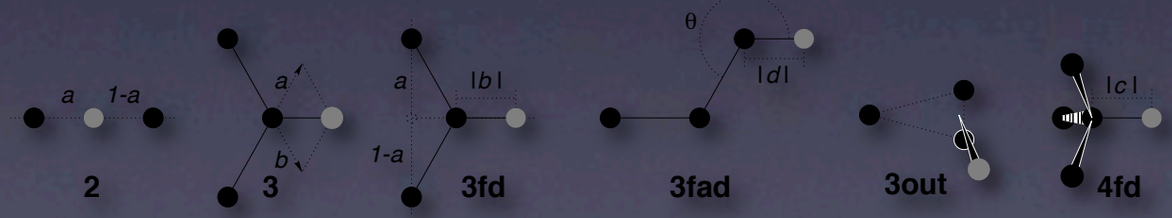
Stuff that might be hard to port to Cuda (2):

Virtual sites

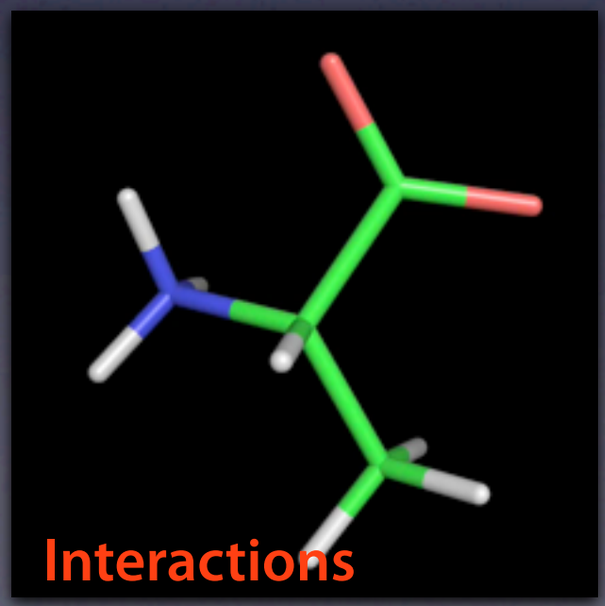


- Next fastest motions is H-angle and rotations of CH_3/NH_2 groups
- Try to remove them:
  - Ideal H position from heavy atoms.
  - CH_3/NH_2 groups are made rigid
  - Calculate forces, then project back onto heavy atoms
  - Integrate only heavy atom positions, reconstruct H's
- Enables 5fs timesteps!

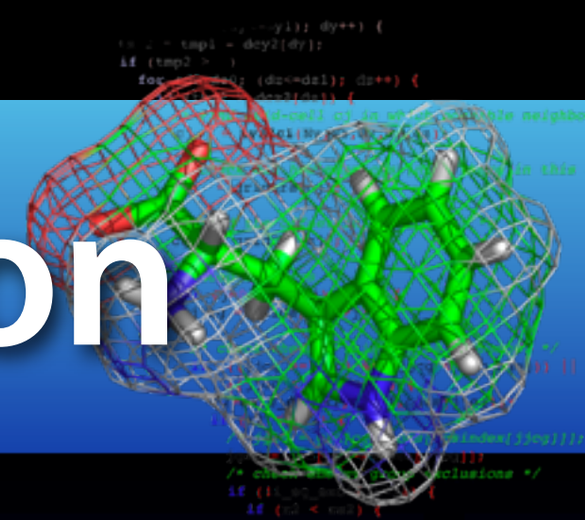


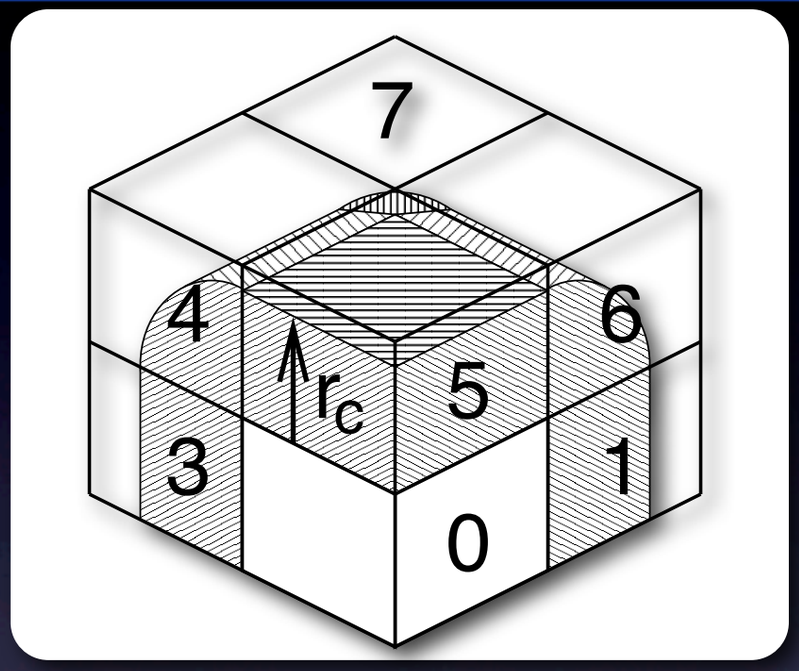


Problem-specific optimization: use our knowledge of chemistry

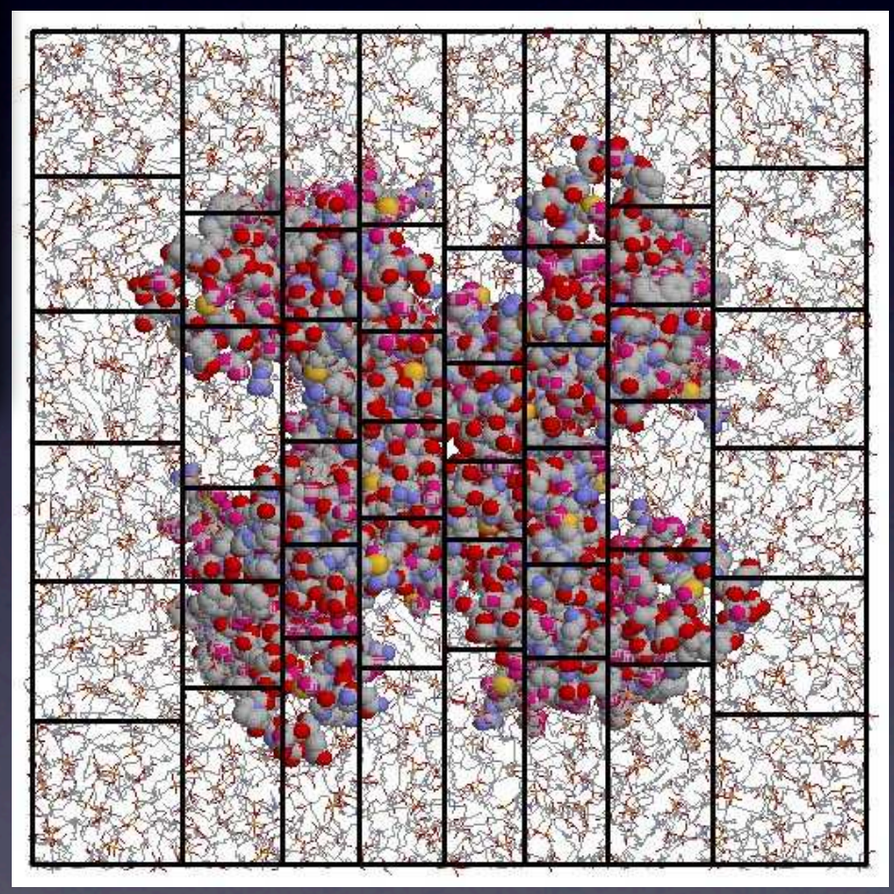


CPU Parallelization





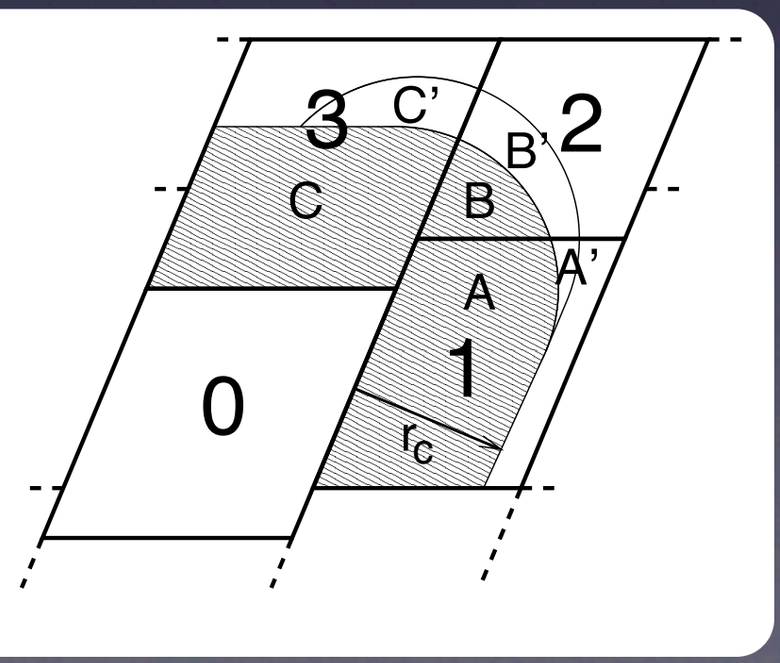
Domain decomposition



CPU limit today:
50-150 ns/day

Hard to decrease
 # atoms / CPU

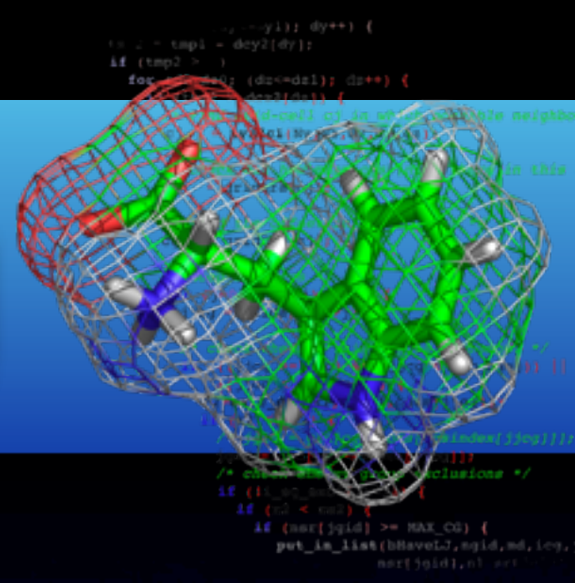
Complex
Load balancing



Lots of book-keeping

Molecular Dynamics on Stream Processors

Stream computing



Our first cards were less fancy!

First Gromacs GPU project in 2002
with Ian Buck & Pat Hanrahan, Stanford

Promise of theoretical high FP performance on GeForce4
Severe limitations in practice...

Lessons:

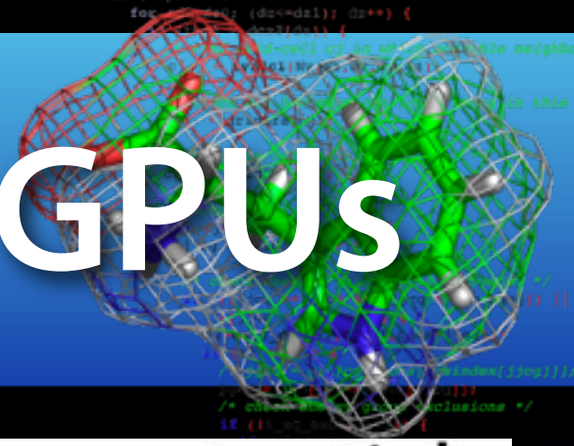
It's easy to achieve speedup relative to a slow reference implementation

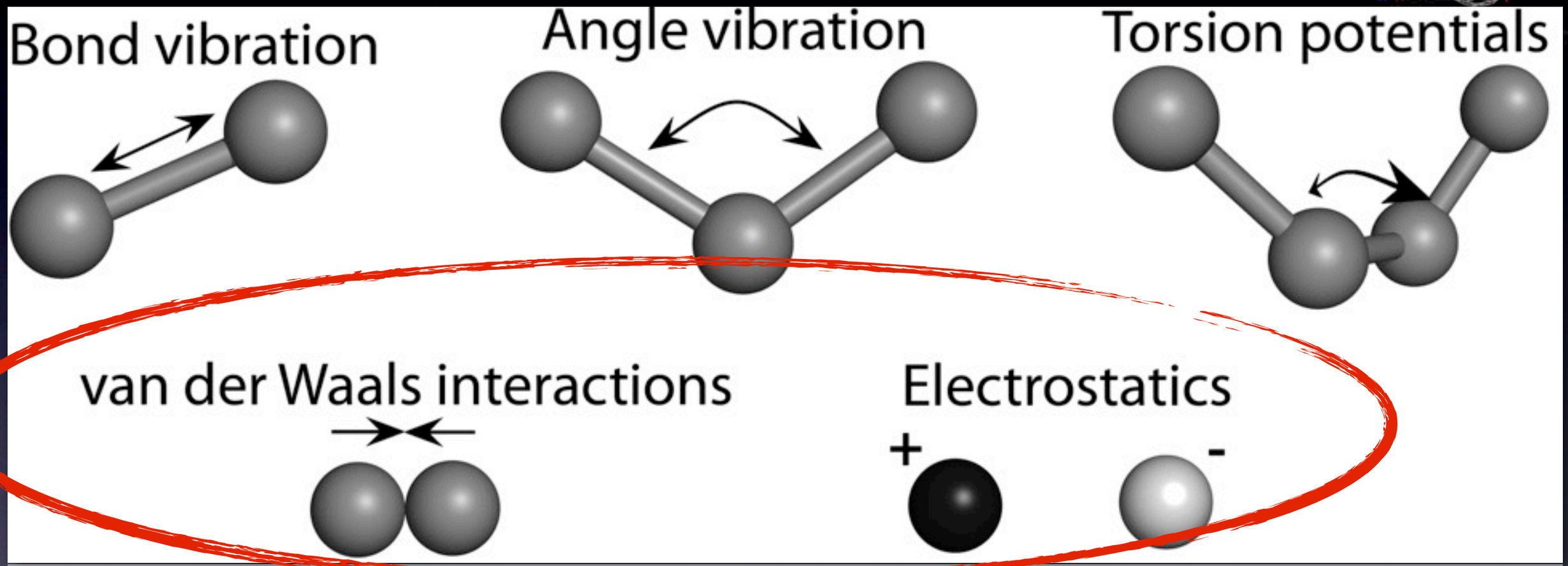
Much harder to beat well-optimized CPU code that can use >1 core

Don't write CPU code for GPUs

Find new optimizations instead

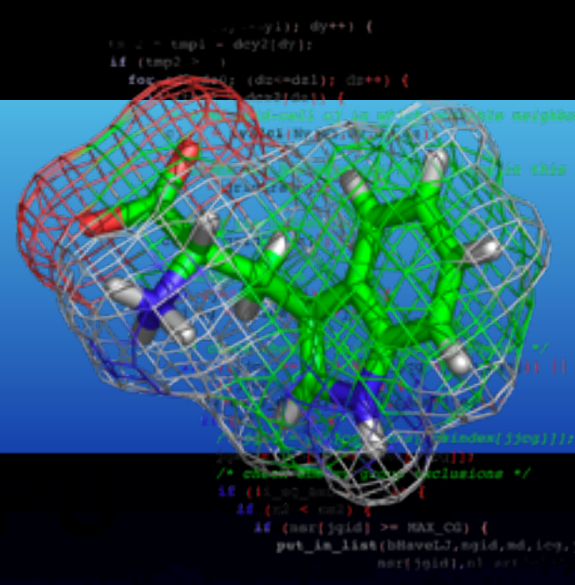
Molecular Dynamics on GPUs

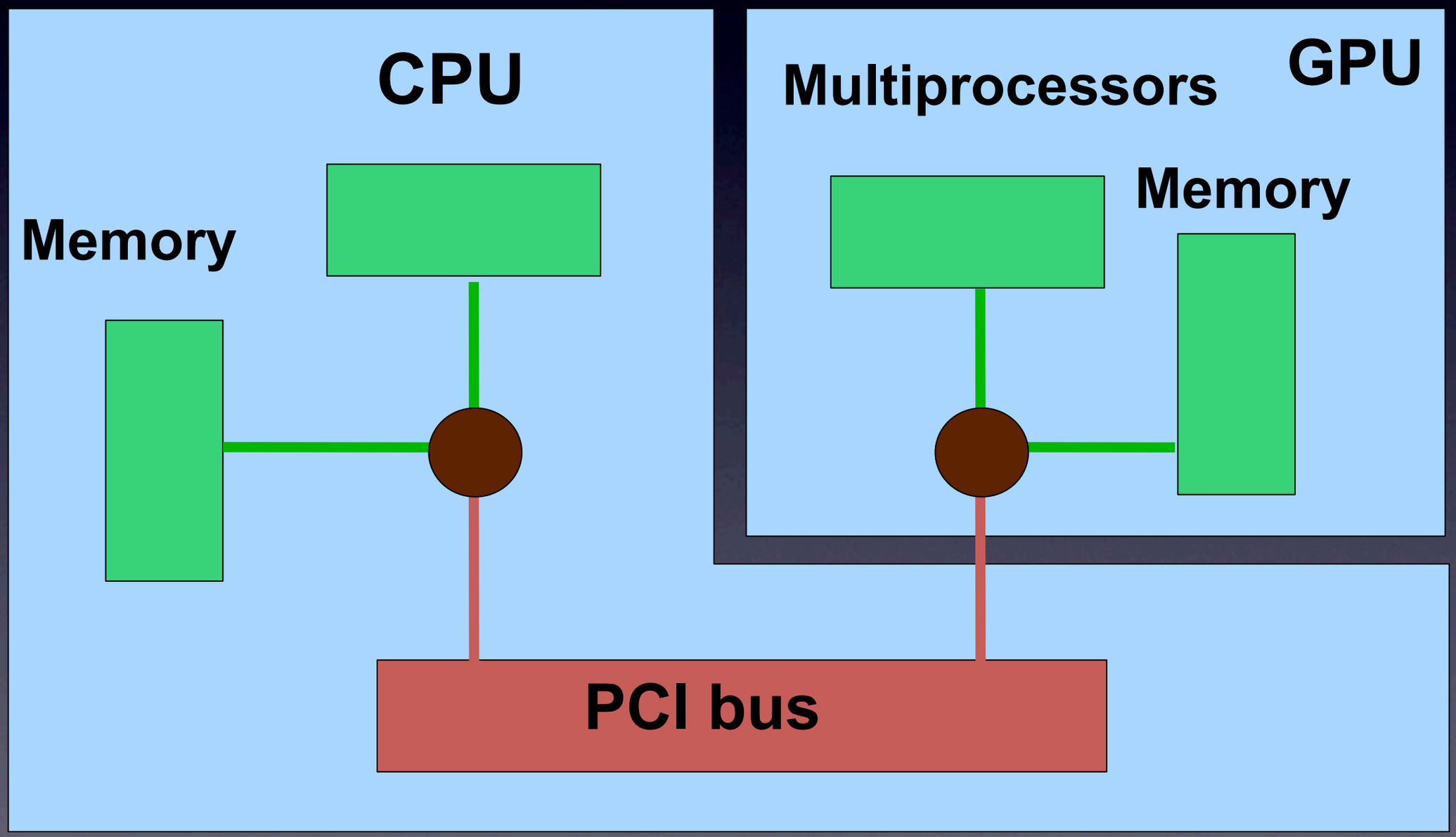




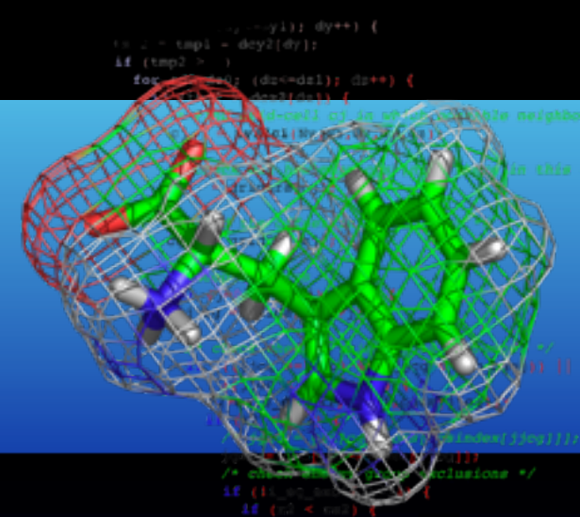
Focus on the code where we spend all the cycles on the CPU

Bottlenecks

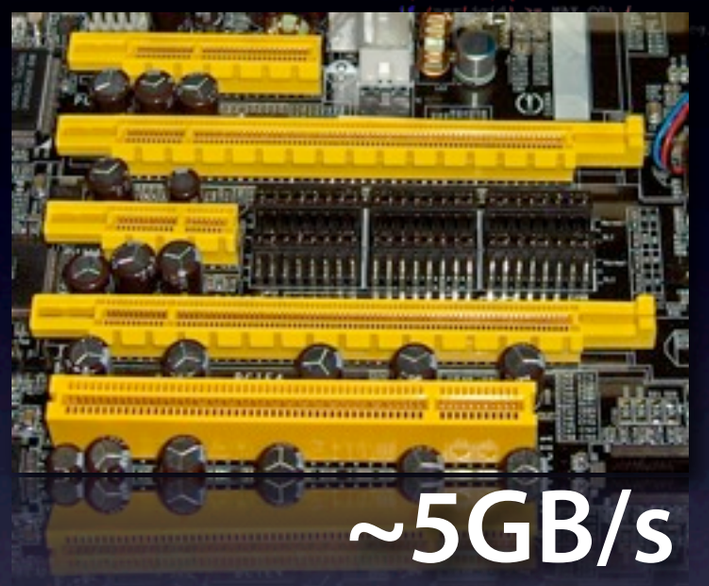




OpenMM



Not sufficient to accelerate nonbonded interactions - need to send to/from GPU



Need to do entire simulation on GPU?

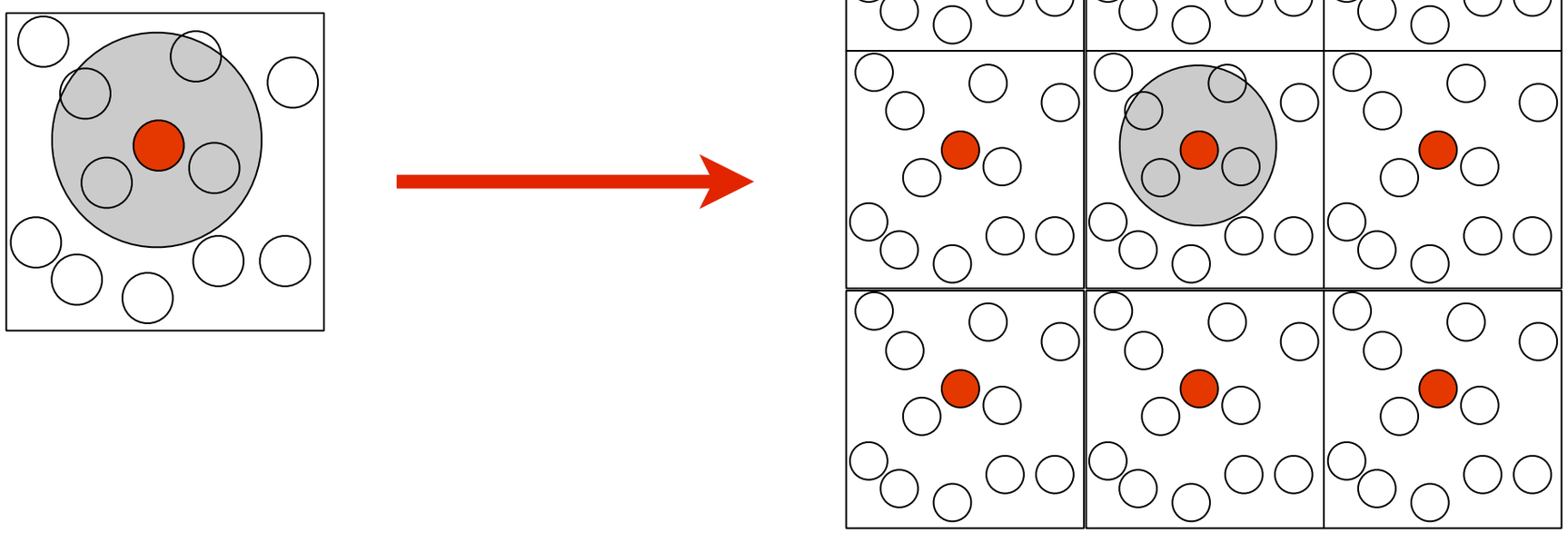
Not fun to rewrite 2M lines-of-code in a separate CUDA-Gromacs...

OpenMM: Core MD functionality in separate library
Stanford (Pande), Stockholm (Us), Nvidia & AMD

Fully public API, hardware-agnostic, use anywhere

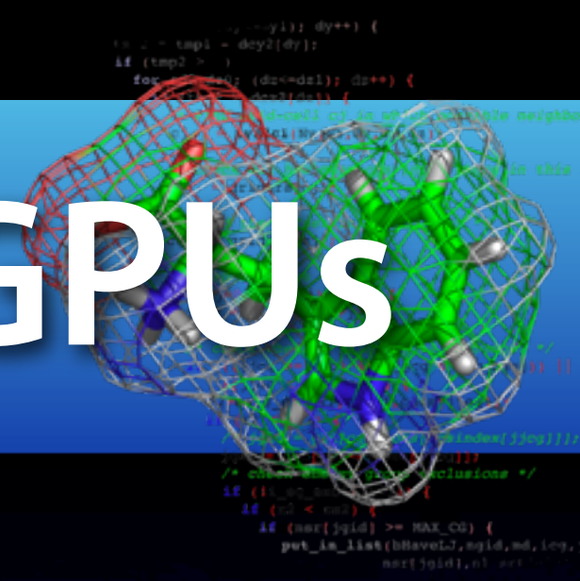
Peter Eastman, Scott Legrand

Nonbonded Interactions



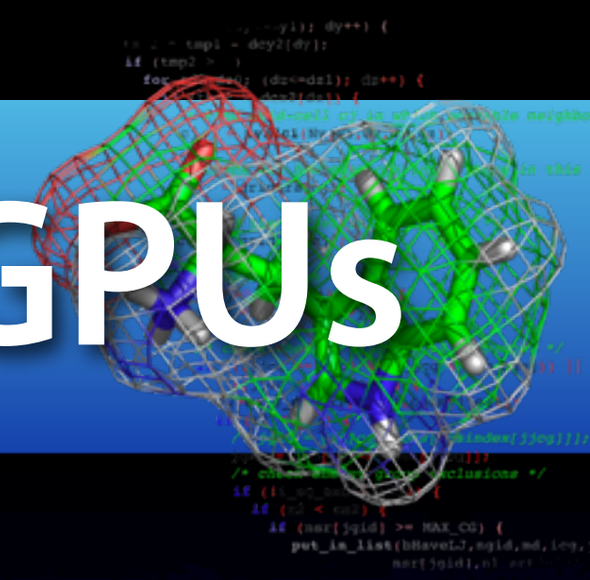
- Divide into short-range / long-range
- Calculate short-range analytically
- Use approximations for the long-range

O(N²) Algorithm on GPUs

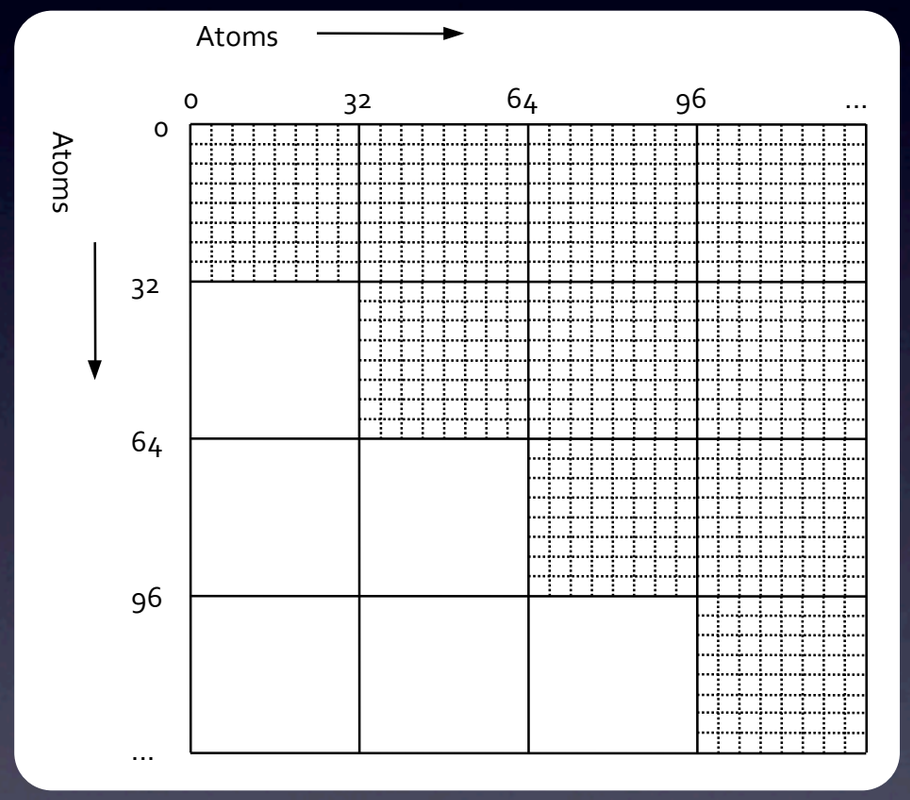


- An efficient algorithm should:
- minimize global memory access
- avoid thread synchronization
- take advantage of symmetry

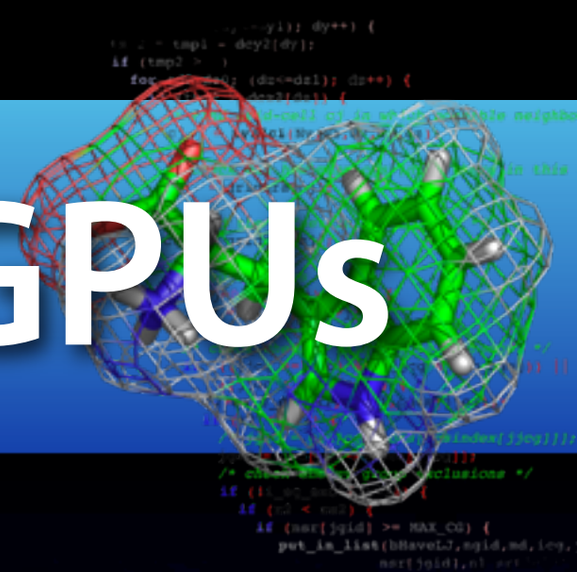
O(N²) Algorithm on GPUs



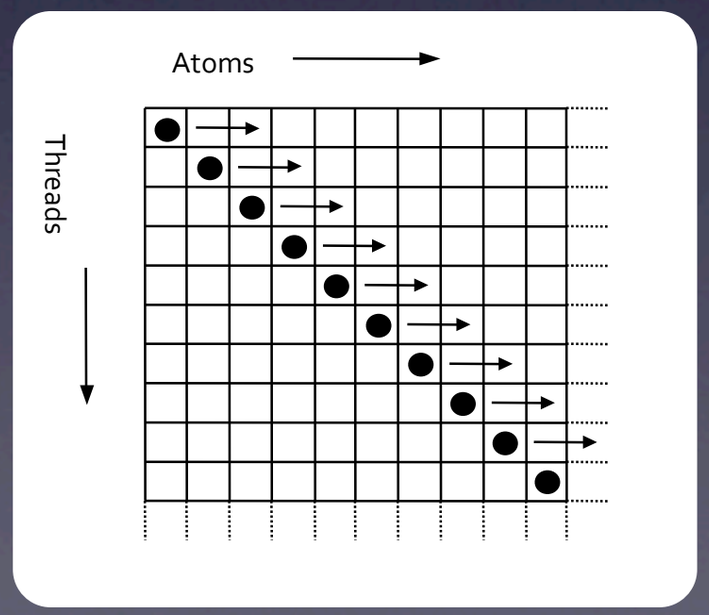
- Group atoms into blocks of 32
- Interactions divide into 32x32 tiles
- Each tile is processed by a group of 32 threads
  - Load atom coordinates and parameters into shared memory
  - Each thread computes interactions of one atom with 32 atoms
  - Use symmetry to skip half the tiles



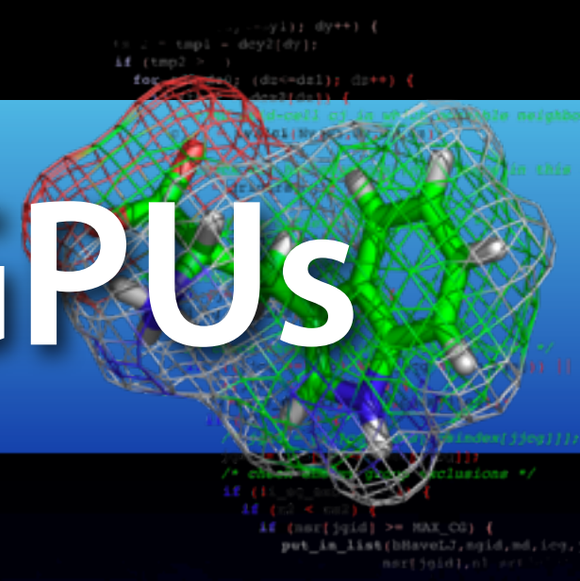
O(N²) Algorithm on GPUs



- Each thread loops over atoms in a different order
  - Avoids conflicts between threads
- No explicit synchronization needed
  - Threads in a warp are always synchronized



O(N) Algorithm on GPUs



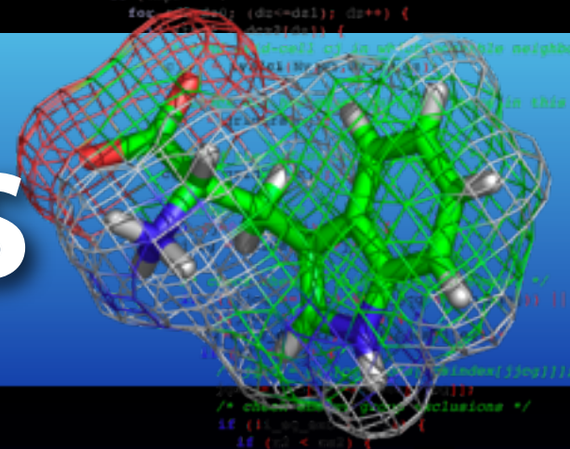
Hard. Why?

- Traditional O(N) methods (e.g. neighbor lists) slow on GPUs
  - Out of order memory access

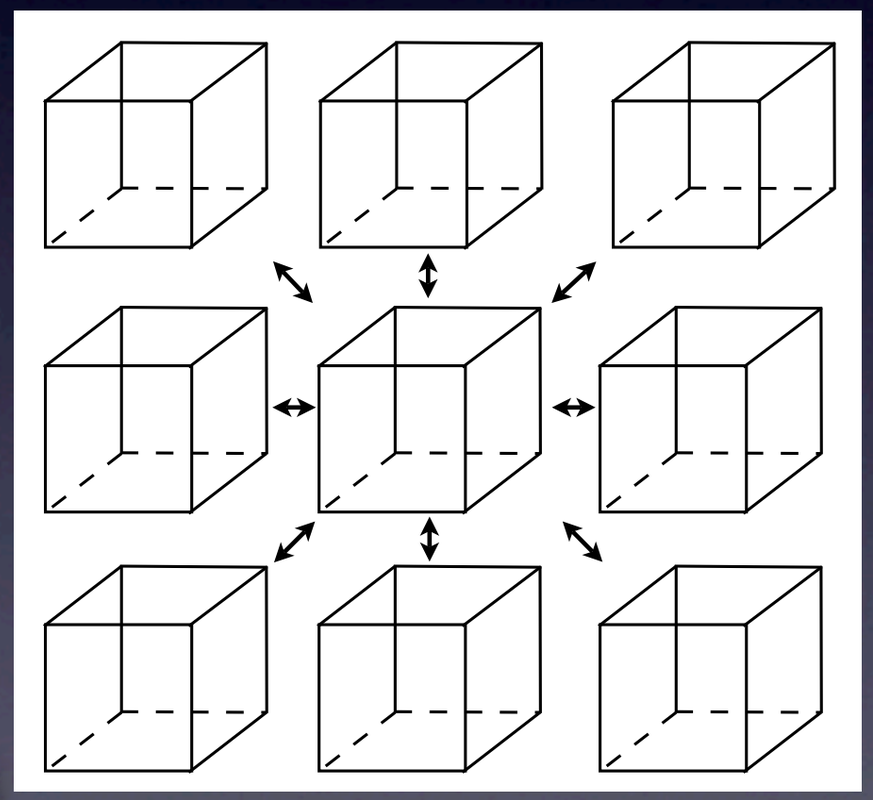
```
for i = 1 to numNeighbors                          slow!
   load coordinates and parameters for neighbor[i]
   compute force                          slow!
   store force for neighbor[i]
```

- Inner loop contains non-coalesced memory access

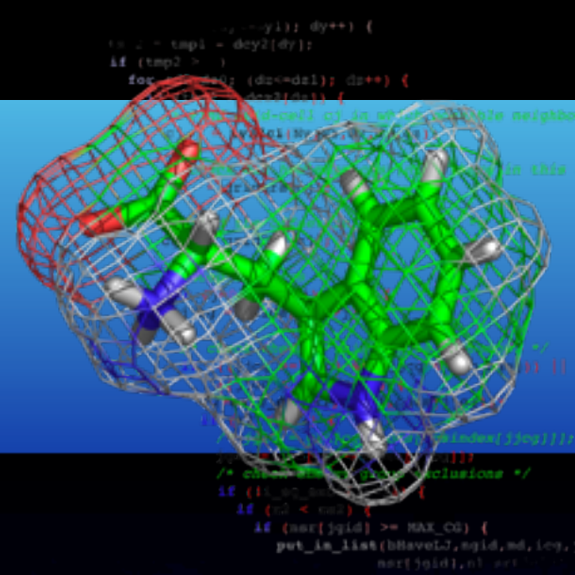
Approach 1: Voxels



- Divide space into smaller voxels
- Compute all-vs-all interactions between adjacent voxels
- LAMMPS, NAMD
- Very efficient when all particles are identical
- Book-keeping can get expensive for complex systems



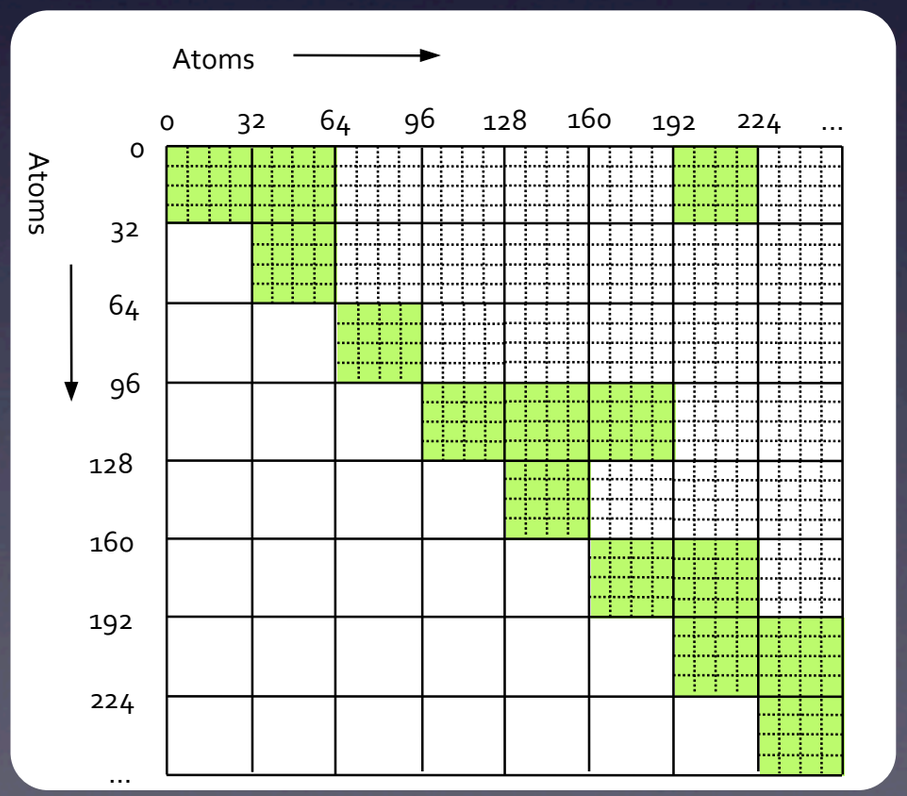
Approach 2: Tiles



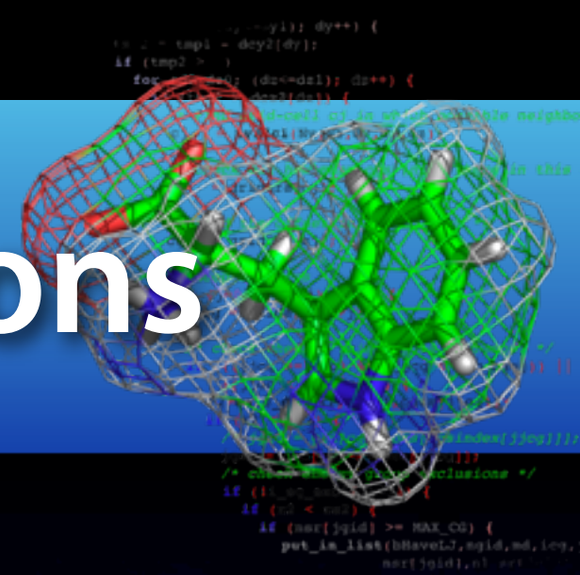
- Start with the O(N²) algorithm
- Exclude tiles with no interactions
  - Like a neighbor list between blocks of 32 atoms

Problem 1: How do you construct good tiles?

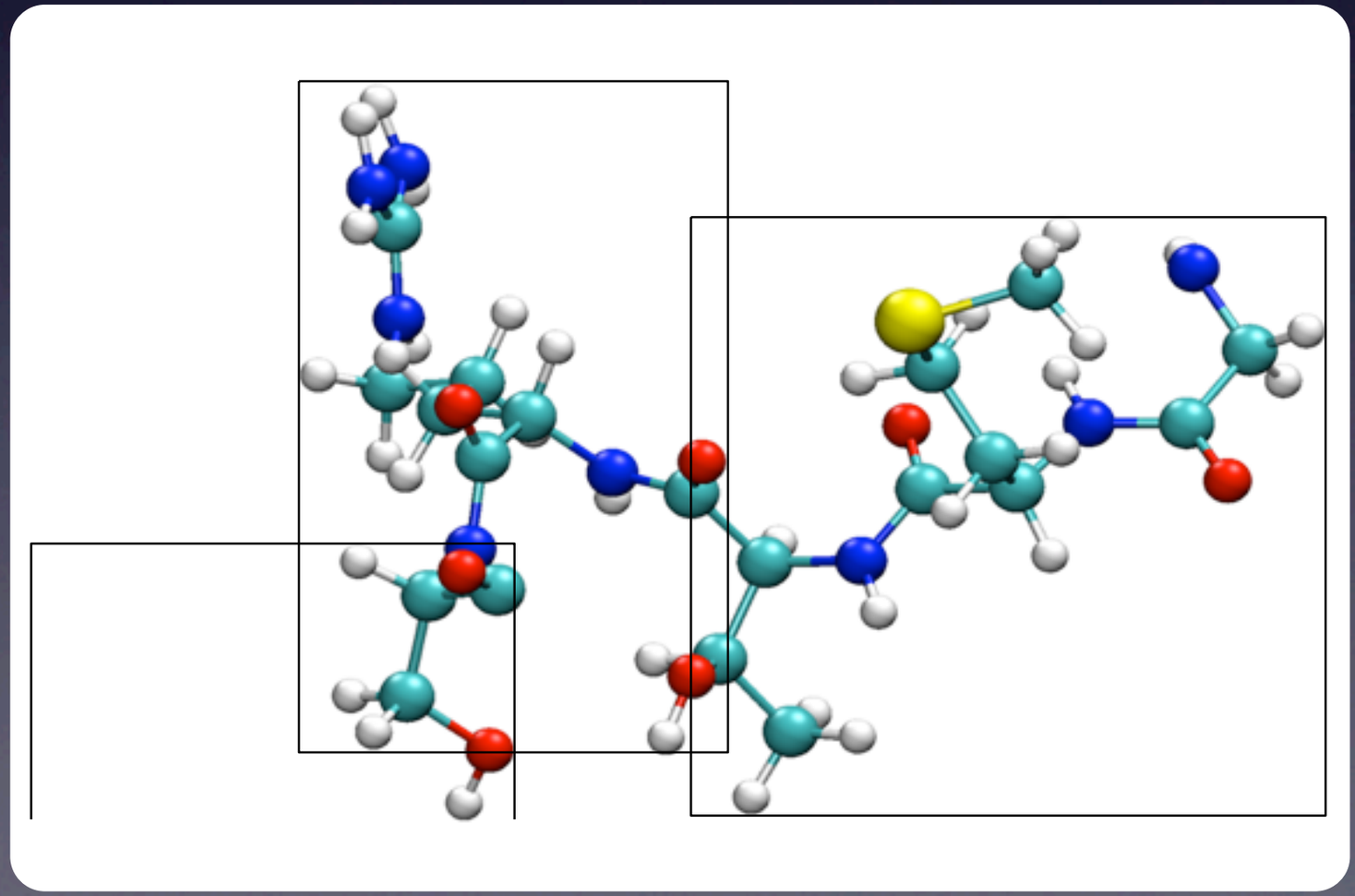
Problem 2: How do you keep them good?



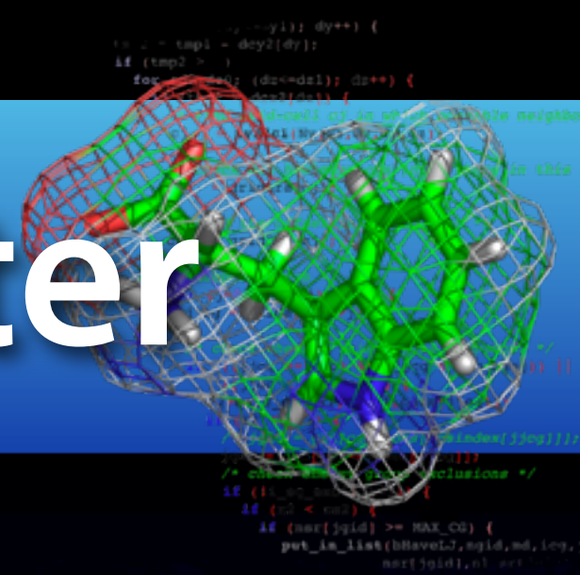
Finding tiles with interactions



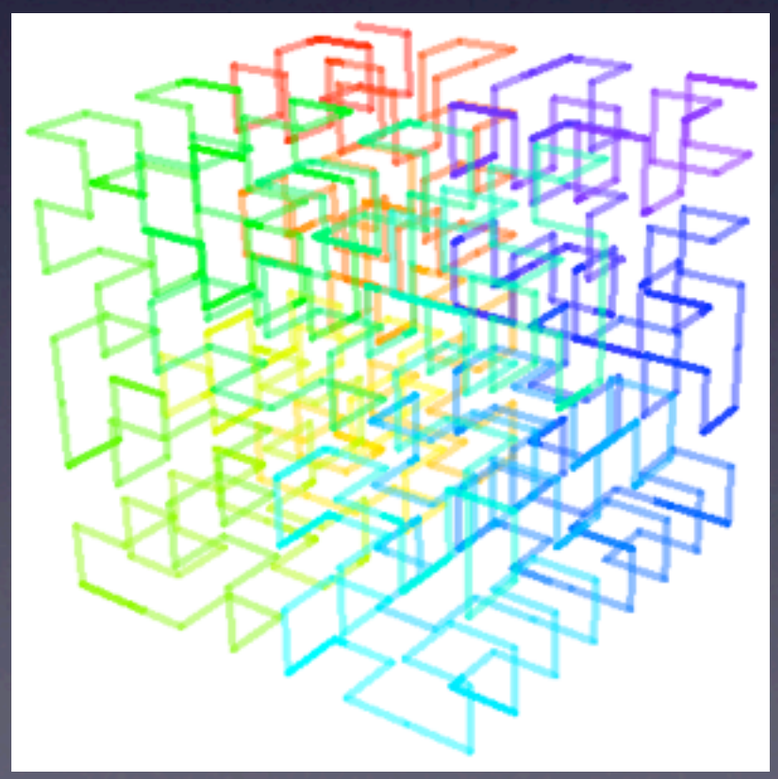
- Compute an axis aligned bounding box for the 32 atoms in each block
- Calculate the distance between boxes



Keeping track of water

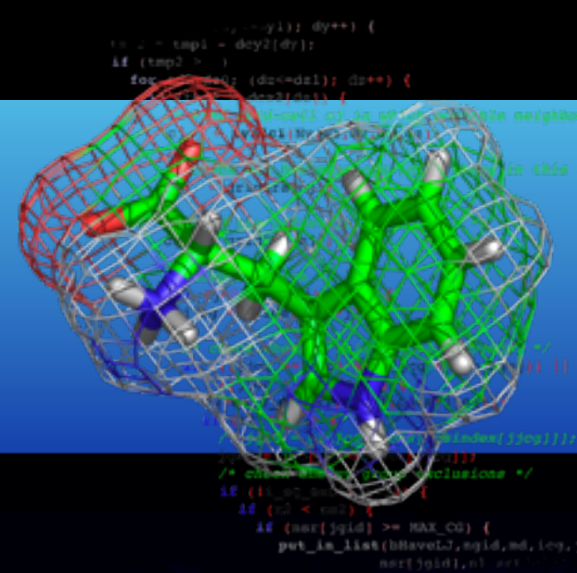


- Solvent molecules must be ordered to be spatially coherent
  - Or bounding boxes will be very large
- Arrange along a space filling curve
- Reorder every ~100 time steps



Swapping is easy: Same parameters!

Performance?

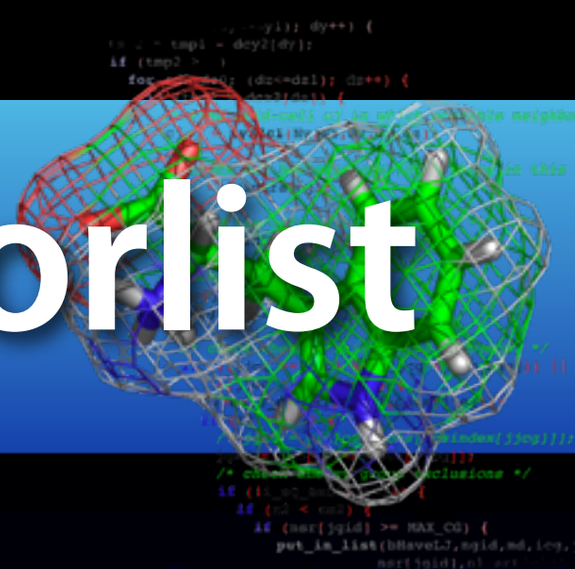


- Much faster than O(N²) for large systems
- Performance scales linearly

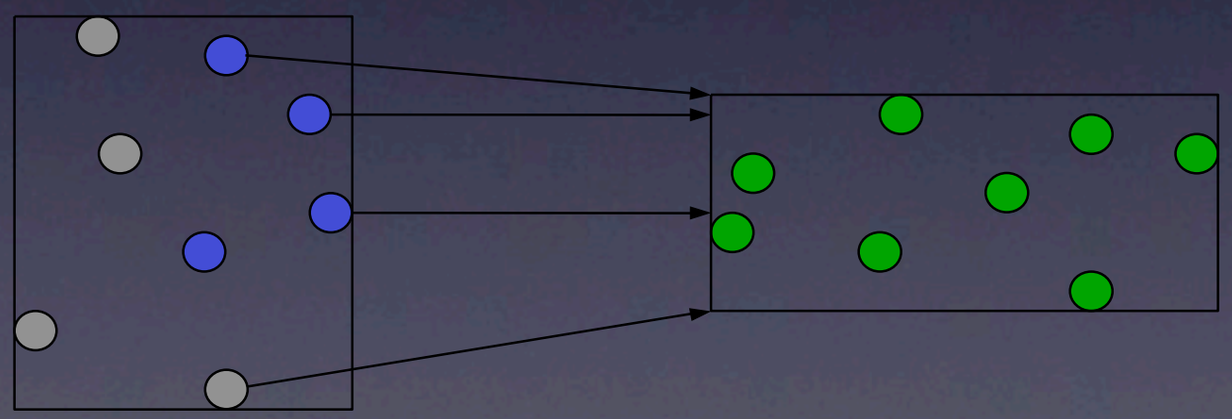
But

- Computes many more interactions than really required
  - Computes all 1024 interactions in a tile, even if few/none are within the cutoff

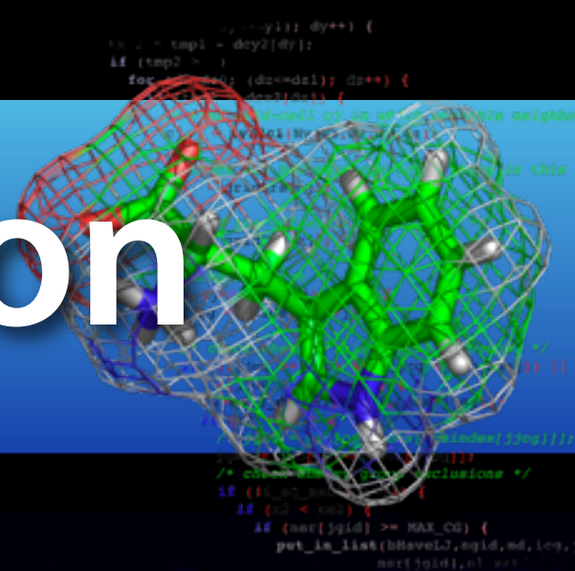
Finer-grained neighborlist



- For each tile with interactions:
  - Compute distance of each atom in one block from the bounding box of the other block
  - Set a flag for each atom



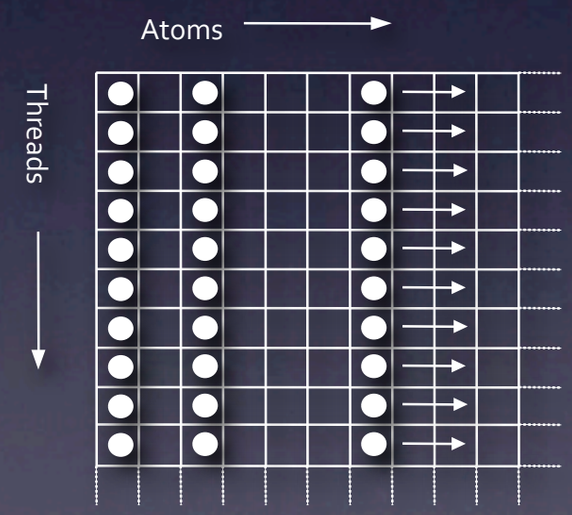
Tile Force Computation



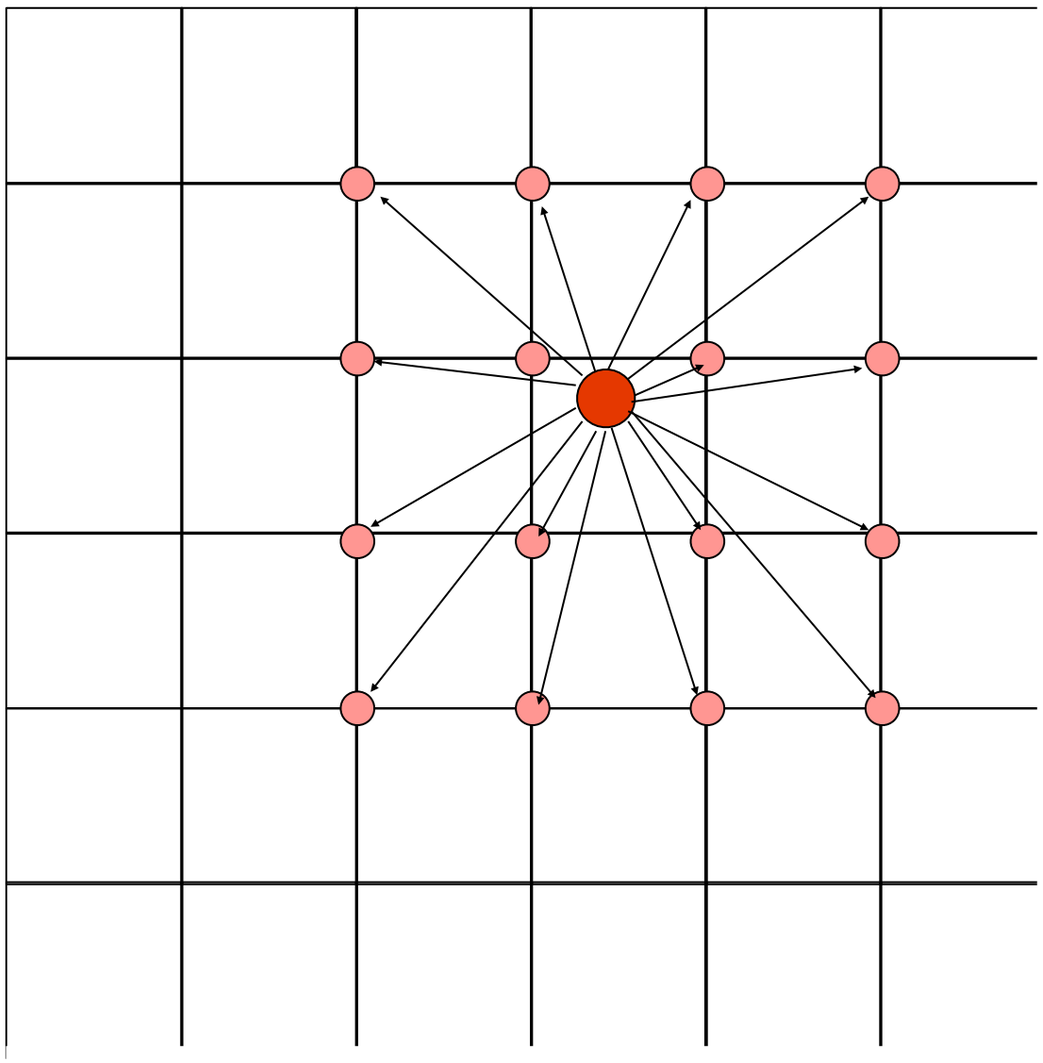
```
for i = 1 to 32
  if (hasInteractions[i])
    compute interaction with atom i
```

} on each thread

- All 32 threads must loop over atoms in the same order
  - Requires a reduction to sum the forces
  - For a few atoms, this is still much faster
  - For many atoms, better to just compute all interactions

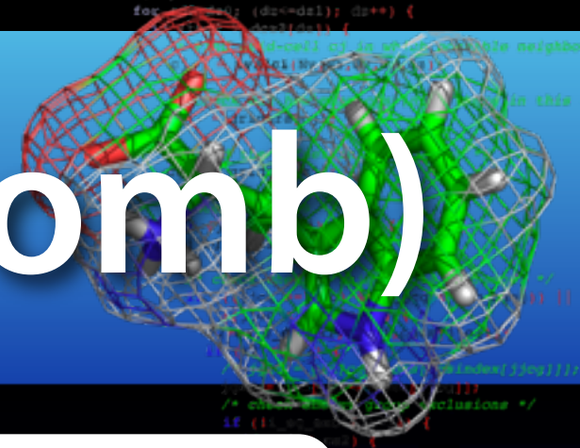


PME (Long range coulomb)

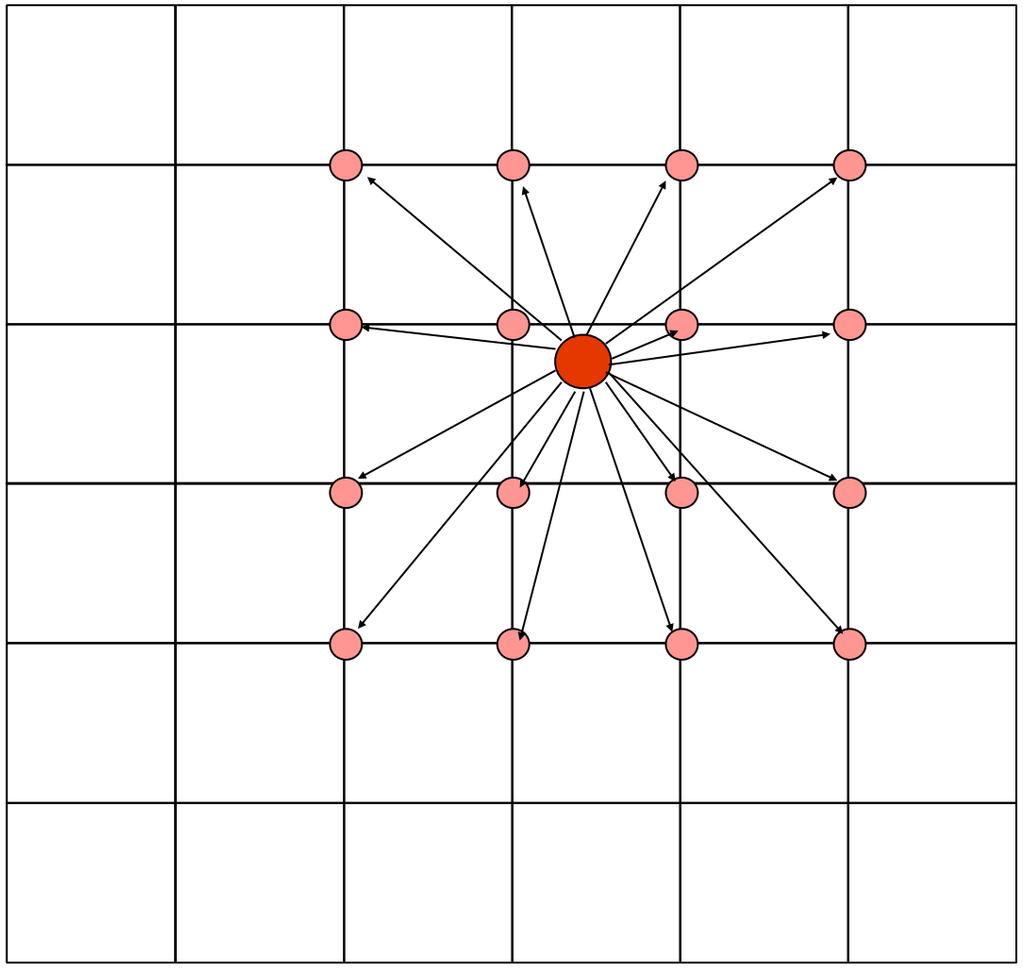


1. Calculate scaled fractionals
2. Calculate B-spline coefficient
3. “smear” the charges over the grid points from spline-coefficient
4. Execute forward FFT
5. Calculate the reciprocal energy
6. Execute backward FFT
7. Calculate force gradient

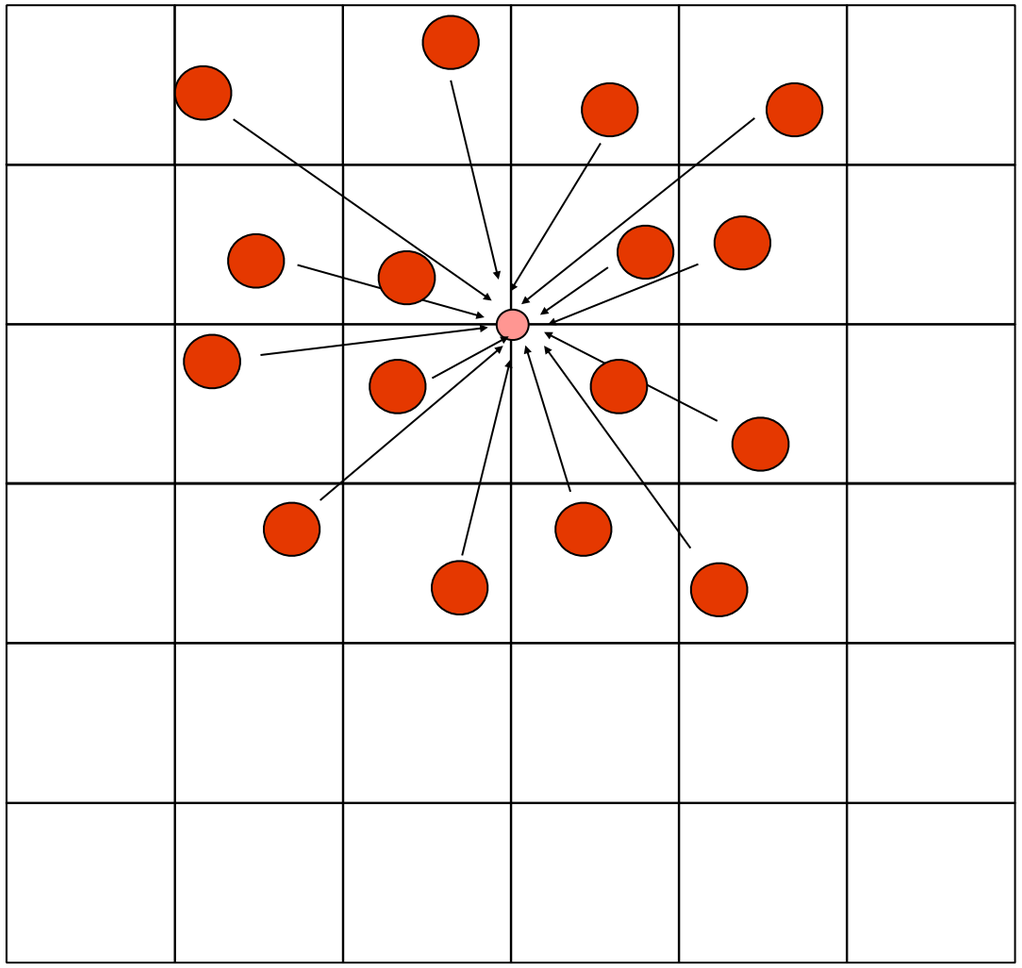
PME (Long range coulomb)



CPU
spreading of charges

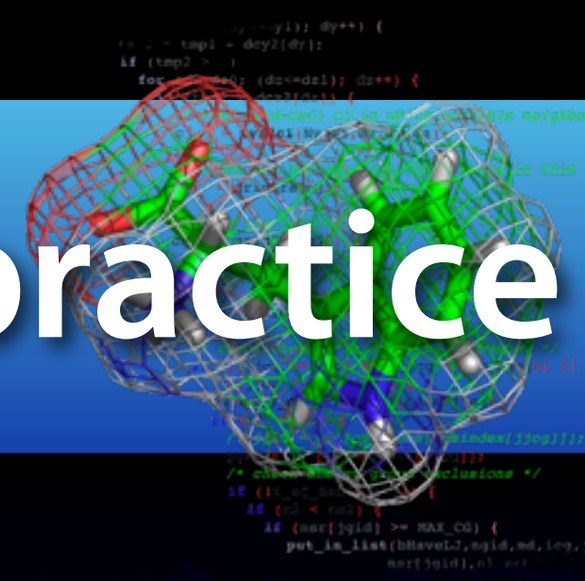


GPU
gathering of charges



- sort the atoms before gather !

Gromacs & OpenMM in practice



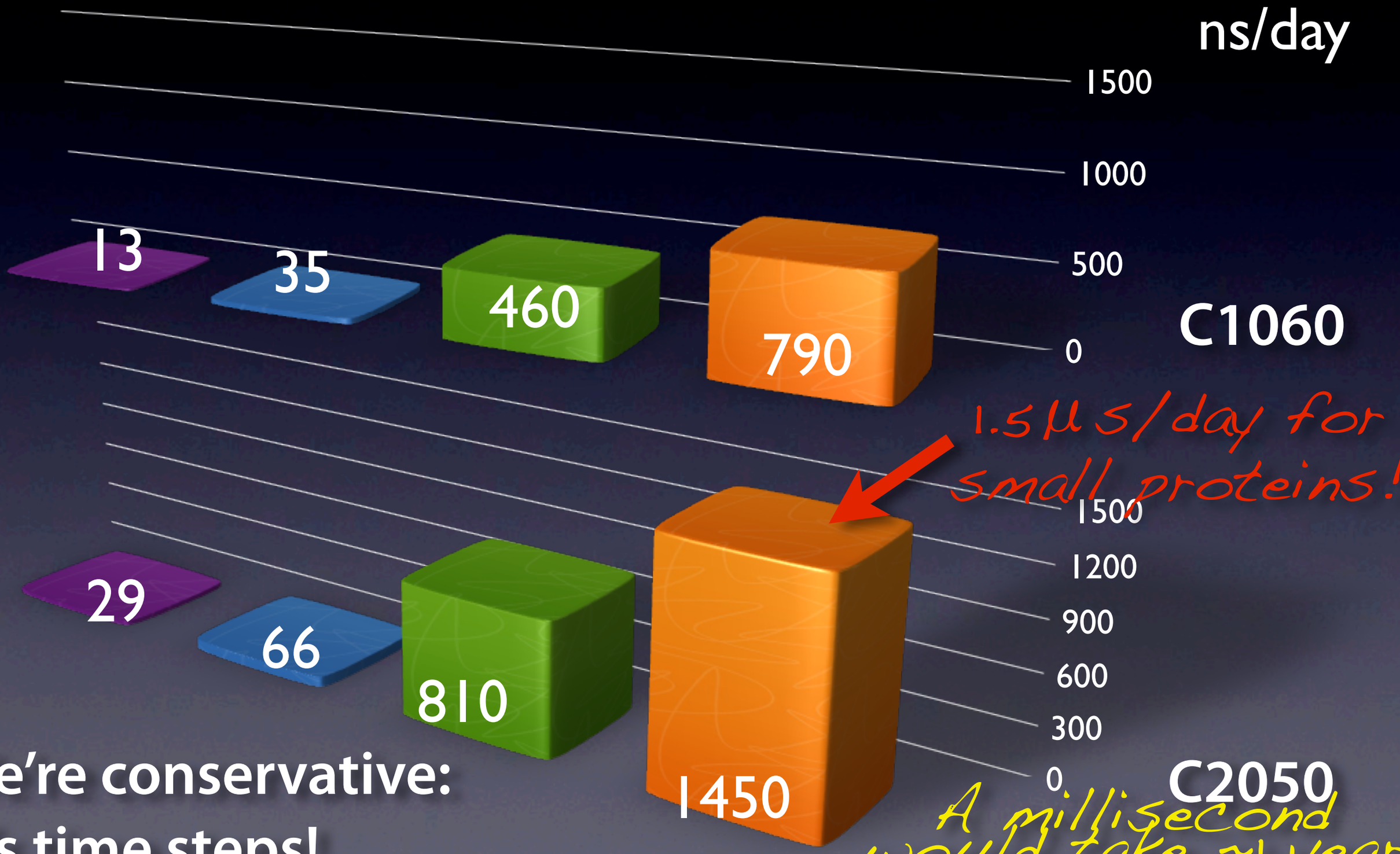
- GPUs supported in Gromacs 4.5
  mdrun ... -device "OpenMM:Cuda"
- Same input files, same output files: "It just works"
- Subset of features work on GPUs for now (checked)
- No shortcuts taken on the GPU:
  - At least same accuracy as on the CPU (<1e-6)
  - Potential energies calculated, free energy works
- Prerelease availability: NOW! www.gromacs.org/gpu

Fermi (C20) performance over C10

BPTI (~21k atoms)

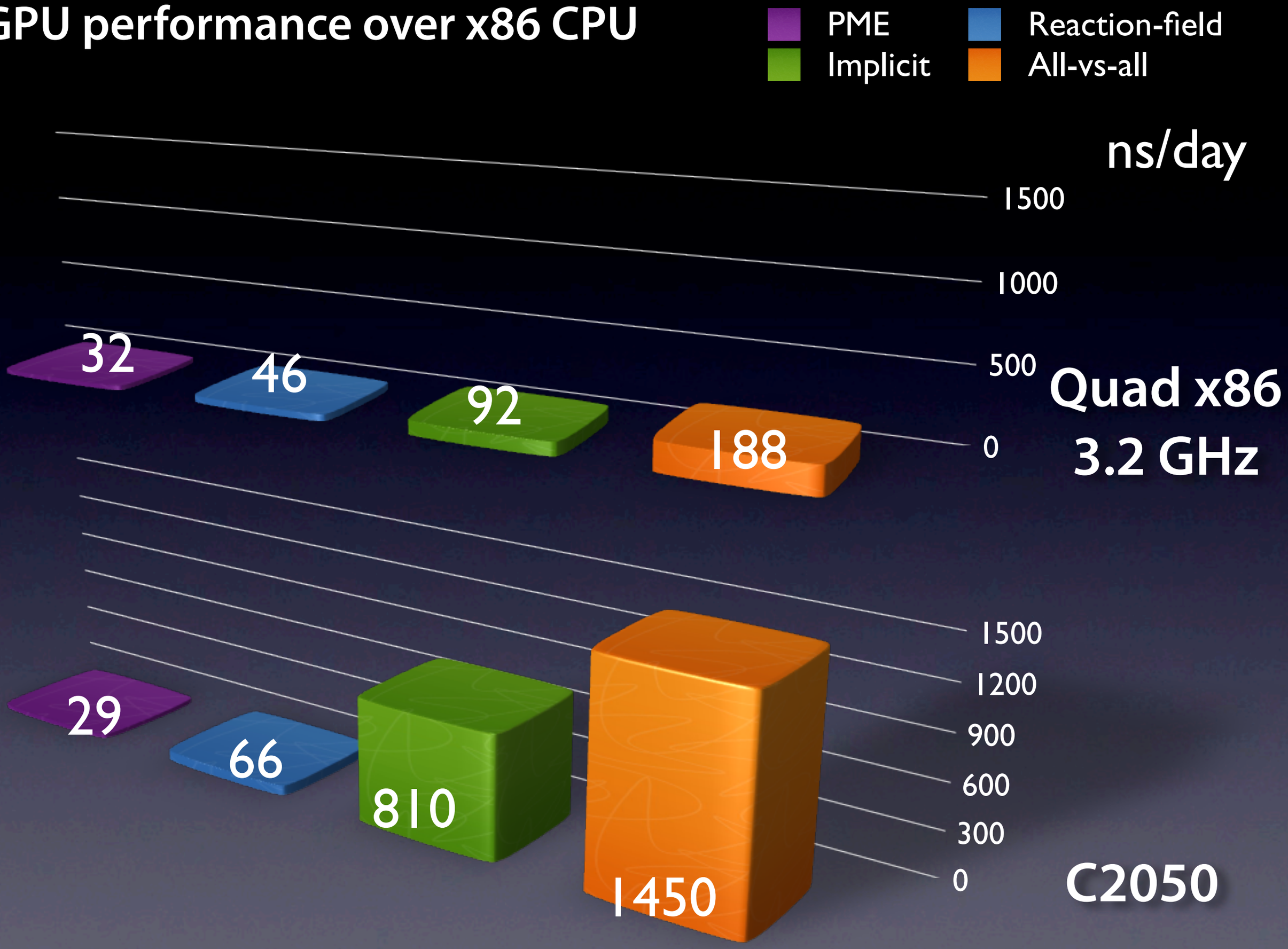
Villin (600 atoms, implicit)

PME
Implicit
Reaction-field
All-vs-all



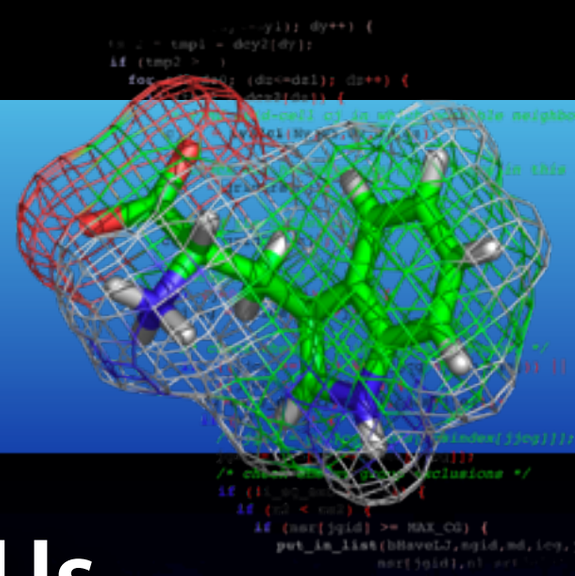
We're conservative: 2fs time steps!

GPU performance over x86 CPU



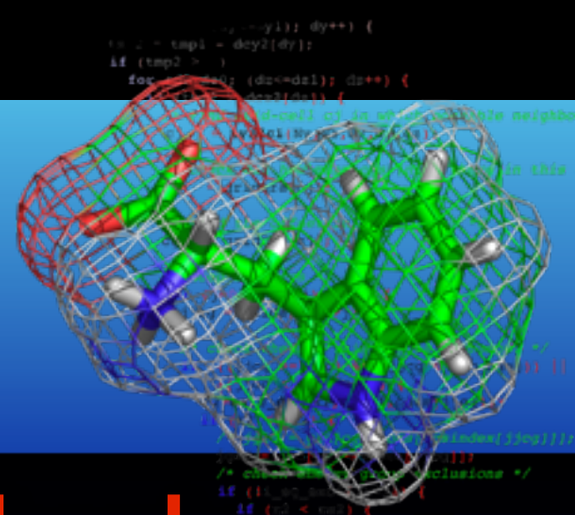
NB: 2fs steps. GPUs still can't do vsites & 5fs like CPUs

Limitations



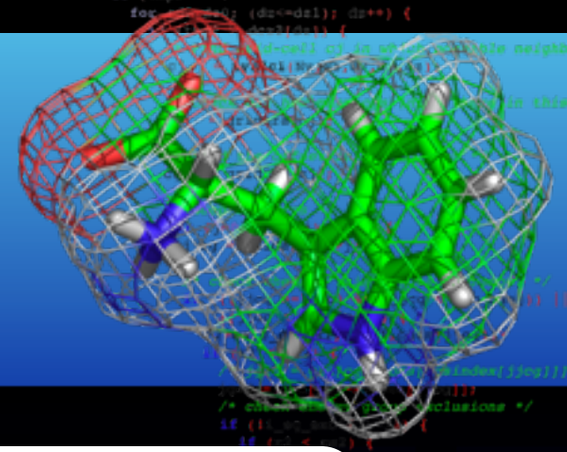
- Still hard to use long time steps on GPUs
  - Virtual sites don't work
  - We don't think you should go higher than 2fs steps without them
- Many of the CPU "tricks" can easily be written in Cuda, but we would end of with lots of kernels that must be called iteratively
- Hard to get multi-node GPU code to beat CPUs
  - In the high end, we're all bandwidth-limited

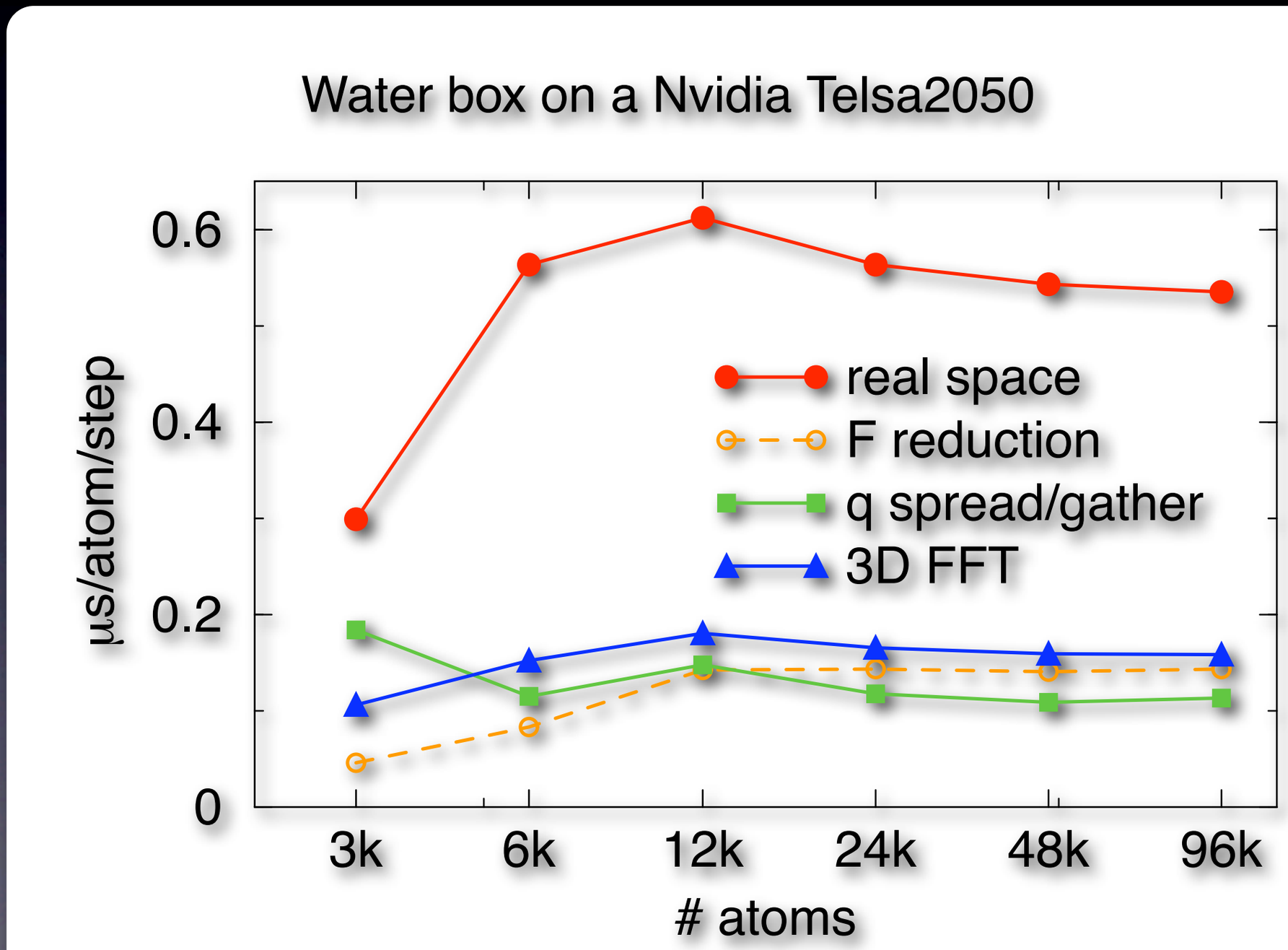
Hardware Caveats



- Beware of Memory Errors: happens on all hardware
- Gromacs runs tests to check for GPU memory errors
  - Low-end consumer cards can sometimes be bad
- Even fine cards can exhibit random errors
- For production scientific work you might want Tesla-class Fermi cards...
  - Why? ECC memory! (C2050/C2070)

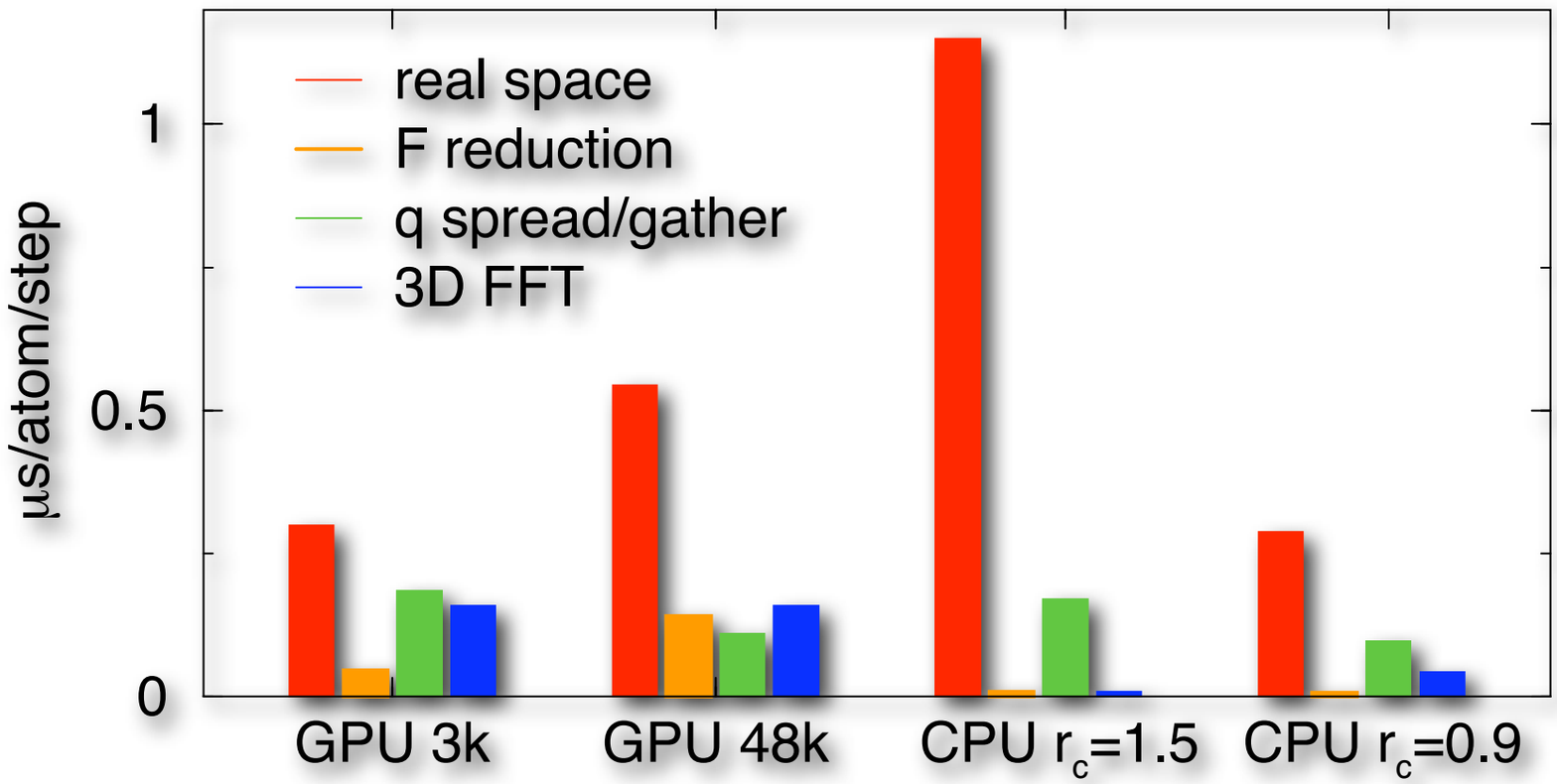
GPU weak scaling





More GPU/CPU comparisons

Nvidia Tesla 2050 vs Intel Core i7 920 (2.66 GHZ, 4 threads)



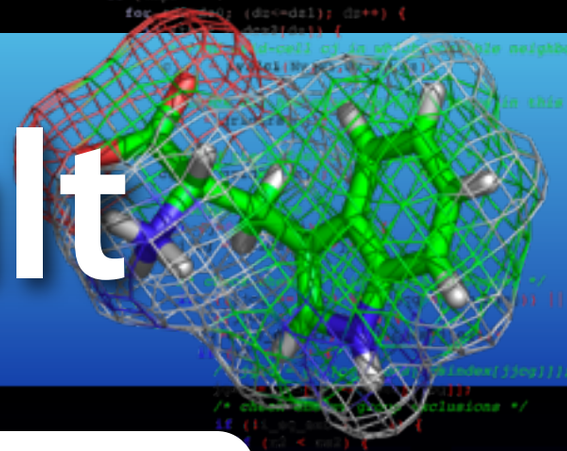
real space cost $\sim r_c^3$

FFT cost $\sim \mathrm{spacing}^{-3}$

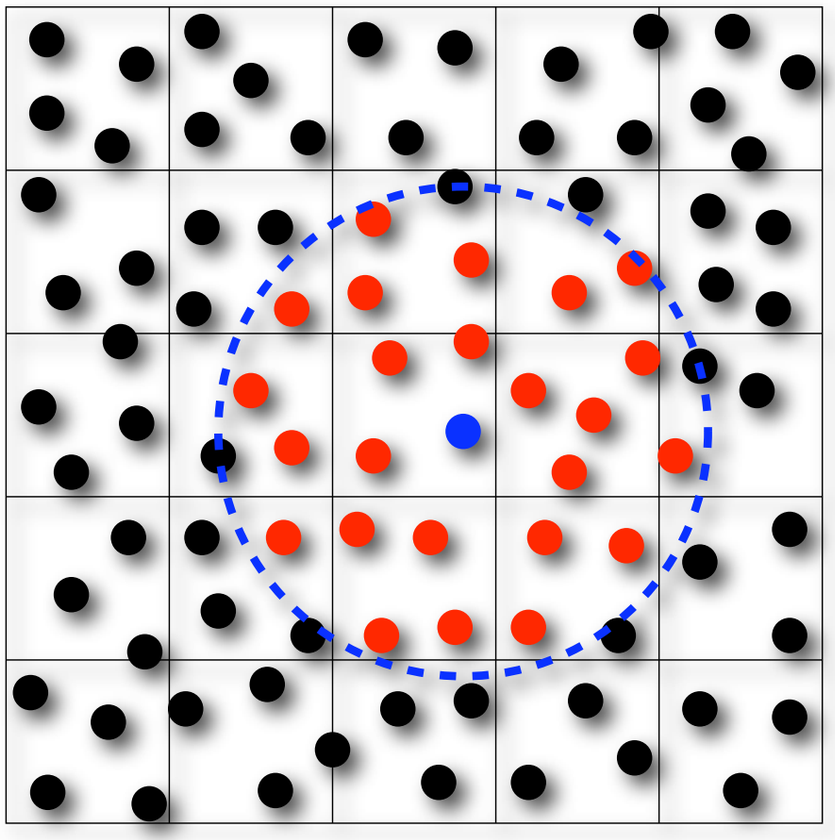
for constant accuracy: $\mathrm{spacing} = r_c/a$

total cost: $C_{\mathrm{pp}}\, r_c^3 \;+\; C_{\mathrm{FFT}}\, a^3\, r_c^{-3}$

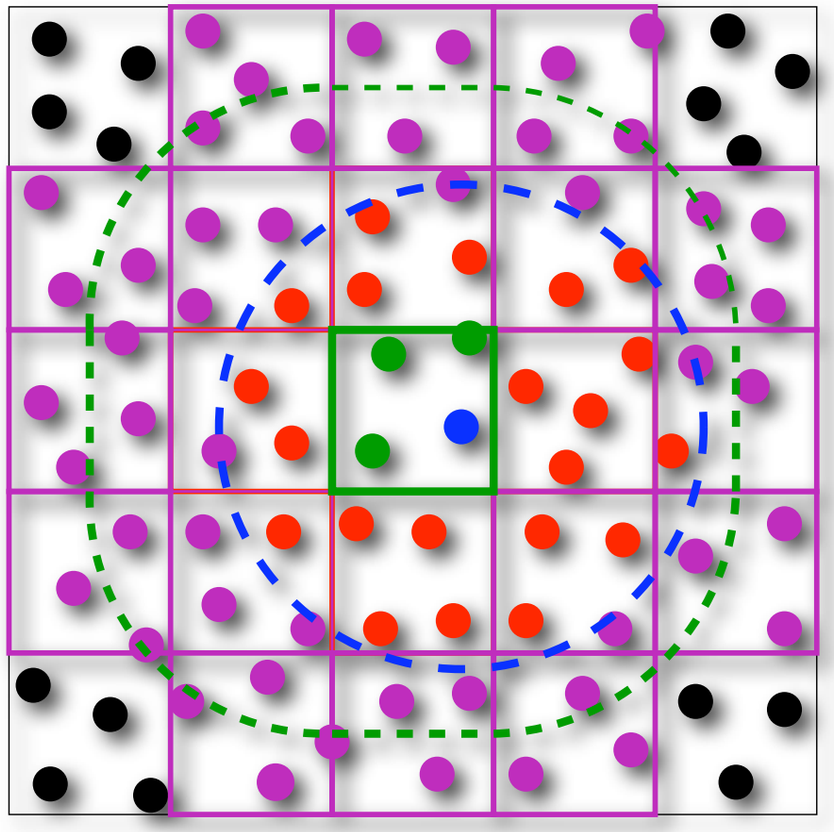
Tiling circles is difficult



serial computing

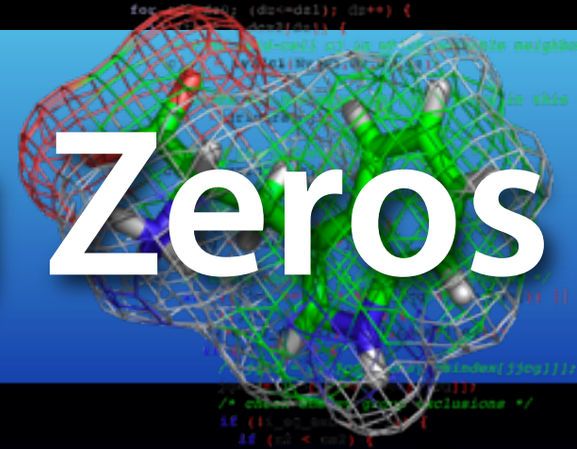


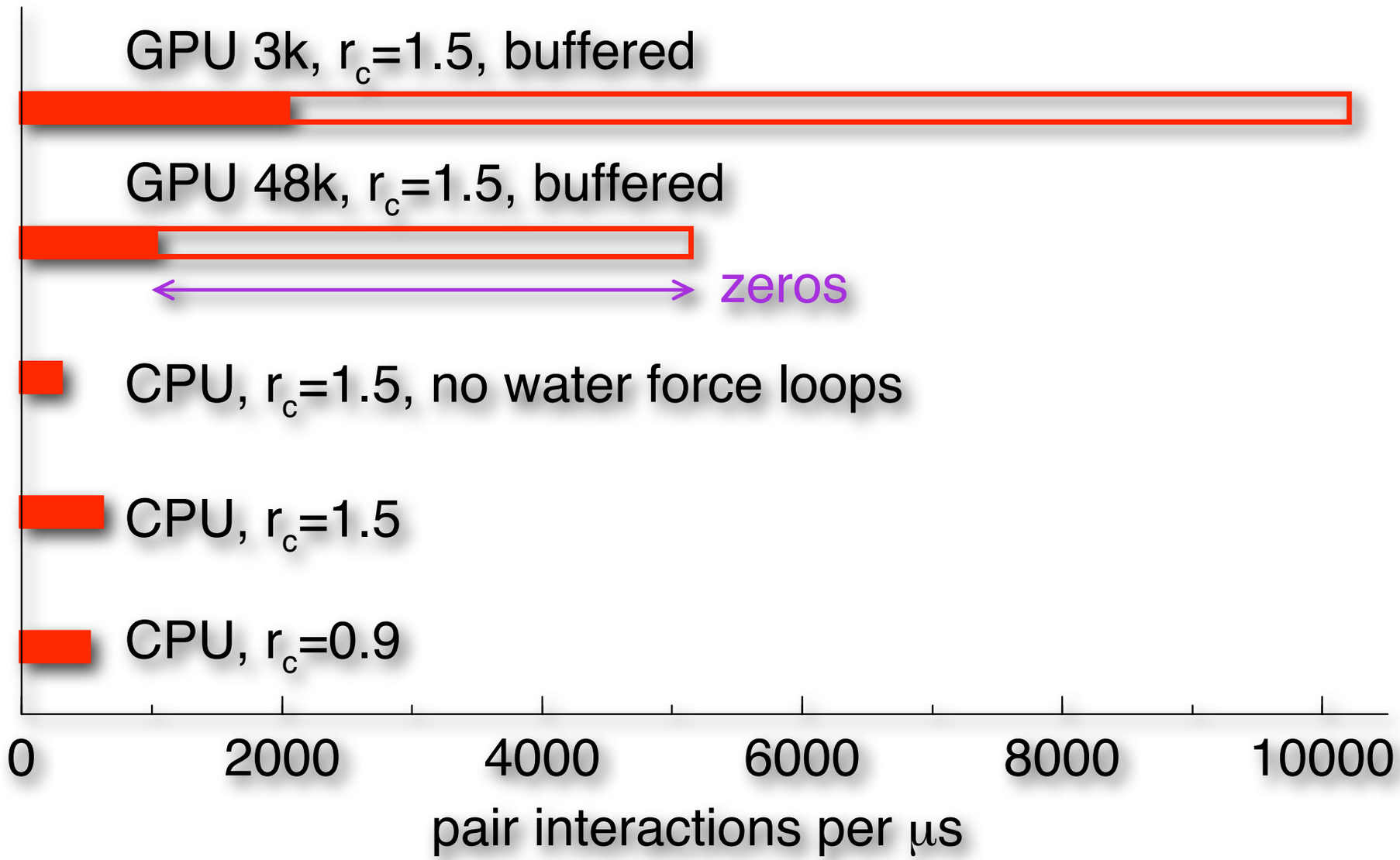
stream computing



1. You need lots of cubes to cover a sphere
2. All interactions beyond cut-off must be zero

The Art of Calculating Zeros





Parallelize the Problem

These will soon be small computers

~2024: 1B 'cores'

2022: ~300M cores

2020: ~100M cores

2018: ~30M cores

2016: ~10M cores

2014: ~3M cores

2012: ~1M cores

2010: ~300,000 cores

How will YOU use a billion cores?

We're all doing Embarrassing Parallelism

But not the way you think.

We're investing huge efforts in parallelizing algorithms that only reach 50-75% scaling efficiency on large problems

Not a chance they will scale to 1B cores

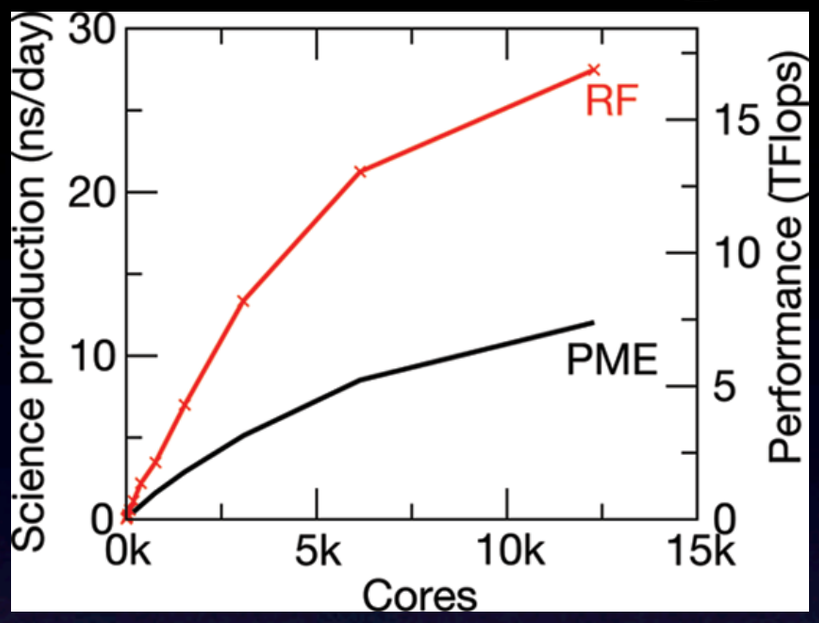
Close-to-useless for smaller problems of commercial interest

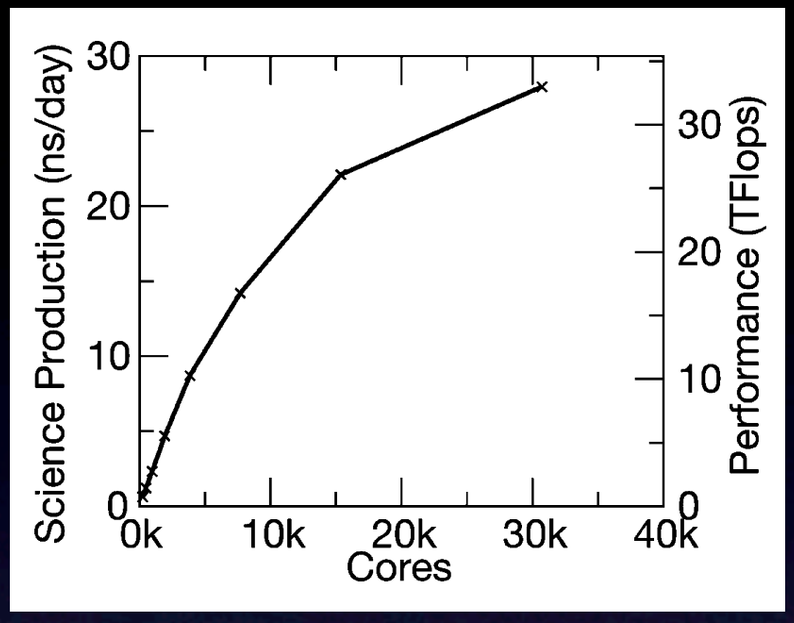
100% focus on programs, forget the problem we're solving

Ask taxpayers to foot the bill

Pretty much the definition of 'embarrassing'?

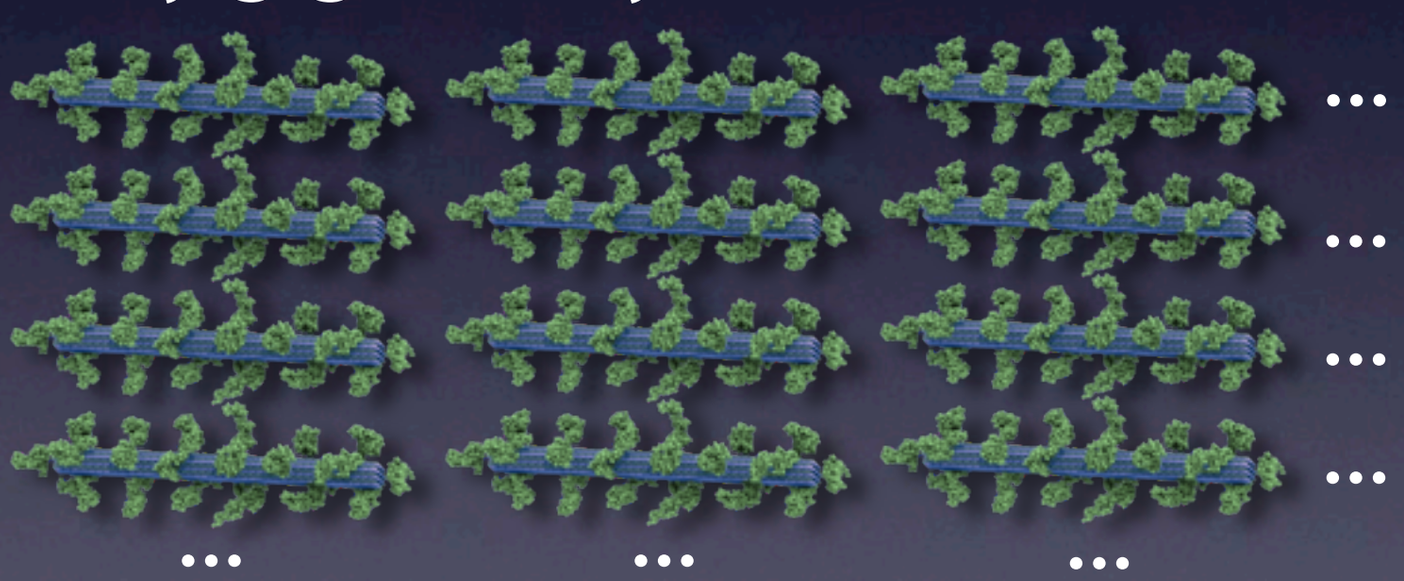
Scaling as an Obsession?



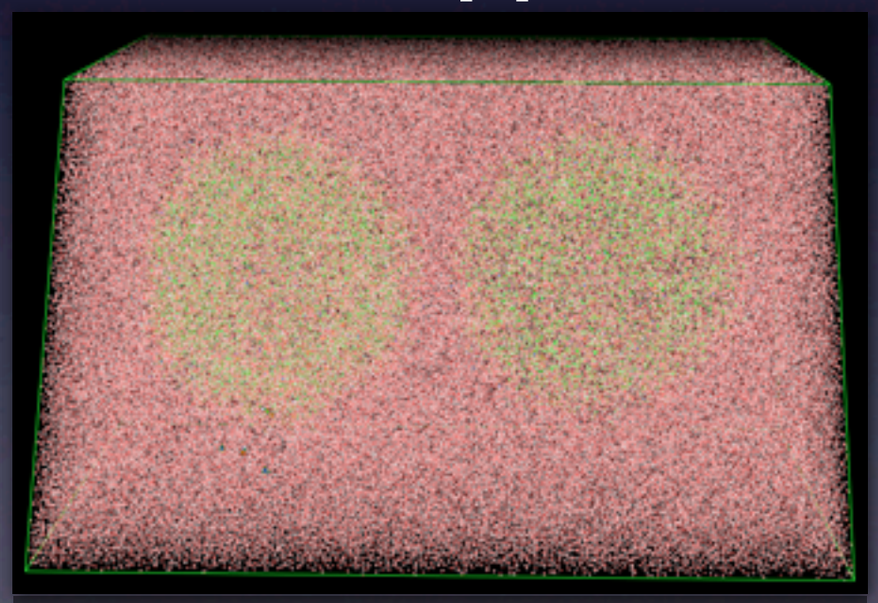


Gromacs has scaled to 150k cores on Jaguar @ ORNL

Only gigantic systems scale - limited number of applications



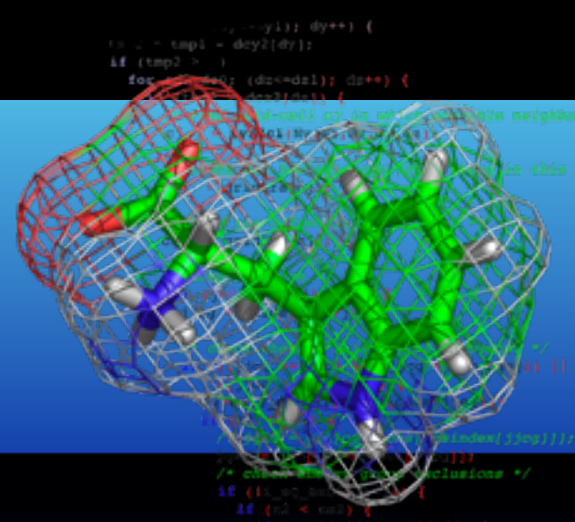
1M-100M atoms

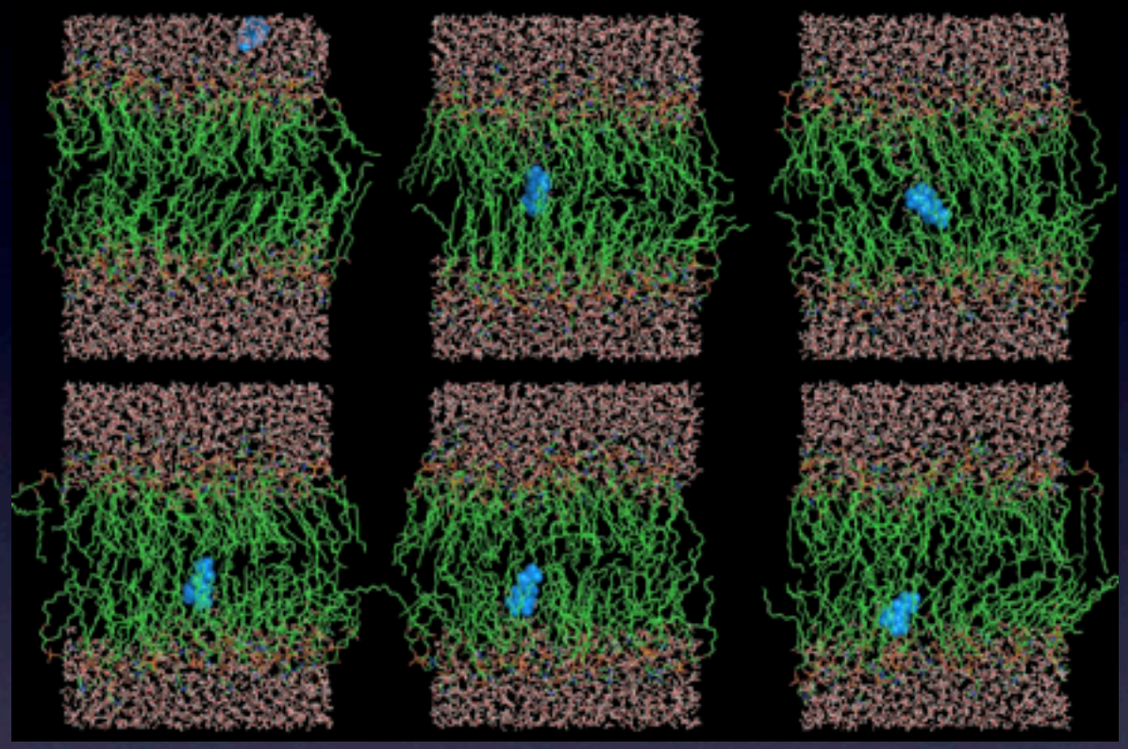


But: <u>Small</u> systems won't scale to large numbers of cores!

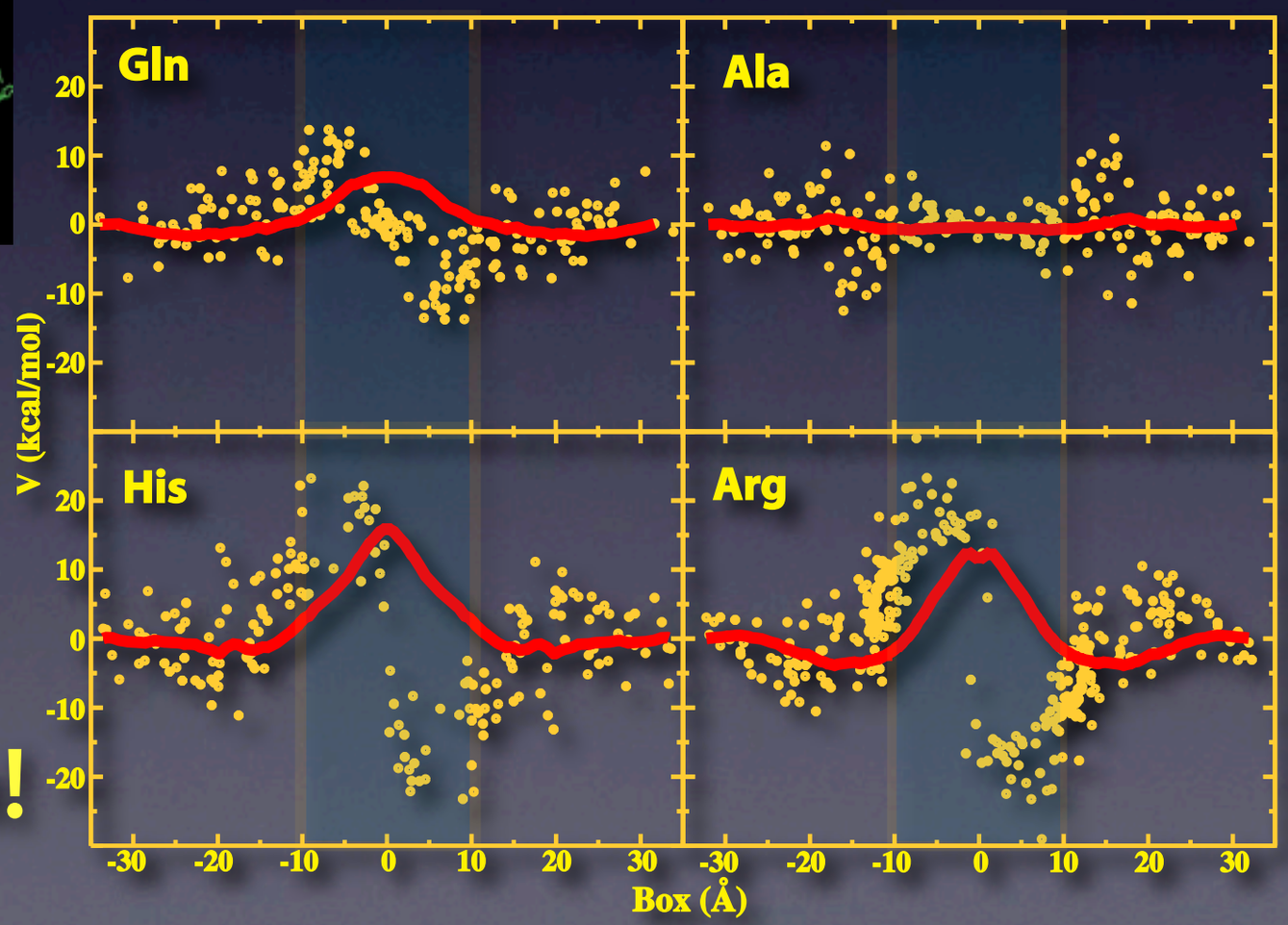
How can we break this impasse?

Ensemble Simulation



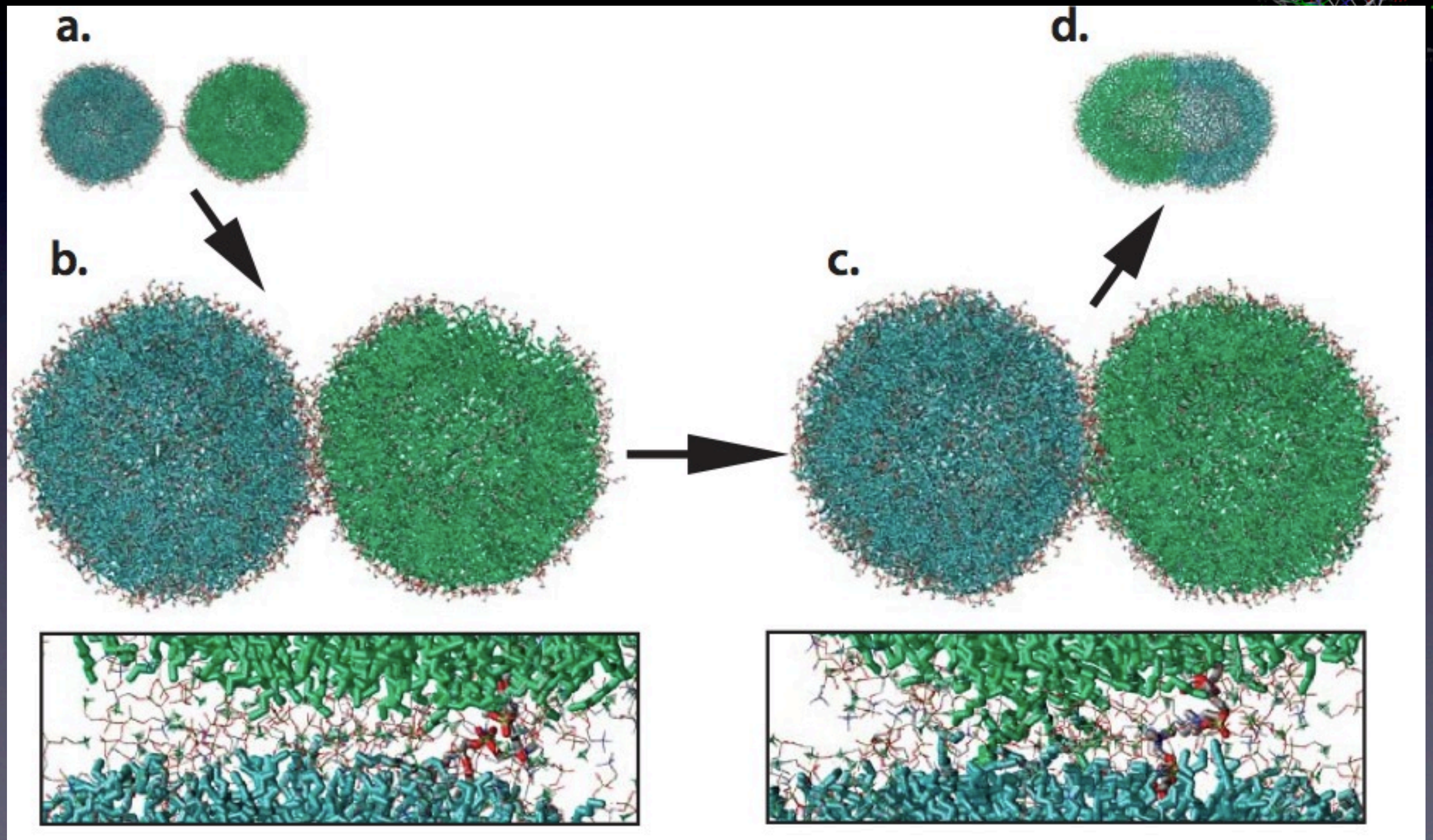


Membrane Protein Insertion Free Energy

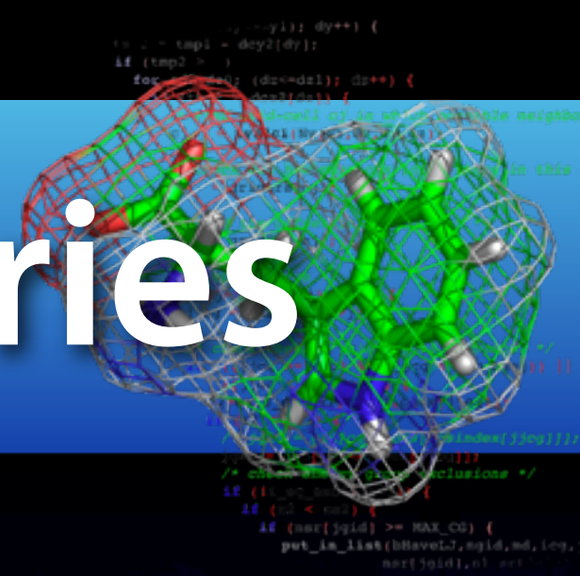


Every dot is a simulation!

Vesicle fusion - 1.5M atoms

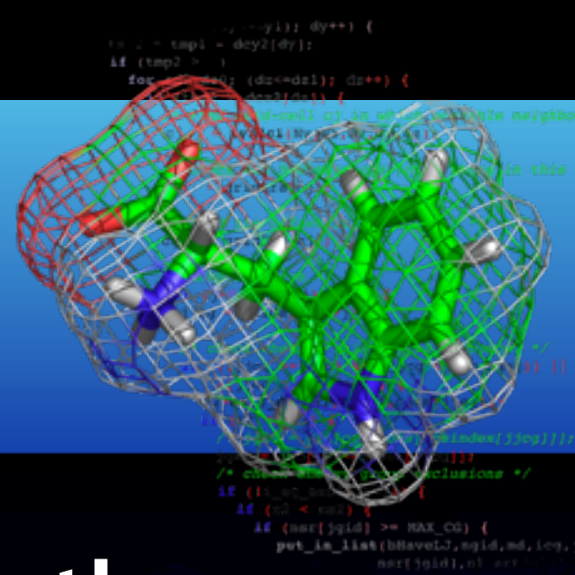


Long fusion trajectories



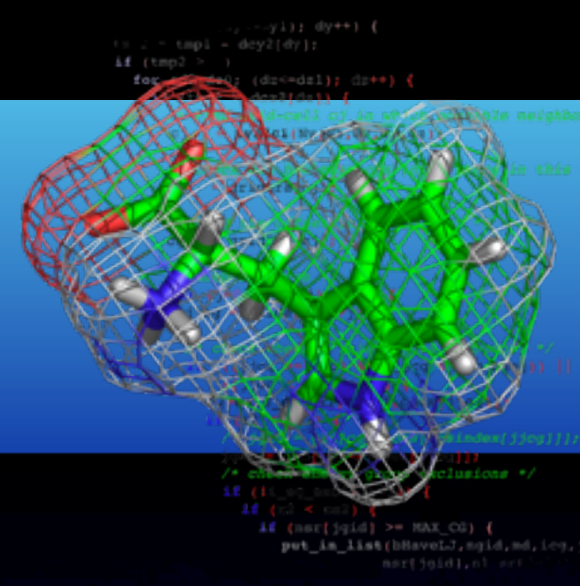
- Run on Infiniband cluster, ~250-500 cores
- 7 vesicle pairs fusing in 100-250ns
  - 75-100% POPE lipids
- 2 non-fusing vesicle pairs, 100ns & 500ns
  - 50% POPE lipids
- Interesting circumstantial observations, but hard to draw conclusions from

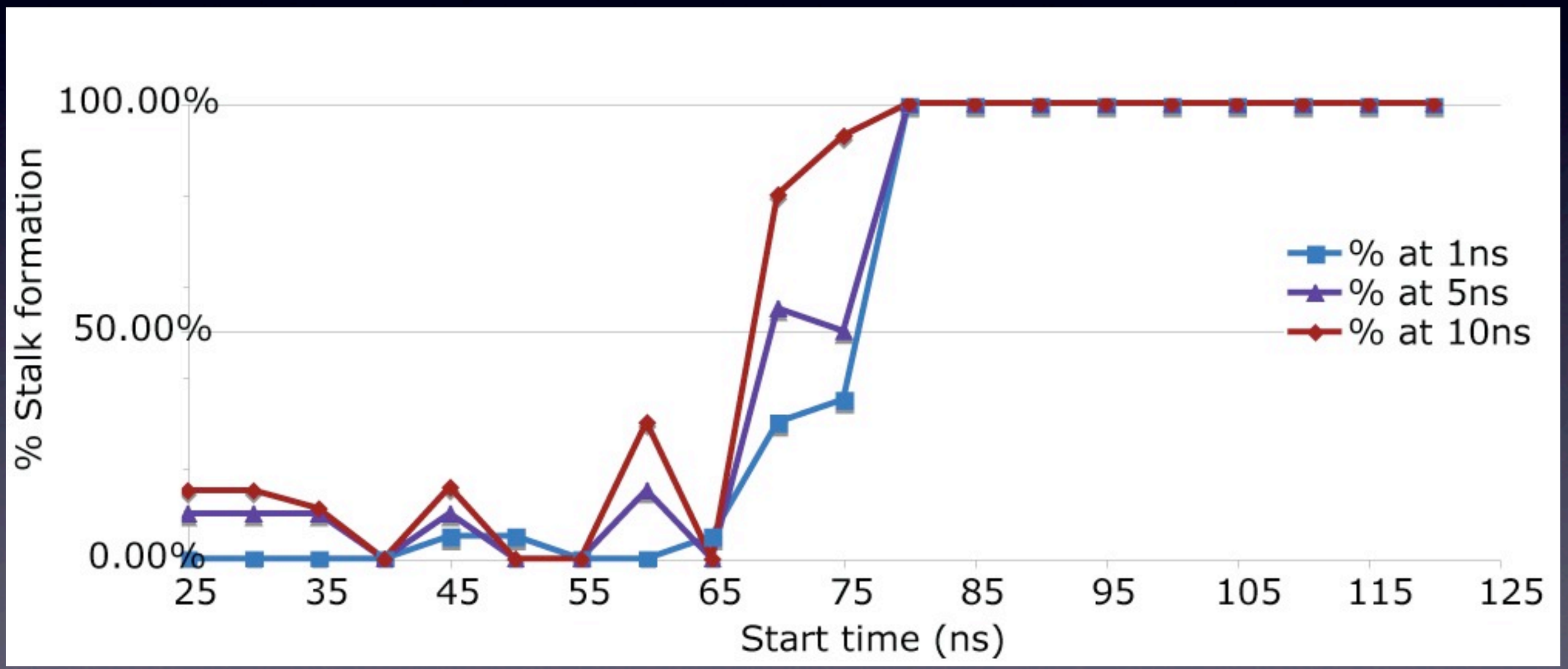
Committer Analysis



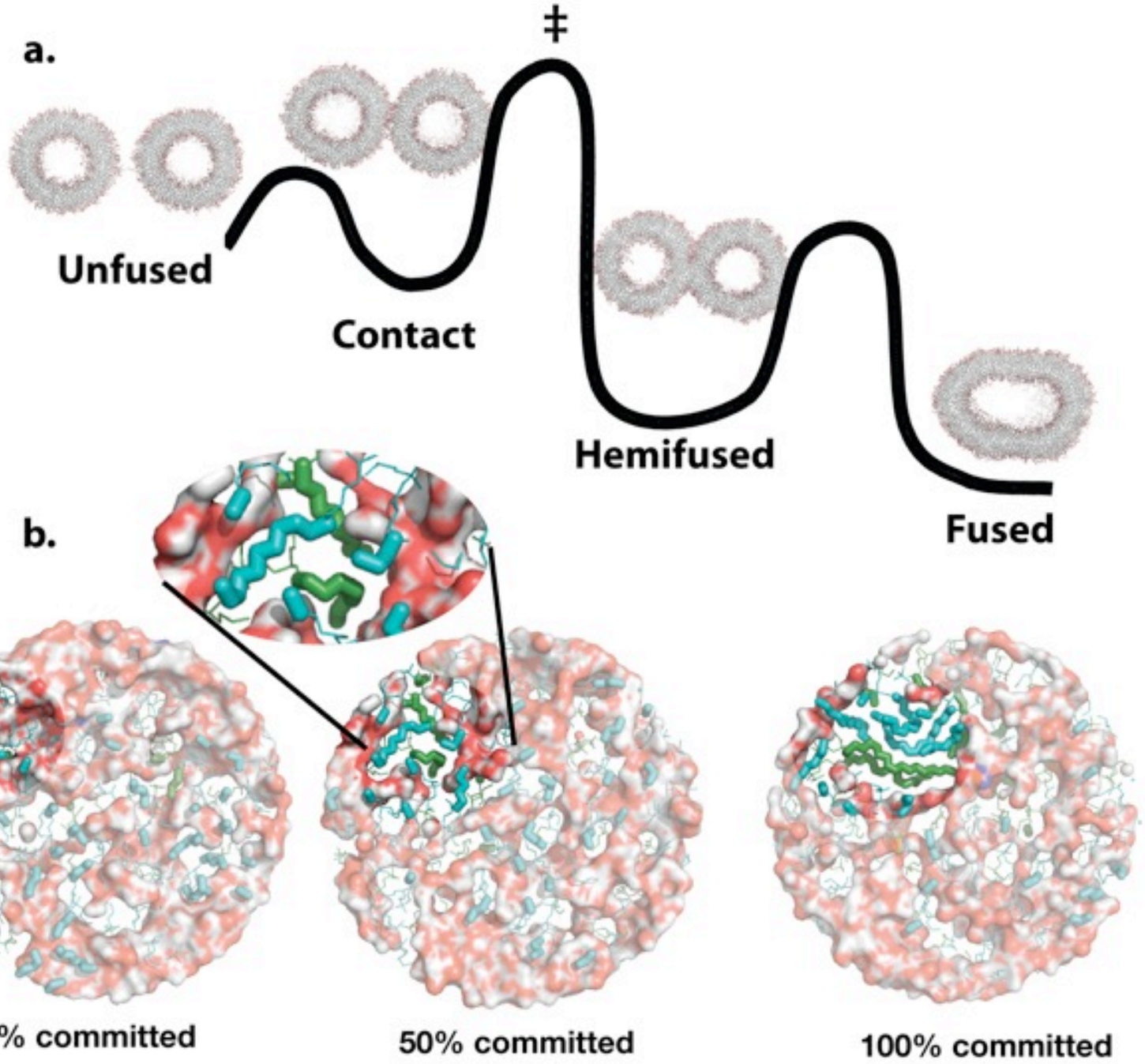
- Pick 20 conformations along fusion path
- Restart with 20 different random seeds
- ‘Shooting trajectories’
- 8 microseconds of additional simulation
- Run on capacity cluster, 16-32 cores each
- Folding@Home as cluster scheduler
- Calculate fusion probabilities

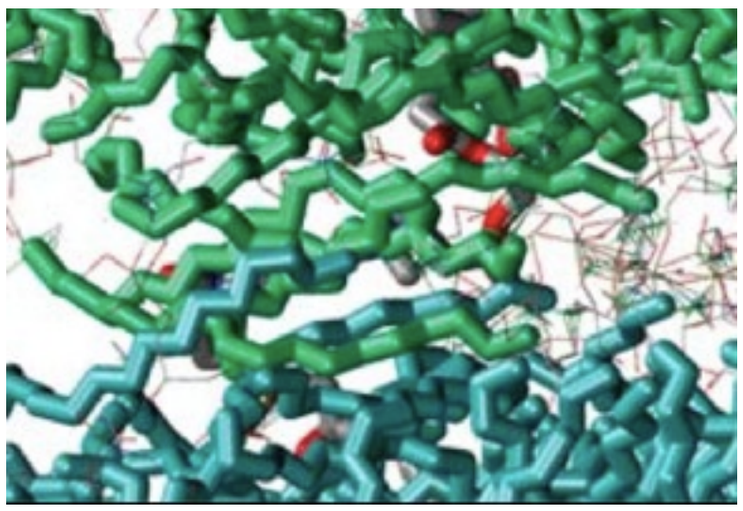
Stalk commitment



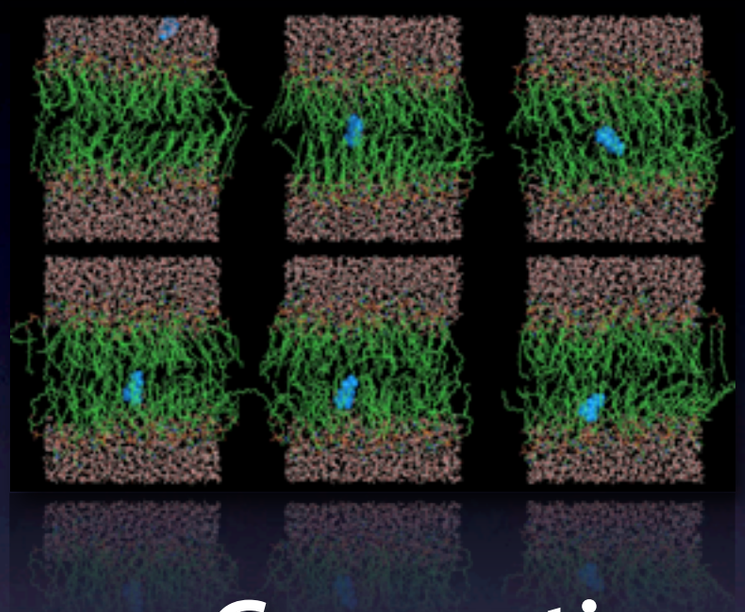


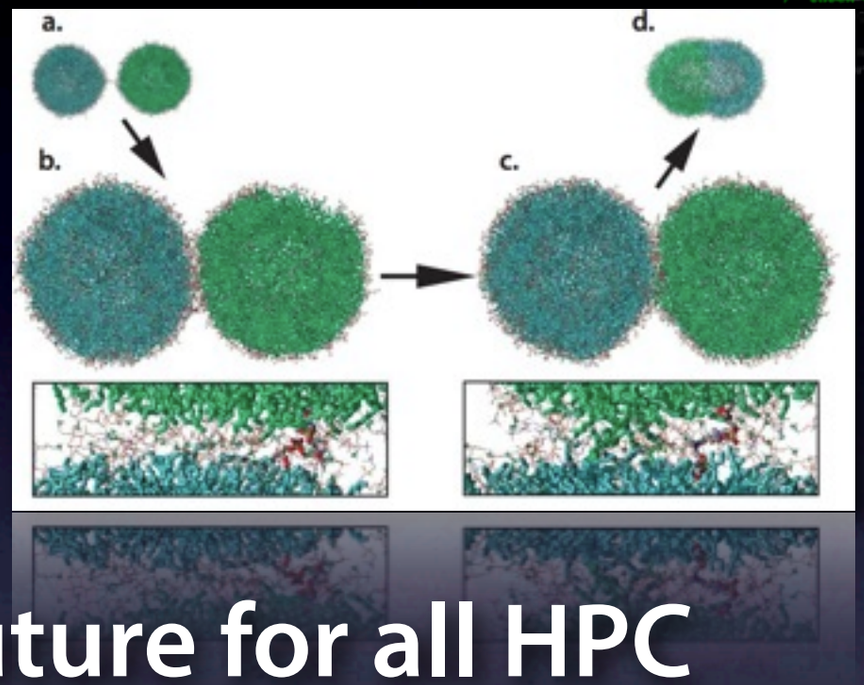
Statistics from 1000's of runs





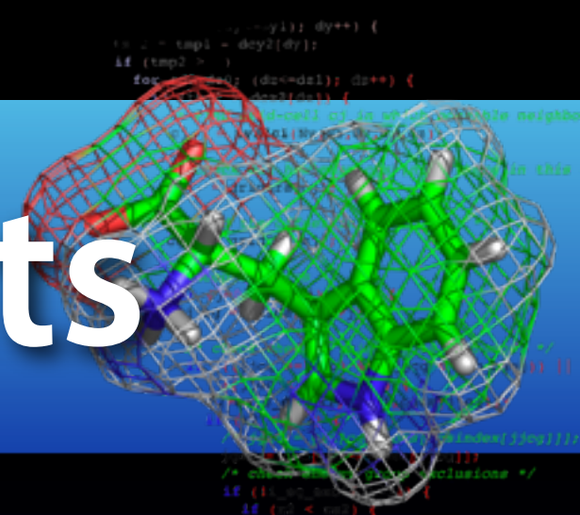
Scale the Problem, not Runs





- Stream Computing is the future for all HPC
- We're doing *statistical* mechanics!
- No algorithm will parallelize 5000 degrees of freedom over 1 billion processors
- Parallelize in the problem domain instead
- Node efficiency becomes the key measure

Acknowledgments



- **GROMACS:** Berk Hess, David van der Spoel, Per Larsson
- **OpenMM:** Rossen Apostolov, Szilard Pall, Peter Eastman, Vijay Pande
- **Nvidia:** Scott LeGrand, Duncan Poole, Andrew Walsh
- **Ensemble Simulations:** Peter Kasson