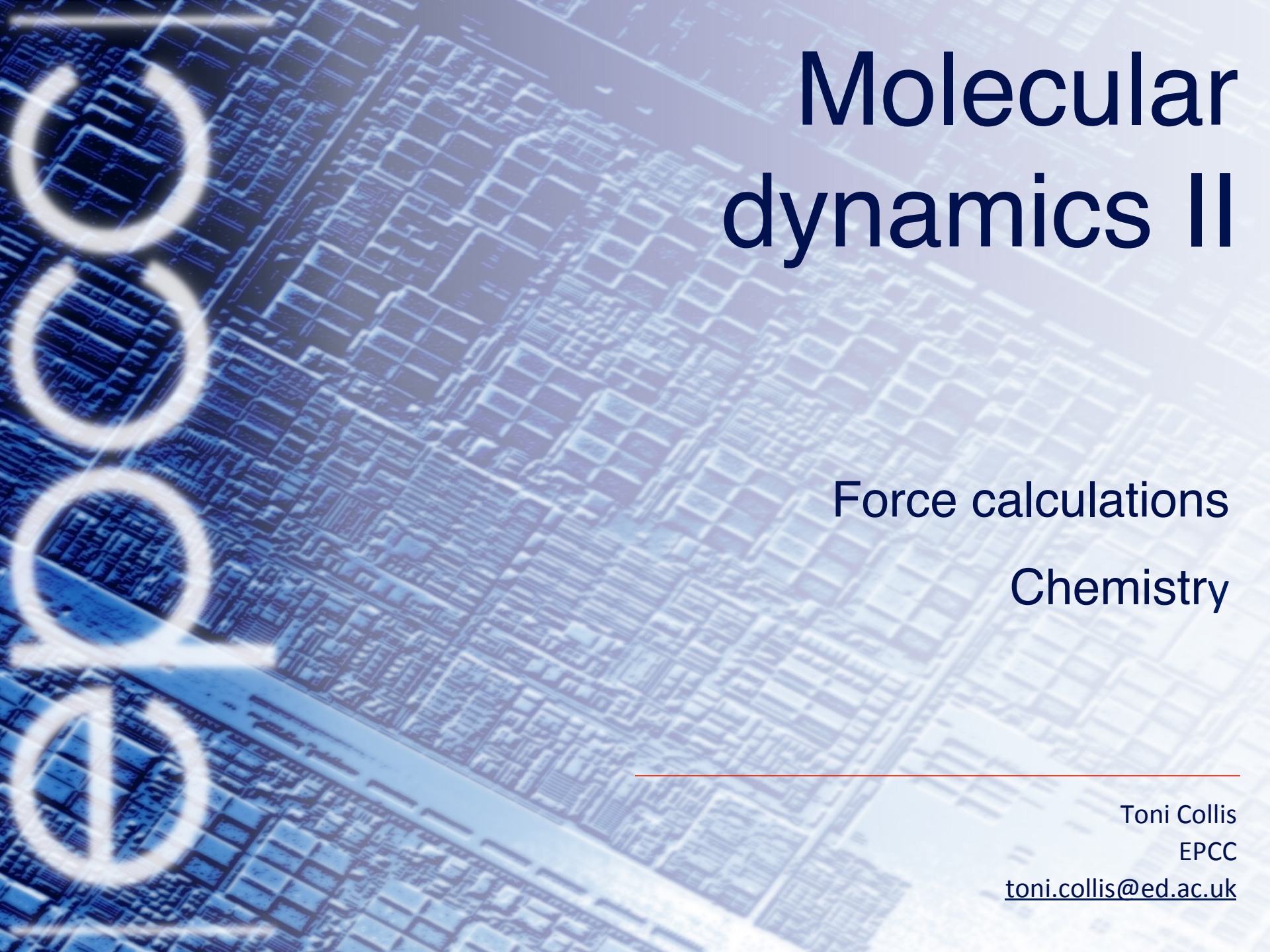


Molecular dynamics II

Force calculations

Chemistry

Toni Collis
EPCC
toni.collis@ed.ac.uk

Outline

- Practical
  - Recap
- Molecular Dynamics
  - Computational Chemistry and MD
  - Why you would want to do MD simulations
  - Using N-body methods for an MD code
- Parallelisation techniques

Orbits – the practical

- Conservation of energy requires:

$$E_T = E_P(t) + E_K(t) = C$$

- Understanding the error
  - Euler and Leapfrog schemes have different errors

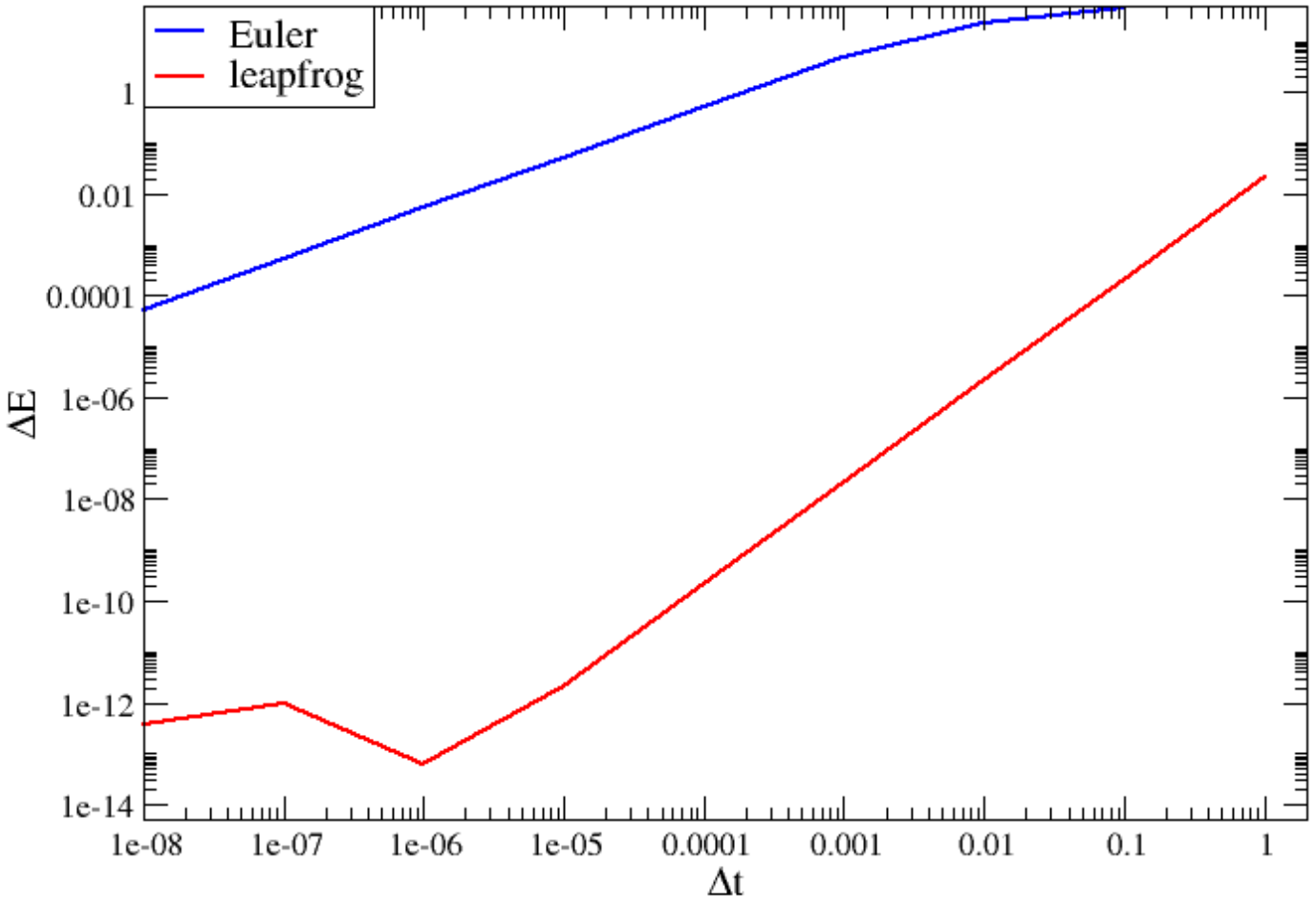
$$\Delta E \propto (\Delta t)^m$$

- Plot ΔE versus Δt on a *log-log* plot

$$\log(\Delta E) \propto \log((\Delta t)^m)$$

$$\log(\Delta E) \propto m \log(\Delta t)$$

Euler versus leapfrog I - Circle

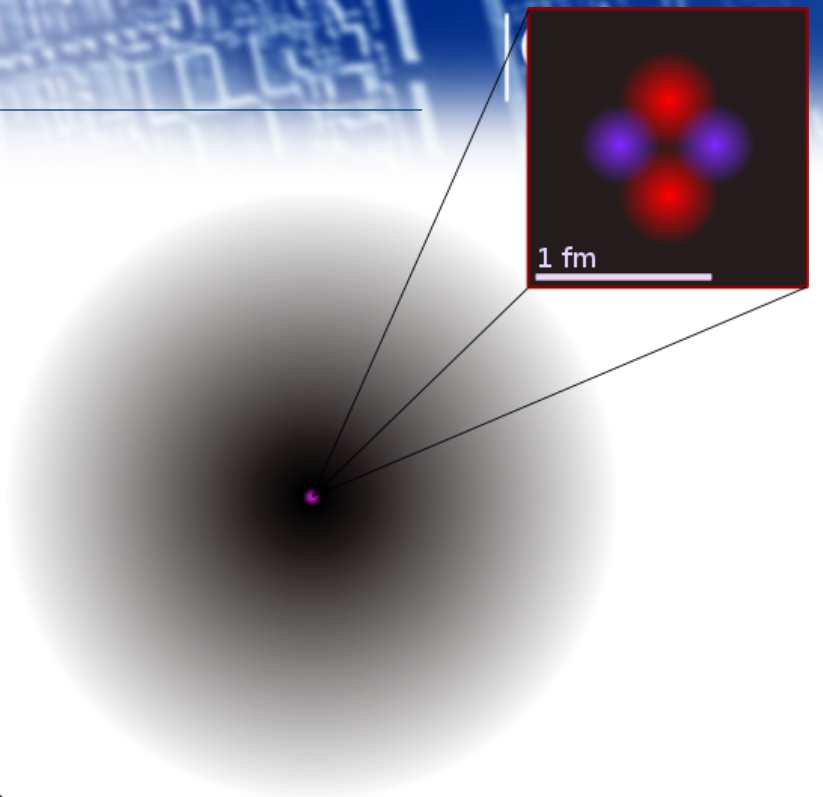


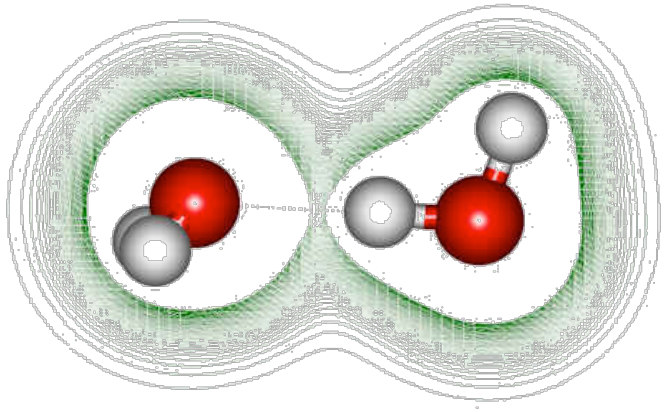
Euler versus leapfrog II

- Even for a simple closed orbit of a two body problem
  - Euler is not sufficient
  - Error proportional to step size
- Leapfrog much better
  - Error proportional to step size squared
- For more complicated systems Higher order schemes required
- Leapfrog good enough to expose rounding errors
  - Complex interplay between discretisation errors and rounding errors

Chemistry

- Chemistry is basically the study of electron interactions.
- The ways in which electrons interact with each other and with atomic nuclei determine the properties of all “stuff”.
- This nanoscopic world is mostly investigated indirectly through experiments on bulk matter.
- We can also *model* chemistry and use computers to gain a deeper understanding between macroscopic observables and their nanoscopic origins.

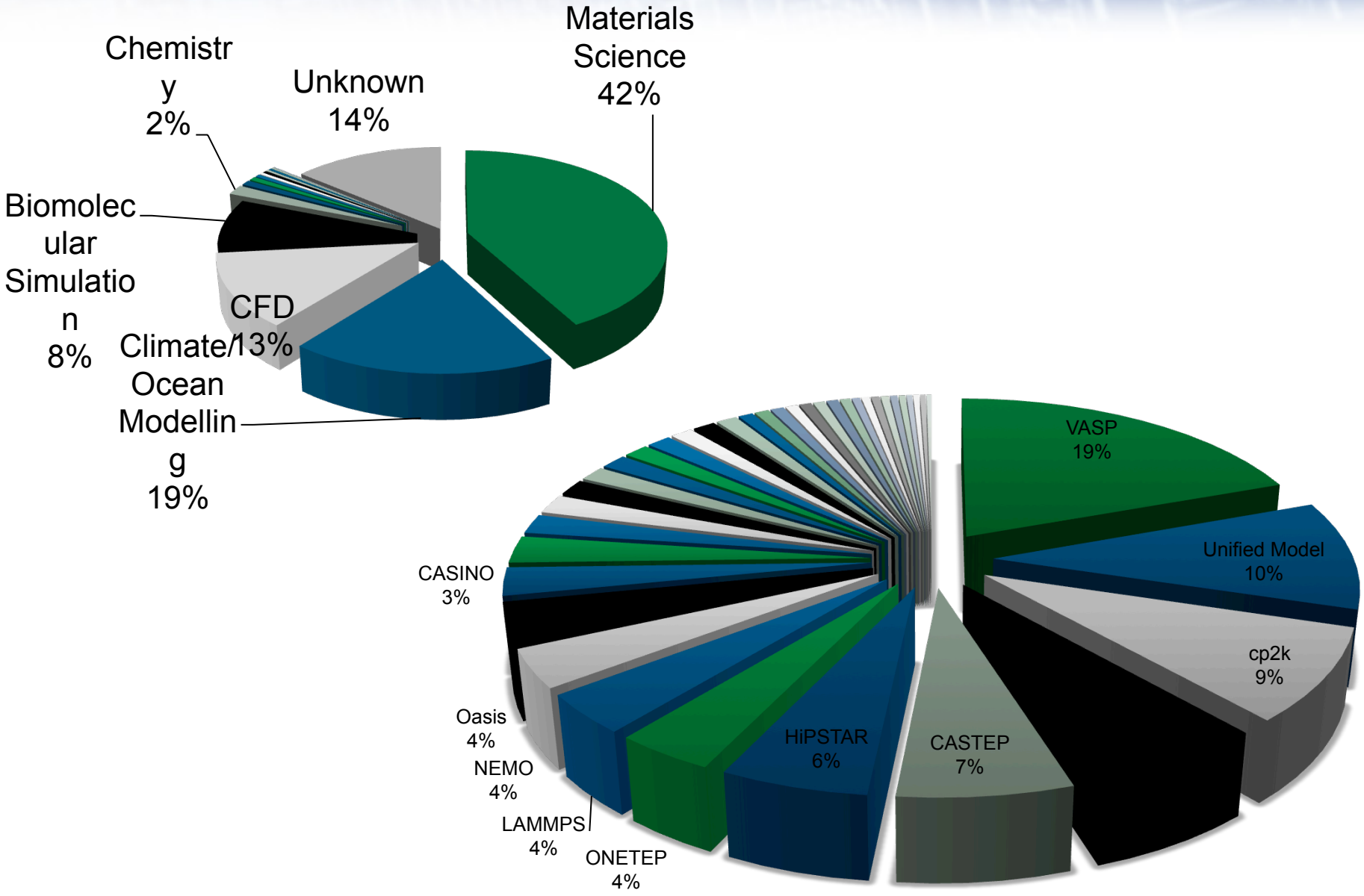




Computational Chemistry

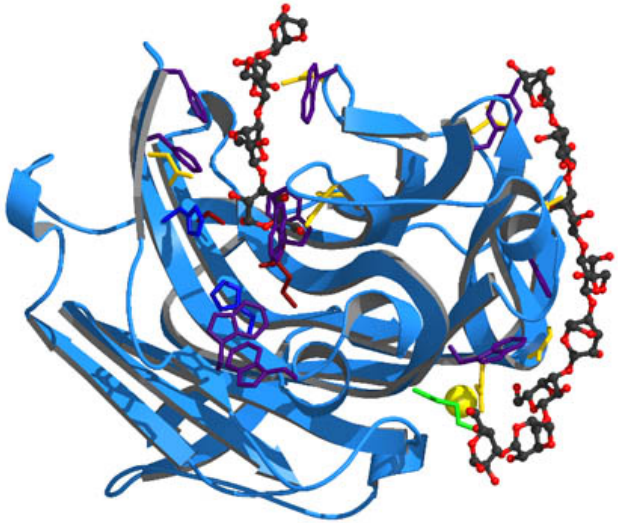
- Computational chemistry models are used by a wide variety of researchers in different fields.
  - Chemists (obviously!)
  - Materials scientists
  - Physicists
  - Biologists
  - Geologists
- Computational chemistry uses around 52% of the active projects on ARCHER (Chemistry, biology, materials science)
- There are a wide variety of computational chemistry approaches from classical simulations to high-level quantum mechanical approaches.

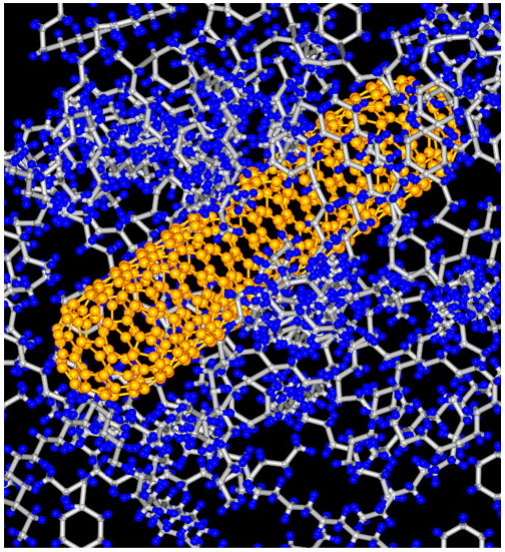
Computational Chemistry



Applications

- Biomolecules
  - Protein dynamics and structure
  - Membranes dynamics and structure
  - Biochemical processes - enzyme catalysis
  - Drug transport and action
- Polymers
  - Structure characterisation in solid and solution
  - Mechanical properties – elasticity, plastic deformation
  - Penetrant diffusion and barrier properties
  - Property modification
- Materials
  - Bulk and surface structure (phase behaviour)
  - Transport properties (conduction - heat, charge, mass)
  - Physical-mechanical properties – strength, elasticity,
  - resilience



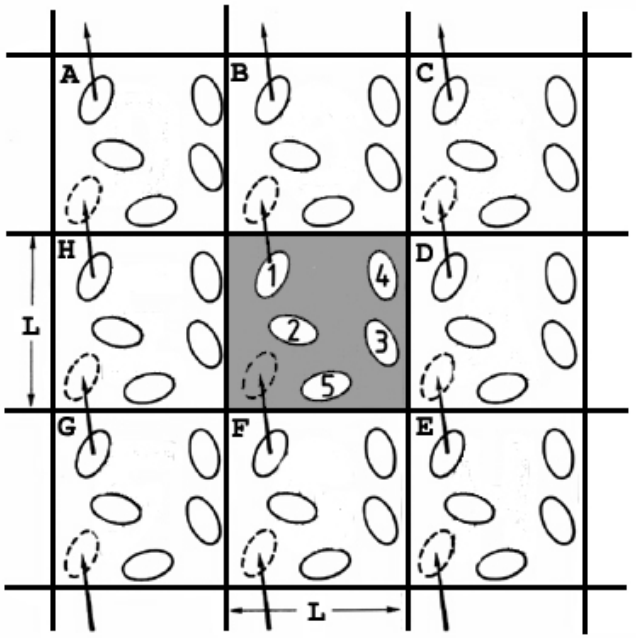


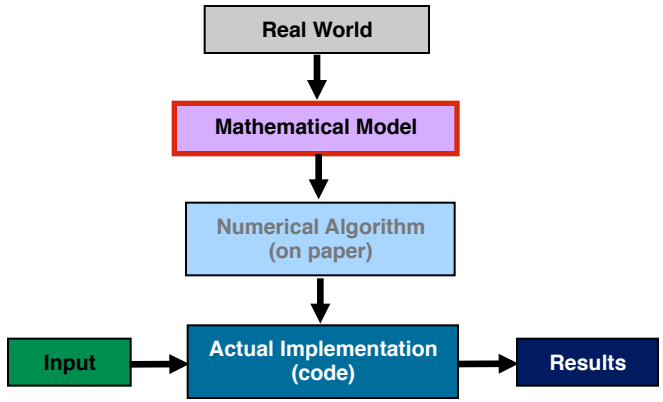
MD Requirements

- Starting structure
  - X-ray crystal structure
  - Previous simulation
  - Amorphous model – this can be harder than the simulation itself!
  - Temperature initialisation
- Integration algorithm
  - Update positions and velocities
- Forcefield
  - Short ranged pair potentials *i.e.* van der Waals terms
  - Long ranged potentials *i.e.* charges, dipoles, polarisation etc
  - Higher order potentials *e.g.* Metals, glasses, semiconductors
  - Intramolecular potentials *e.g.* Bonds, angles, dihedrals etc
- Analysis tools
  - Trajectory

Periodic Description

- Cannot study realistic number of atoms (1 mole = 6.026×10^{23})
  - Employ periodic boundary conditions
  - Works well for crystalline solids – metals *etc.*
  - Also works for amorphous condensed phase though larger cells are needed.

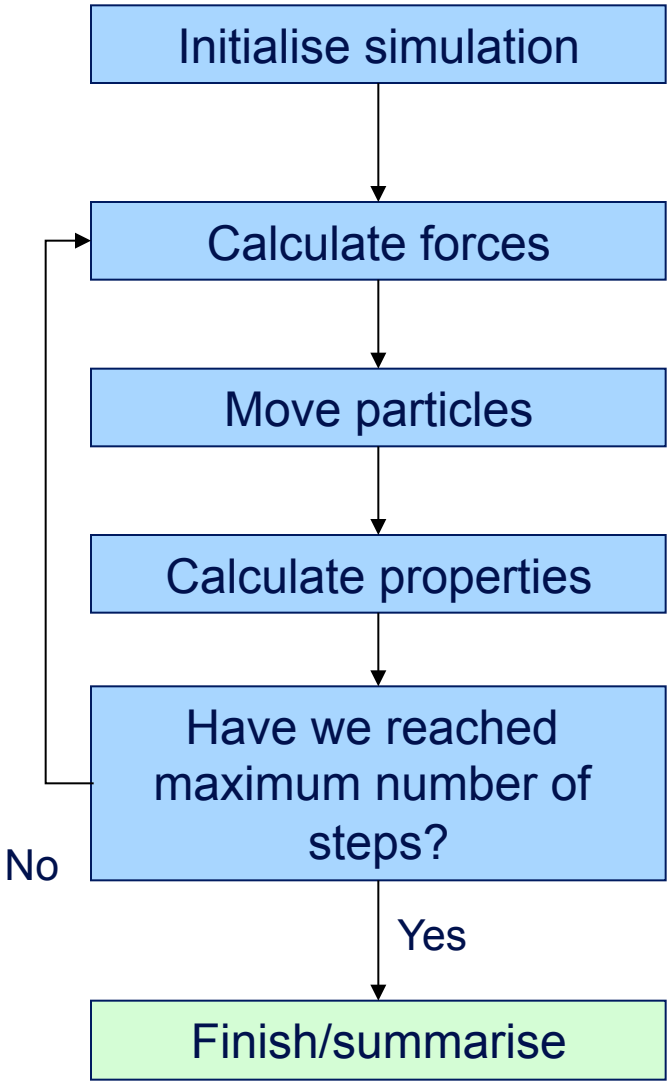




Recap: What is Molecular Dynamics (MD)?

- MD is the solution of the classical equations of motion for atoms and molecules to obtain the time evolution of the system.
- Applied to many-particle systems - a general analytical solution not possible. Must resort to numerical methods and computers
- Classical mechanics only - fully fledged many-particle time dependent quantum method not yet available

Overview of MD simulation



Key stages in MD simulation:

- Set up initial system
- Calculate atomic forces
- Calculate atomic motion
- Calculate physical properties
- Repeat!
- Produce final summary

Checks & Sums

- Conservation of energy

$$E_K(t) = \sum_i \frac{1}{2} m_i \vec{v}_i(t)$$

$$E_P(t) = \sum_{\langle i,j \rangle} U(|\vec{r}_j(t) - \vec{r}_i(t)|)$$

$$E_T = E_P(t) + E_K(t) = C$$

Constant: Not time dependant

Similarly Conservation of linear momentum

$$\vec{P} = \sum_i m_i \vec{v}_i(t)$$

Linear momentum should also be conserved in each dimension individually, e.g. *x, y, z*

Classical forcefield

- Forces between particles expressed as simple, parametrised mathematical functions.
  - Except for electrostatic forces – more complicated.
- Usually confined to two-body (pairwise) interactions.
- The complete set of mathematical functions and parameters for a system is known as a *forcefield.*
- Parameters usually generated by empirical fitting rather than derived from first principles.
  - Fitting may be to experimental data or more accurate, *ab initio* simulations (including quantum mechanical description).
- Many standard forcefields exist for commonly simulated systems or you can generate your own (can be difficult).

Forcefield examples

- Short-range interactions: *e.g.* Lennard-Jones

$$V(r_{ij}) = 4\varepsilon_{ij}\left(\left[\frac{\sigma_{ij}}{r_{ij}}\right]^{12} - \left[\frac{\sigma_{ij}}{r_{ij}}\right]^{6}\right)$$

- Coulombic: charge-charge interactions (simplistic)

$$V(r_{ij}) = \frac{q_i q_j}{r_{ij}}$$

- Intramolecular: bonds, angles, torsions

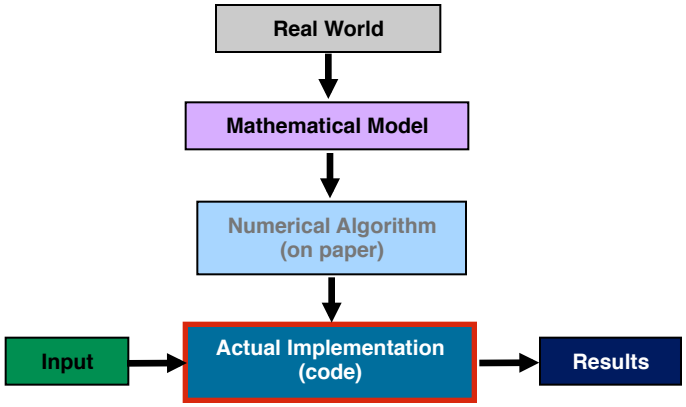
$$V(r_{ij}) = \frac{1}{2}k(r_{ij} - r_{ij,0})^2$$

Common MD Codes

- Amber
  - Long lived, popular biosimulation code for proteins, carbohydrates, nucleic acids. Parallel replicated data. Fortran code.
- DL_POLY
  - General purpose parallel MD code developed in UK. Fortran replicated data and domain decomposition versions.
- GROMACS
  - Biosimulation code spun off from the GROMOS package. Proteins and nucleic acids. An optimised parallel Fortran replicated data code.
- NAMD
  - Biosimulation code for massively parallel machines developed by University of Illinois. C++ domain decomposition code. Dynamically load balanced

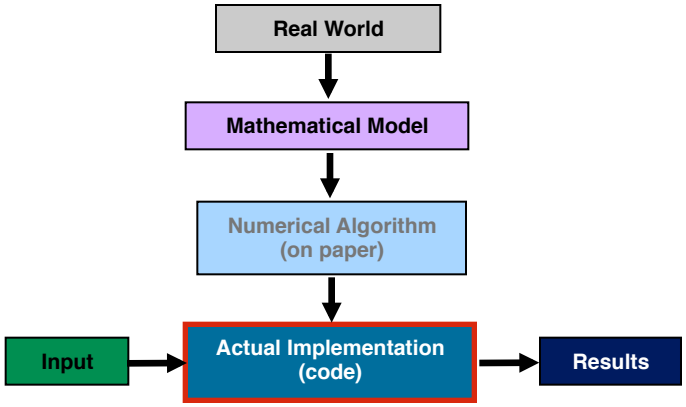
Algorithms

- The most computationally demanding part of the simulation is the force calculation:
  - Loop over all non-equivalent pairs of particles
  - Compute long-range electrostatic forces
  - Can be tricky to parallelise for complicated forcefields
- Long-range force interactions often span more than one simulation box so we need to do something special.
  - Use the Ewald summation – compute electrostatic interaction based on a real and reciprocal space part.
  - Amenable to parallelisation using FFTs

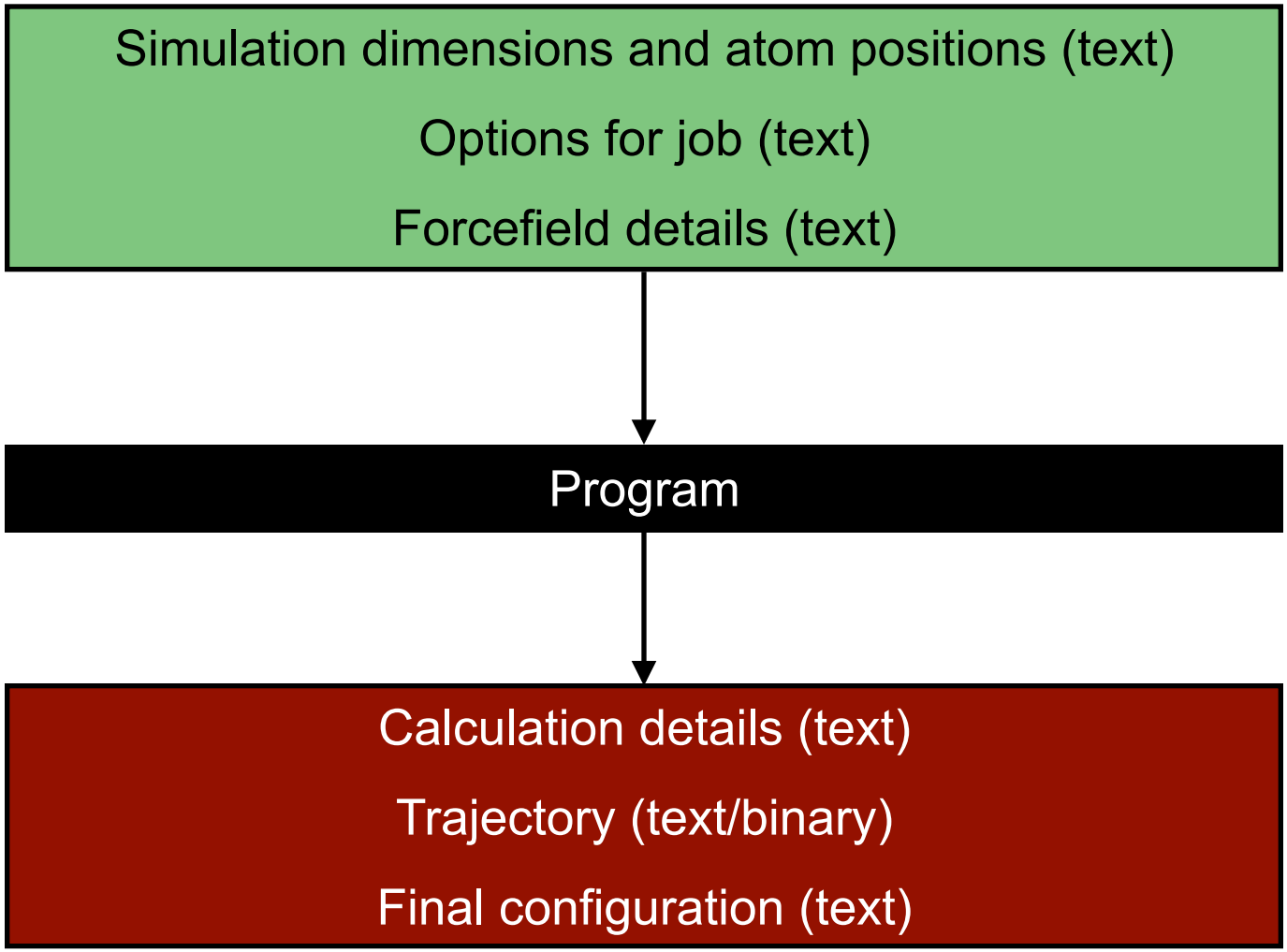


Coding

- Coding of serial version is generally trivial.
  - Just a time integration loop.
  - Some accumulation of properties.
  - Can be written very efficiently.
- Electrostatic and complex (three-body) forces can introduce complications.
  - But still nothing too demanding.
- Complexity arises in parallel implementation.



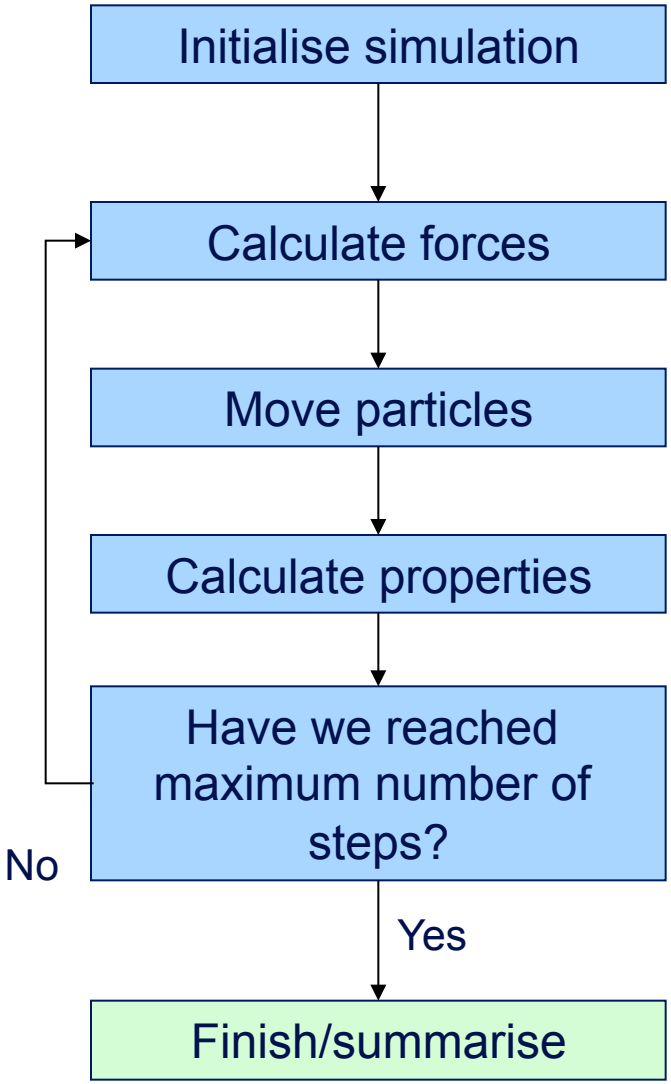
Simple Input and Output



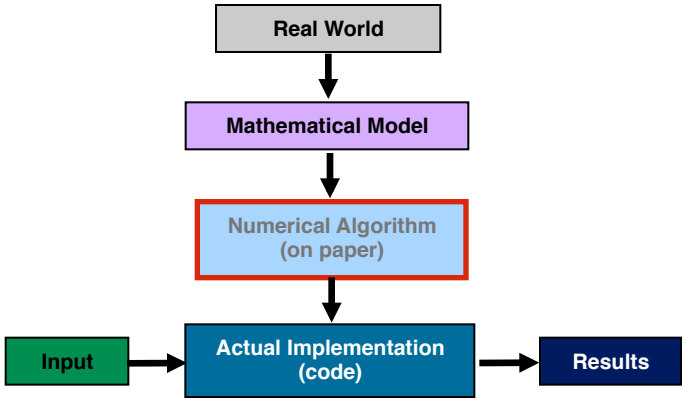
Parallelisation: Parallel issues

- Load Balancing:
  - Sharing work equally between processors
  - Sharing memory requirement equally
  - Maximum concurrent use of each processor
- Communication:
  - Maximum size of messages passed
  - Minimum number of messages passed
  - Local versus global communications
  - Asynchronous communication

Overview of algorithm



- Parallelisation efforts concentrated on force calculation
  - Has implications for other parts of MD scheme
- Parallel I/O has also been implemented in some packages



Parallel Ewald summation

- Ewald summation scheme is a way of handling the long-range electrostatic potential efficiently in a periodic system.
  - Real-space part – standard charge-charge interaction with close ions and *self-interaction*.
  - Reciprocal-space part – interaction with ions images in replica cells. Use Fourier expansion.
- In parallel version
  - Self interaction correction - as is.
  - Real-space terms:
    - Handle as for short ranged forces (*i.e.* domain decomposition, *etc.*)
  - Reciprocal-space terms:
    - Distribute over atoms
    - Distribute over $\boldsymbol{k}$-vectors

Parallelisation

- The problem
  - Each atom is treated as a point mass
  - Easily split space into domains, with atoms located in a domain and assigned to a particular process
  - Every single pair of atoms interacts (even across domains)
  - Atoms move: do they move domains?
  - Needs to be usable on distributed parallel machine
- Solutions?
  - Atom decomposition
  - Force based decomposition
  - Spatial decomposition

Parallelisation: considerations

- Physics is contained in a potential energy function:
  - Can then derive force, position and velocities for each particle
- Typically not memory intensive
  - Only vectors of atom information are stored
- Simulations are ‘large’ in two domains:
  - Number of atoms
  - Number of timesteps (femtosecond scale stepsize, millions of steps)
- Three dimensions: thousands or millions of atoms
- Liquids and solids:
  - Timestep size dictated by vibrational frequencies

MD is ‘inherently parallel’

- Nested loops: perfect for replicated data and shared memory
- Sparse matrices: can disregard long-distance interactions, reducing number of calculations

- BUT: the atoms move!

Atom-decomposition

- Subset of atoms is permanently assigned to a processor
- N/P atoms assigned to each of P processors at beginning of the simulation
- No need to have a spatial relationship
- Each process updates positions and velocities for duration of simulation no matter where they move in physical domain
- Force F(i,j) is sparse as exclude non-local interactions and F(i,j)=-F(i,j)
- Each process is assigned a sub-block of F: N/P rows of the matrix

Atom-decomposition

- For processor P_z with elements F_z many more atom positions than those held on process P_z are needed for the force calculation.
- At each time-step each processor must receive updated atom positions from all other processors (all-to-all communication)
- Requires O(N) storage on every processor
- Best algorithms: $\log_2(P)$ sends and receives and exchanges of (N-N/P) data values

Force based-decomposition methods

- Subset of inter-atomic forces assigned to each process
- Block-decomposition of the force matrix rather than a row-wise decomposition.
- Communication cost is smaller: $O(N/\sqrt{P})$
- F_z sub-block is $(N/\sqrt{P})x(N/\sqrt{P})$
- Neighbour lists are done checking all N^2/P possible pairs in block F_z

Spacial-decomposition methods

- Assign a fixed spatial region to each process
- Each processor computes forces on and update the positions and velocities of all atoms within its box at each timestep
- In order to compute forces on its atoms a proc only needs to know positions of atoms in nearby boxes
- Local communications only
- Two data structures:
  - N/P atoms in local box and for nearby boxes using linked list to allow insertions/deletions of force, pos, vel etc.
  - N/P atoms in local box and for nearby boxes on pos only
- O(N/P) scaling: fastest algorithm. But complex to implement!

CASE study: NAMD

- NAMD exploits MD as a tool to understand the structure and function of biomolecules
  - proteins, DNA, membranes
- NAMD is a production quality MD program
  - Active use by biophysicists (science publications)
  - 50,000+ lines of C++ code
  - 1000+ registered users
  - Features and “accessories” such as
    - VMD: visualization and analysis
    - BioCoRE: collaboratory
    - Steered and Interactive Molecular Dynamics
- Load balancing ref:
  - L.V. Kale, M. Bhandarkar and R. Brunner, *Lecture Notes in Computer Science* 1998, *1457*, 251-261.

NAMD: Initial static loading

- Essentially domain decomposition
- Allocate patches (link cells) to processors so that
  - Each processor has same number of atoms (approx.)
  - Neighbouring patches share same processor if possible
- Weighing the workload on each processor
  - Calculate forces internal to each patch (weight ~ $n_p^2/2$)
  - Calculate forces between patches (i.e. one *compute object*) on the same processor (weight ~ $w*n_1*n_2$). Factor w depends on connection (face-face > edge-edge > corner-corner)
  - If two patches on different processors – send *proxy* patch to lesser loaded processor.
- Dynamic load balancing used during simulation run.

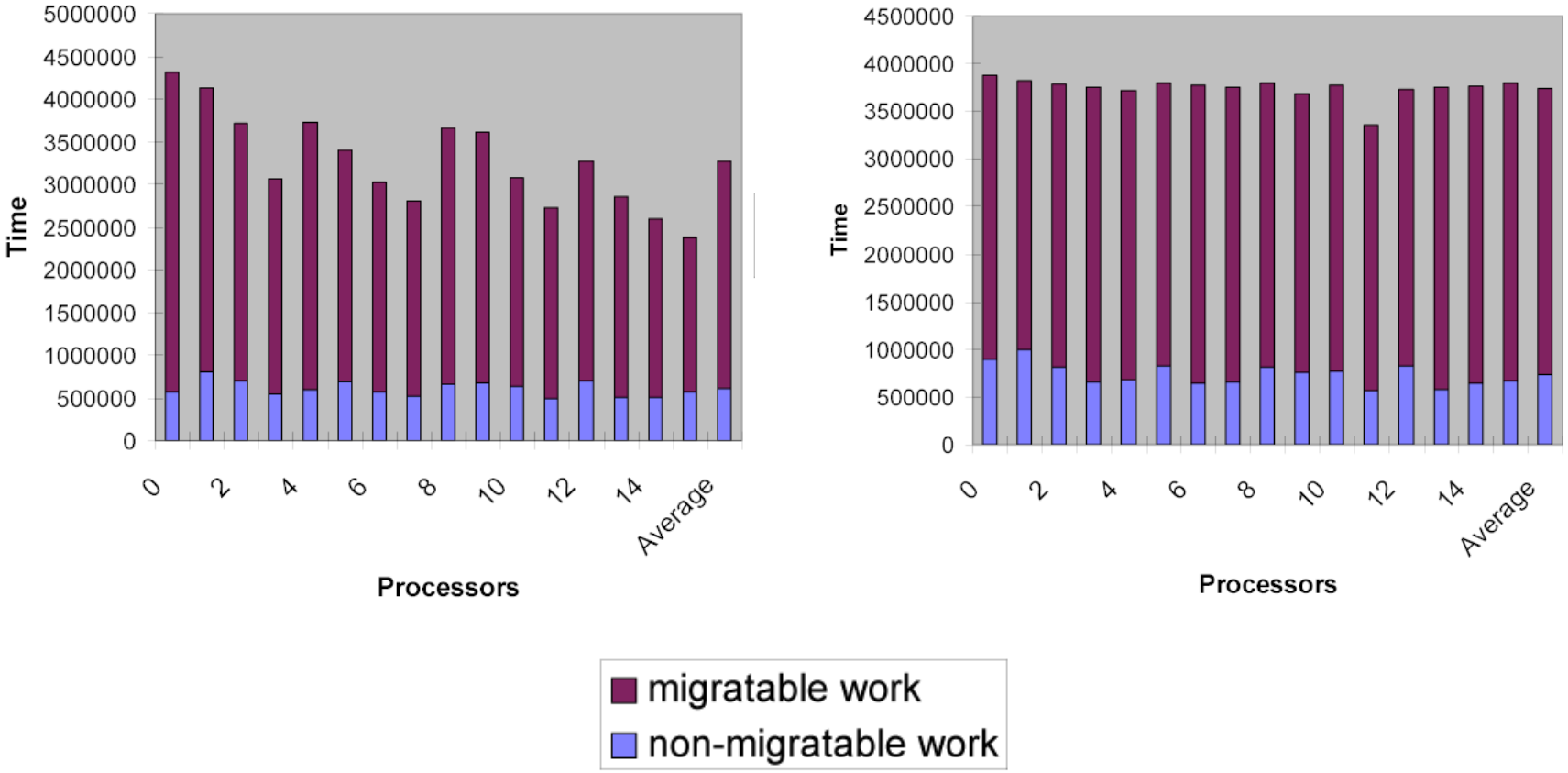
NAMD: Monitoring load

- Balance maintained by a *Distributed Load Balance Coordinator* which monitors on each processor:
  - Background load (non migratable work)
  - Idle time
  - Migratable compute objects and their associated compute load
  - The patches that compute objects depend upon
  - The home processor of each patch
  - The proxy patches required by each processor
- The monitored data is used to determine load balancing

NAMD: Dynamic load balancing

- Greedy load balancing strategy:
  - Sort migratable compute objects in order of heaviest load
  - Sort processors in order of “hungriest”
  - Share out compute objects so hungriest ranked processor gets largest compute object available
  - BUT: this does not take into account communication cost
- Modification:
  - Identify least loaded processors with:
    - Both patches or proxies to complete a compute object (no comms)
    - One patch necessary for a compute object (moderate comms)
    - No patches for a compute object (high comms)
  - Allocate compute object to processor giving best compromise in cost (compute plus communication).

NAMD: Impact of load balancing



Specialist hardware

- Anton MD machine
  - Designed specifically for protein/ water simulations
  - Investigate protein-folding challenge
- ASICs
  - High-throughput interaction subsystem - short range intermolecular interactions
  - Flexible subsystem – intramolecular interactions and FFTs (SPME)
- 512 ASICs in a 3D torus
- Performance
  - Over 17,000 ns of simulated time per day for a protein-water system consisting of 23,558 atoms on 512 cores.
  - Traditional HPC 256-1024 cores manage few hundred ns per day.