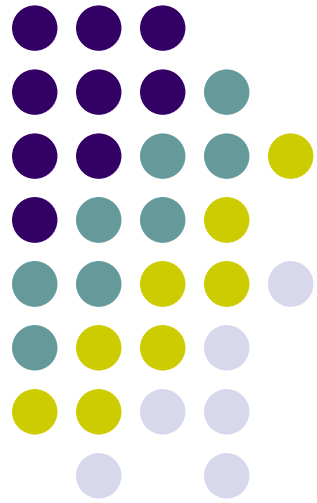
Parallel Algorithms



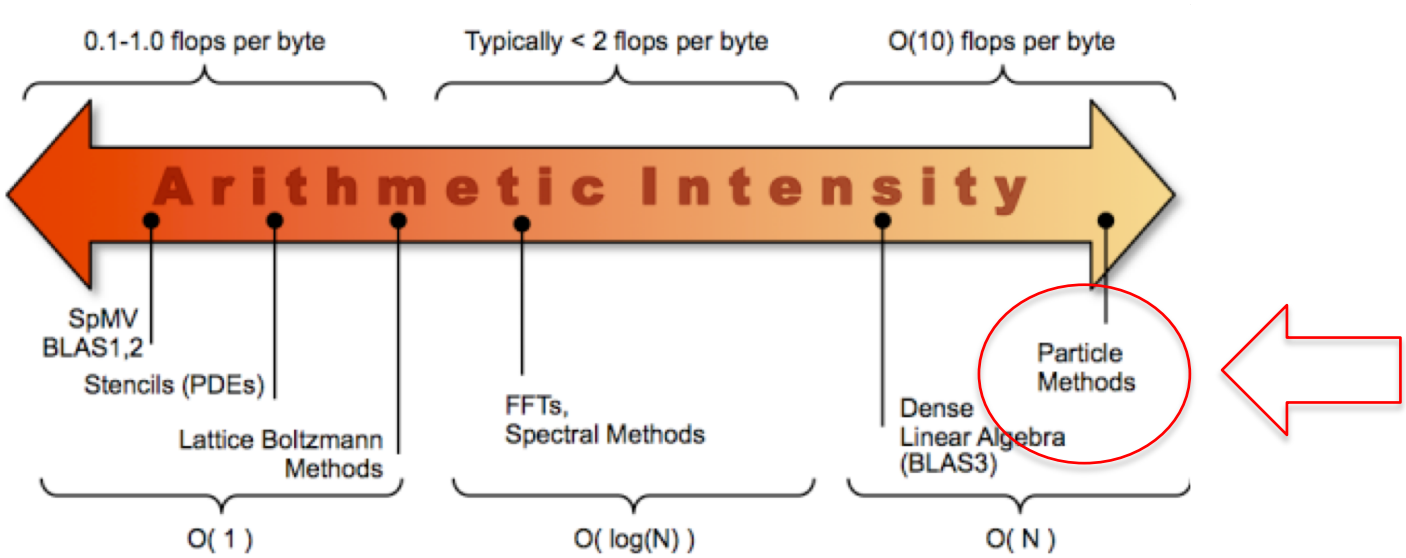
Molecular Dynamics Simulations / N-Body

João Sobral, Rui Silva
Departamento do Informática
Universidade do Minho

4 May 2021

Parallel Algorithms

Arithmetic Intensity – Molecular Dynamics



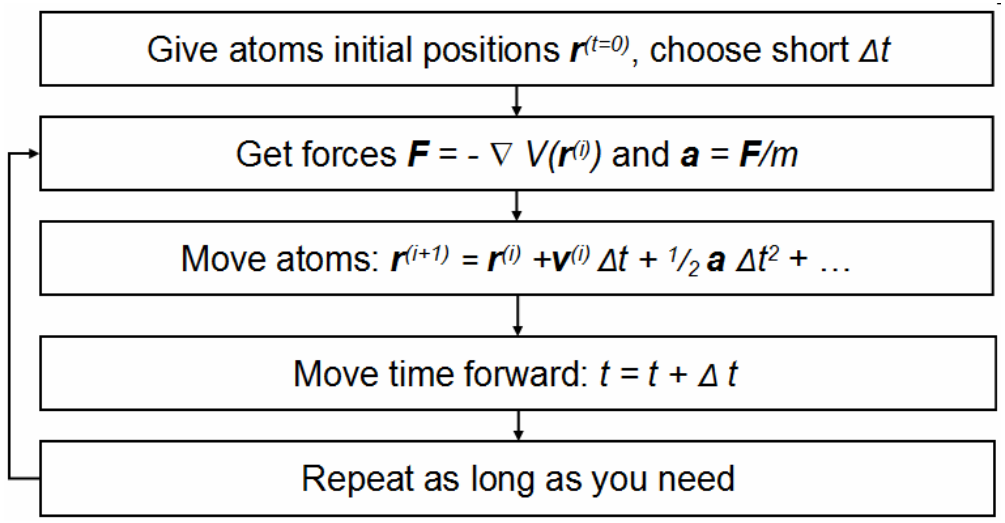
Data structures

- **Regular: Vector<Particles>**
- **Opt: Irregular: Map<Particle,Set<Particle>>**

Molecular Dynamics (MD) Simulations

- Computer simulation of processes that entail forces between particles and movement of bodies
  - **Chemistry: study of proteins (folding and kinetics)**
  - **Physics: thin-films**
  - **Nanotechnologies: device simulations**
- Belongs to a broader category of N-Body methods (presented at the end of this session)

Simplified code skeleton of a simulation



MD Simulations

Algorithm analysis

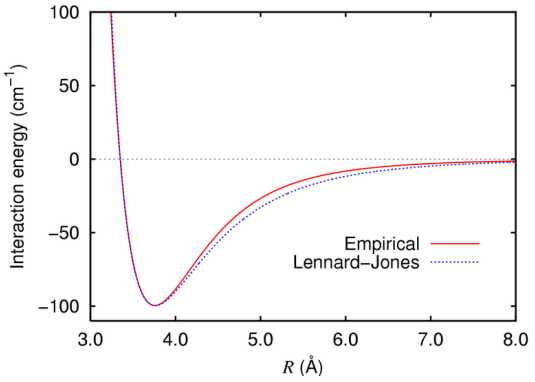
- Complexity: $O(n^2)$
  - Can be reduced to $O(nlog(n))$ on specific cases (use different algorithms for short and long distance forces)

- The simulation time also depends on ”Time” simulated and on the “Time-step” of the simulation
  - Common simulation times are between 1us and 1ns
  - Time-steps of **10^{-12} s**

Force computation: Lennard-Jones Potential

- Total energy is the sum of the interactions between all pair of particles in the system.
- Particle interactions are given by:

$$\phi(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$



  - r – distance between two particles
  - ε - field depth
  - σ - distance where the field is null.
  - ε and σ are specific of the material (generally 1 on MD simulations)

MD Simulations

Simulation Model (simplified)

- The potential system energy (V) can be the sum the interactions between all pairs of particles

$$V(r_1,\ldots,r_N) = \sum_i \sum_{j>i} \phi(|r_i - r_j|) \qquad K(t) = \frac{1}{2}\sum_i m_i[v_i(t)]^2$$

  - **Note: this approach is too limited for some kinds of material (e.g., metals)**
- Forces between particles are given by the gradient of the potential

$$F_i = -\nabla_{r_i}(r_1,\ldots,r_N) \qquad \mathbf{F}(r) = -\nabla V(r) = -\frac{d}{dr}V(r)\hat{\mathbf{r}} = 4\epsilon\left(12\frac{\sigma^{12}}{r^{13}} - 6\frac{\sigma^6}{r^7}\right)\hat{\mathbf{r}}$$

- Generally there is a radius R_c where the forces outside are considered null
  - **To avoid discontinuity, when a particle gets outsider that range, we discount** $\phi(R_c)$
  - **Frequent values of Rc are from 2,5r to 3,2r**
- How to deal with the simulations borders?
  - **Replicate the simulation (*periodic boundary conditions)***
  - **Use image sizes less than 2Rc**
- Integration algorithm (Verlet)

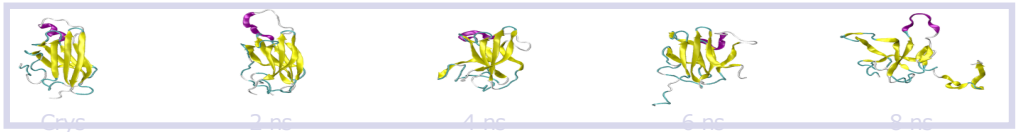
$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \mathbf{v}(t)\Delta t + (1/2)\mathbf{a}(t)\Delta t^2$$
$$\mathbf{v}(t+\Delta t/2) = \mathbf{v}(t) + (1/2)\mathbf{a}(t)\Delta t$$
$$\mathbf{a}(t+\Delta t) = -(1/m)\nabla V\left(\mathbf{r}(t+\Delta t)\right)$$
$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t+\Delta t/2) + (1/2)\mathbf{a}(t+\Delta t)\Delta t$$

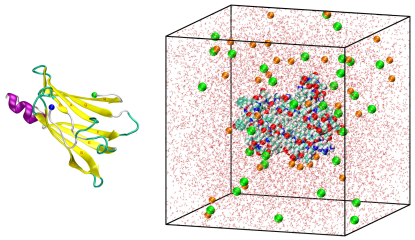
MD Simulations

Simulation examples

- **Satellite Tobacco Mosaic Virus – one of the smallest reproducing units in nature**
  - 2006, 1 million atoms for over 50 ns
  - NAMD software (would require 35 years of simulation on a desktop computer at the time)

- P-Found – L55P-TTR - Transtirretina, “Doença dos Pezinhos” (1:1000 in Portugal)
  - **2004, 44k** atoms **(**Protein: 1912 atoms; Water: 3*14137 atoms; Na^+ Cl^-: 71 ions**)**
  - **Computation run time: 4-6 weeks** using 8-12 Pentium-4 CPUs
  - **Binary file capturing the coordinates of all atoms: ≈ 4 GB**
  - **Binary file capturing the coordinates of protein’s atoms: ≈ 180 MB**

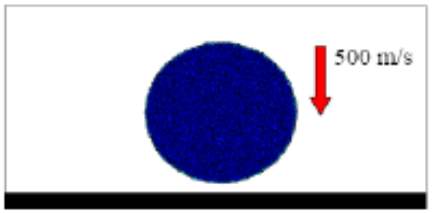




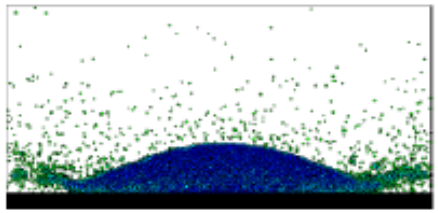
Molecular Dynamics

Example: collision of a droplet with a substrate

(by Yasushi Katsumi)



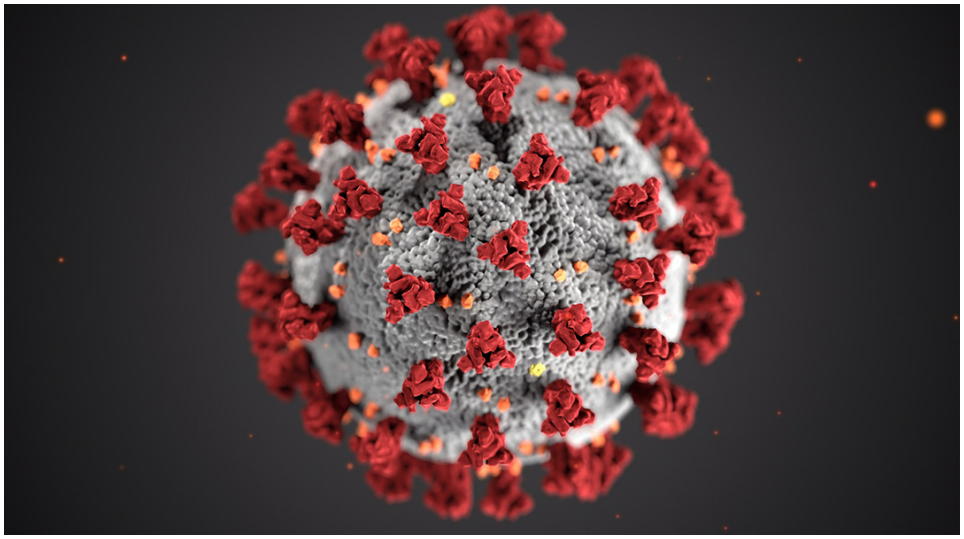
Initial conditions are specified, $r_i(t_0)$ and $v_i(t_0)$



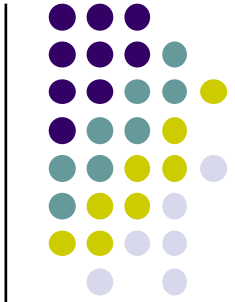
Snapshot from MD simulation at time t_n = 100 ps

MD Simulations

Supercomputer simulates molecular model of SARS-CoV-2



- Rommie Amaro is leading efforts to build the first complete **all-atom model** of the SARS-COV-2 coronavirus envelope
- The coronavirus model is anticipated contain roughly **200 million atoms**
- The Frontera supercomputer aided efforts of the Amaro Lab on March 12-13, 2020, by running NAMD molecular dynamics simulations on up to 4,000 nodes, or about **250,000** cores.



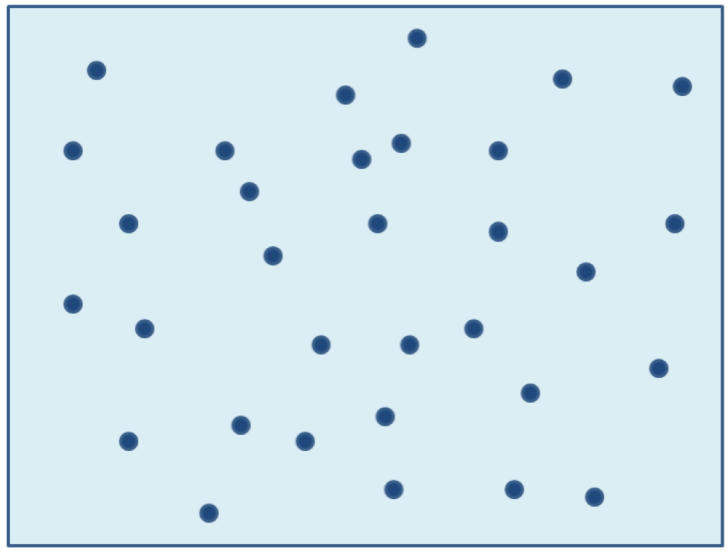
MD Simulations

Optimizations to the sequential code

- Domain specific
  - 3rd newton law
  - Cut-off radius
    - Neighbours list
    - Cell division
- Generic
  - Improve spatial and temporal locality
  - Vectorization

Base implementation - All pairs

- All particles interact with all others
- Ease of implementation
- Complexity N^2

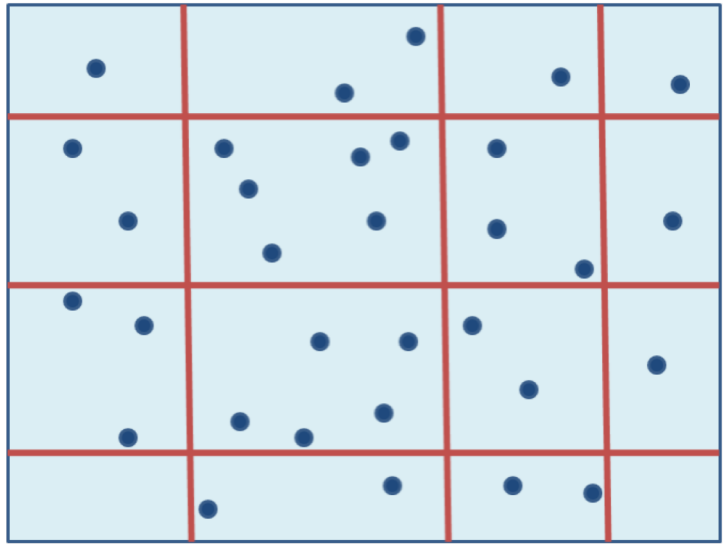


Cell Division (domain decomposition)

- Partition of particles in space
- Reduces the calculations when compared with all-2-all approach
- Improves locality of memory accesses

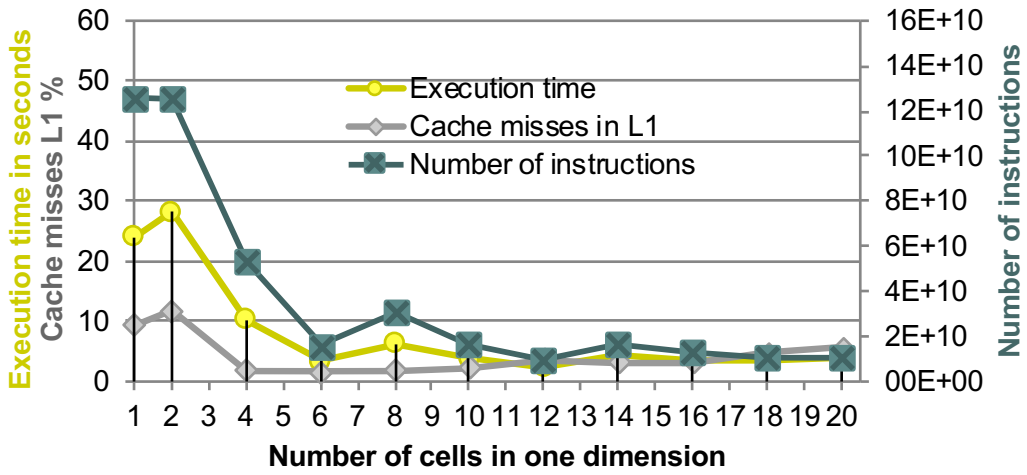
  if “2 x dim(Cell)” fits in cache

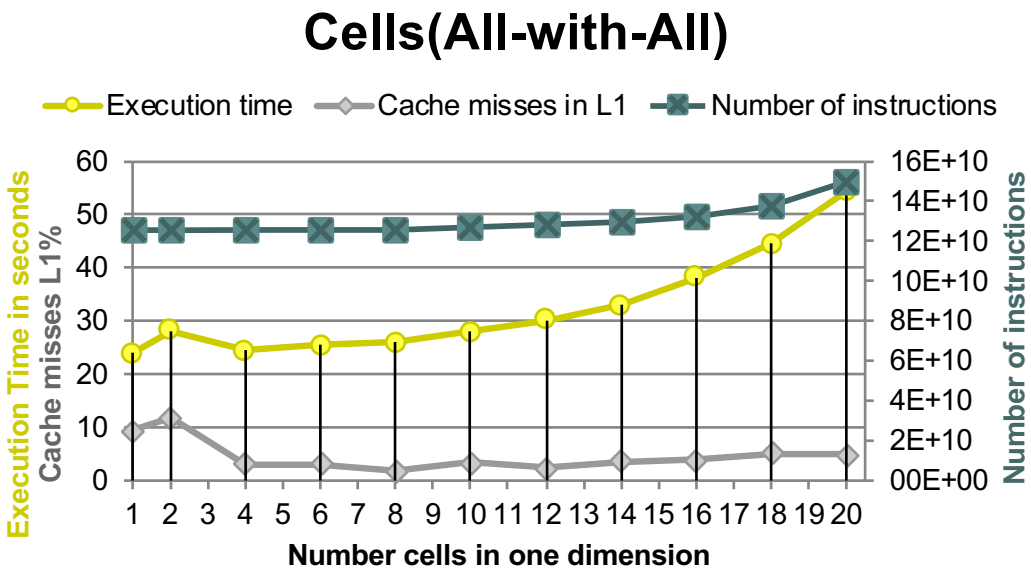
- Has to deal with Particle moves across cells



Cells optimization profiling

- Reduction of cache misses
- Decreases the number of instructions
- Minimum execution time when cell size is multiple of cut-off radius





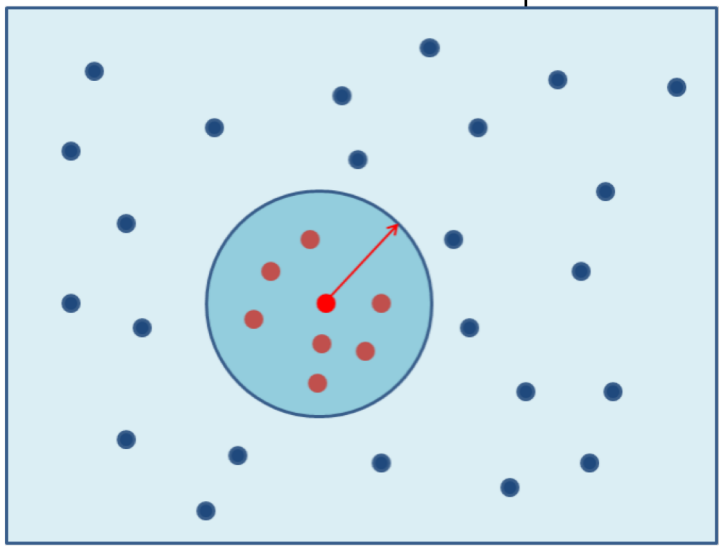
Neighbours list

Every particle contains one list of neighbour particles

Further reduces the calculations

It is not necessary to update the list at all iterations

- **Use an extra radius delta**
- **Reduces overhead of list maintenance**

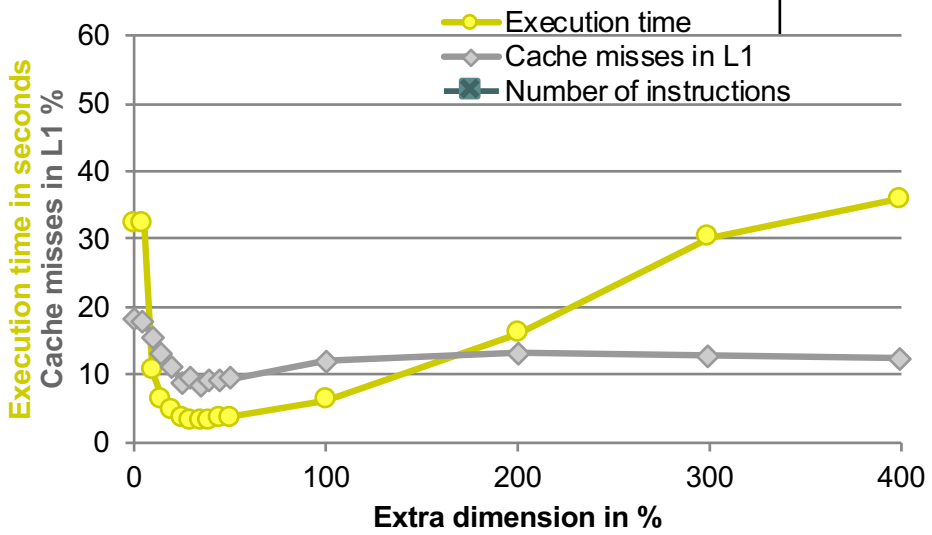


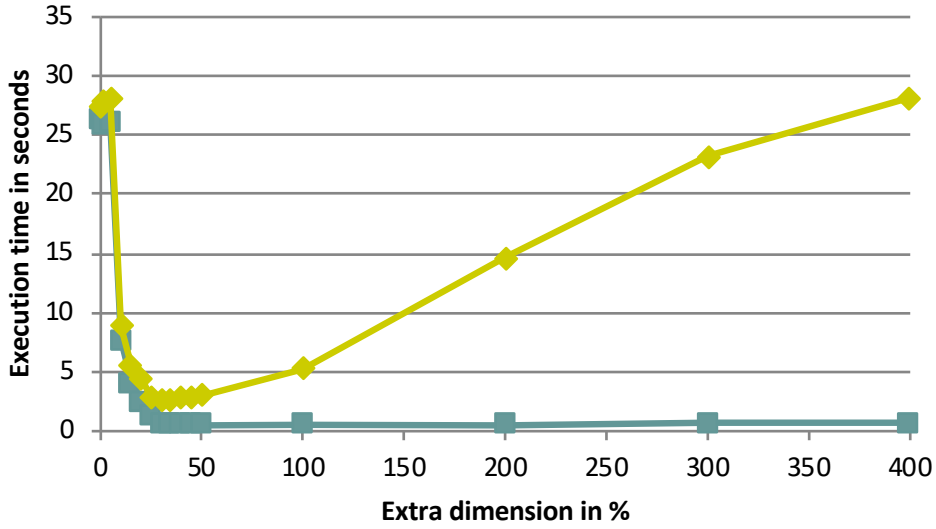
- Irregular data structure

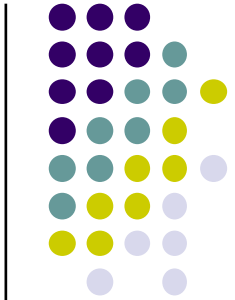
  Map<Particle,Set<Particle>>

Neighbour list optimization profiling

- High number of cache misses
- Decreases number of instructions
- List update can introduce an high overhead







Molecular Dynamics Simulation

Options to explore parallelism

- Decomposition of the set of particles
- Decomposition of the set of forces
- Domain decomposition (cells)

Parallel Implementations

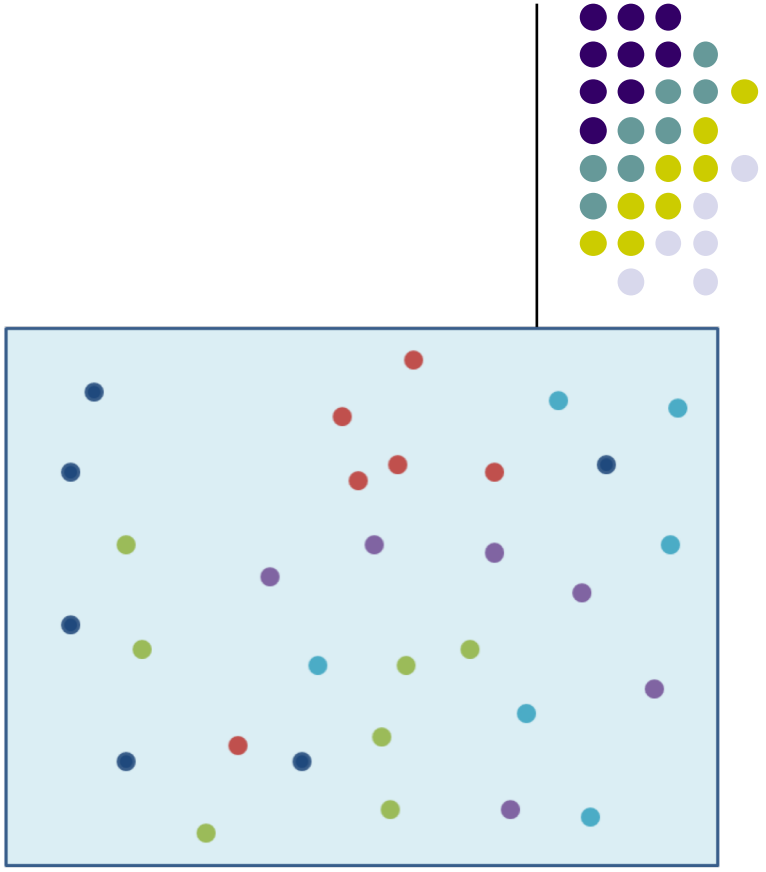
- NAMD
- GROMACS
- MOIL

Particle decomposition

The particles are associated with a processing element

Different particles have different numbers of interactions

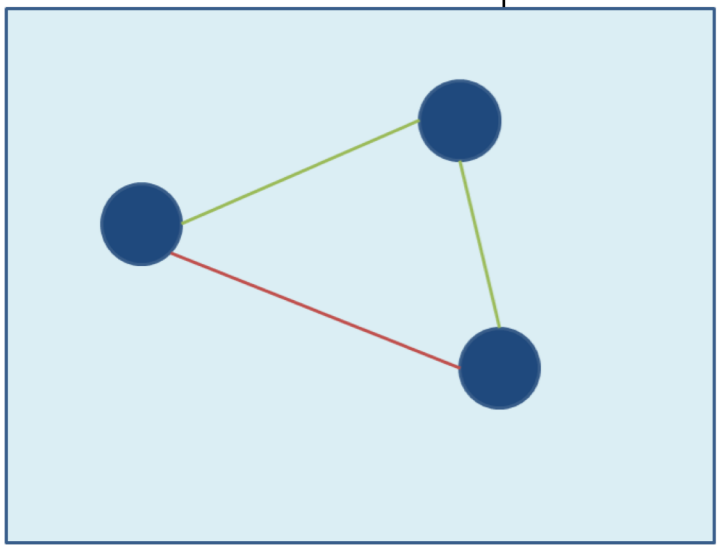
Each PE has to know the positions of the particles (requires global communication)



Force decomposition

Division of interactions among PEs

Good load balancing



Space (cells) decomposition

The partition is based on the cell division

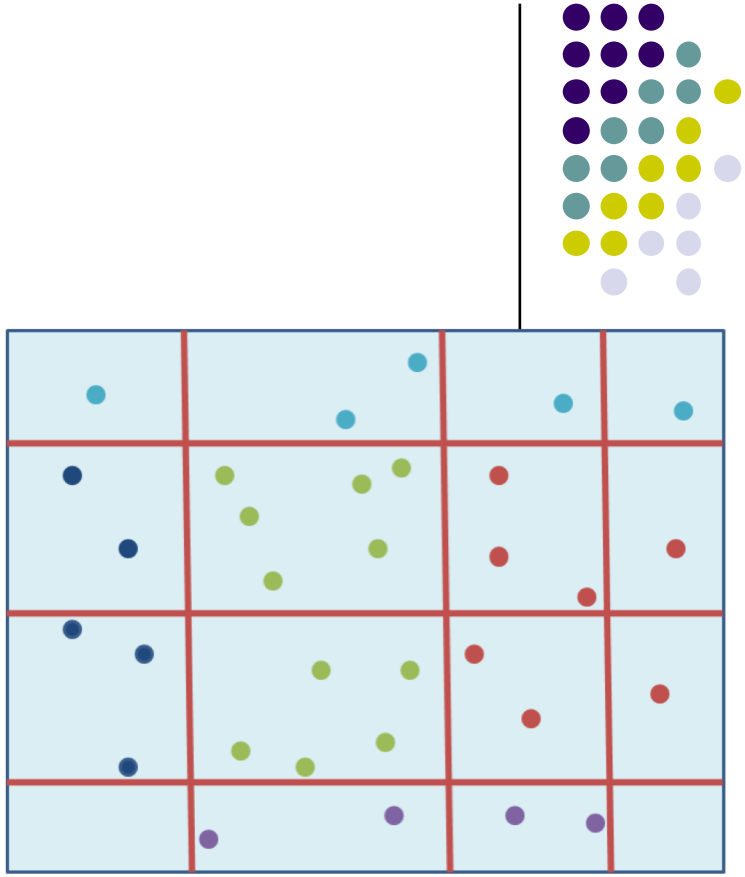
Each PE is connected to one or more cells

Each cell needs the positions of the particles from neighbouring cells

- **depends on the cut-off/cell ratio**

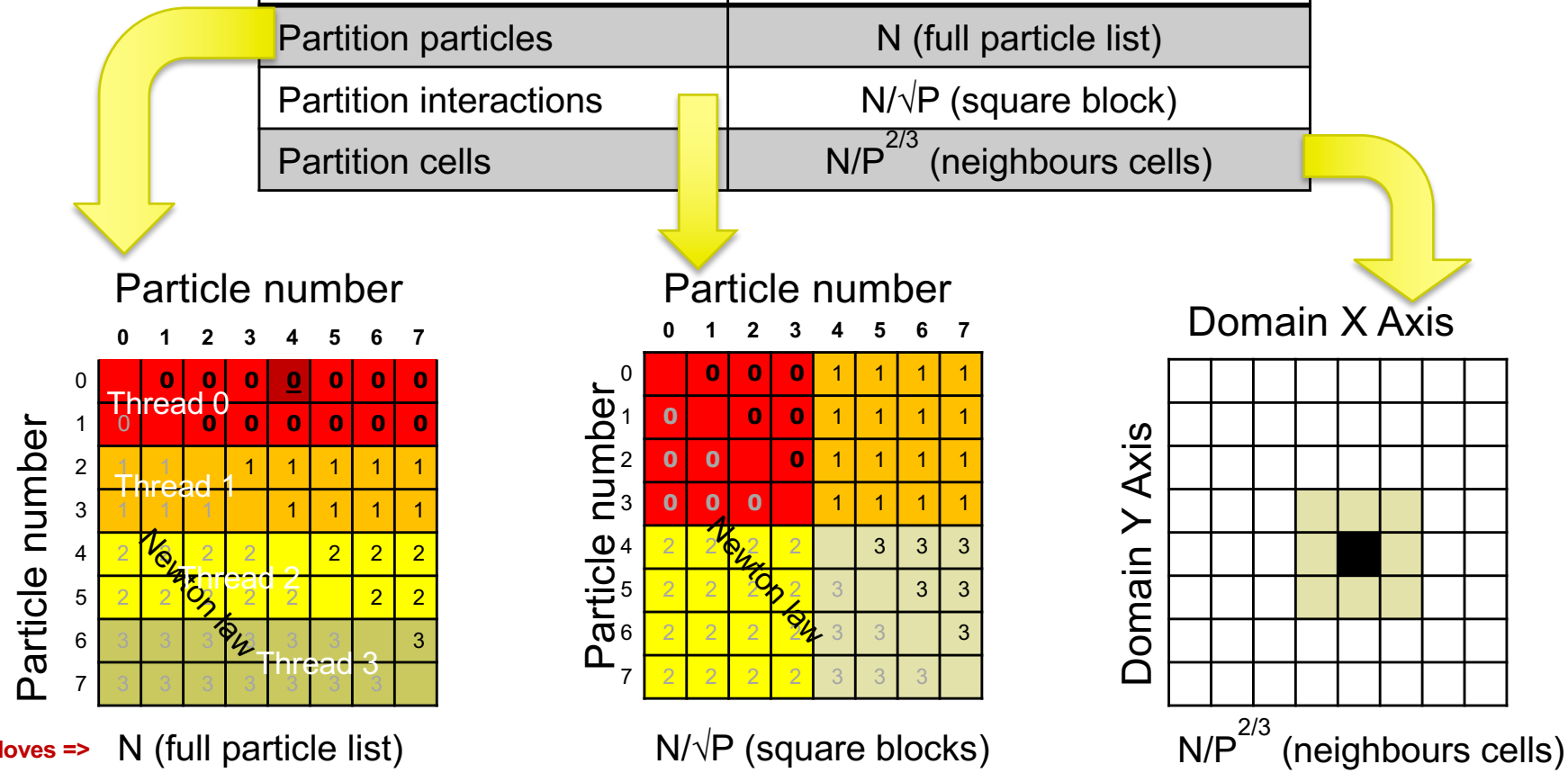
Particle migration among PE

Problems with load balancing



Parallel algorithm complexity

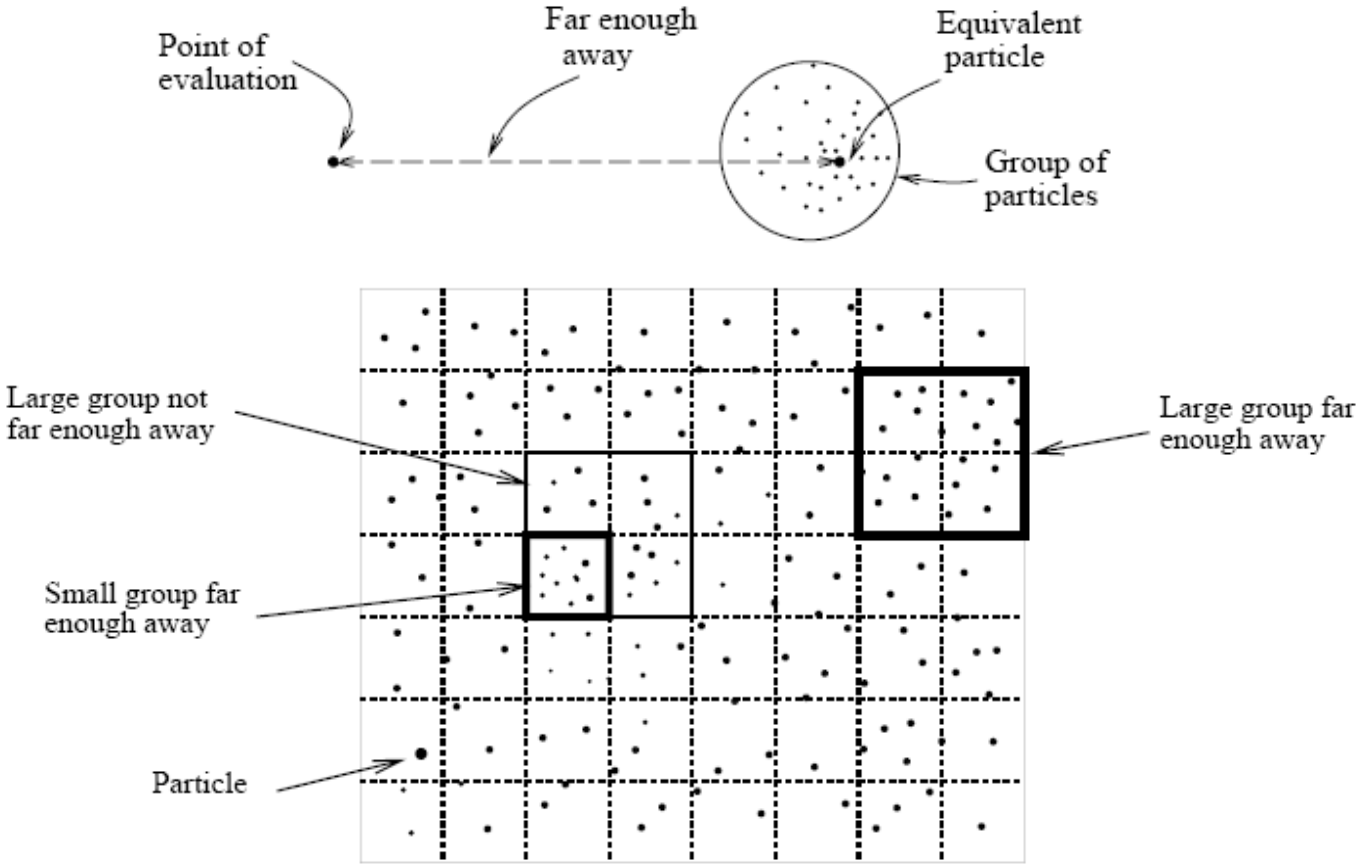
	Data moves per PE
Partition particles	N (full particle list)
Partition interactions	N/√P (square block)
Partition cells	$N/P^{2/3}$ (neighbours cells)

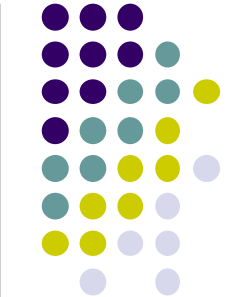


Force matrix: F_{ij} is the computation of force between particle i and particle j. In the left example F_{04} is computed by Thread 0. Newton Law states that forces are symmetric $F_{04}=F_{40}$

N-Body Methods

Distant points have less influence for force calculation

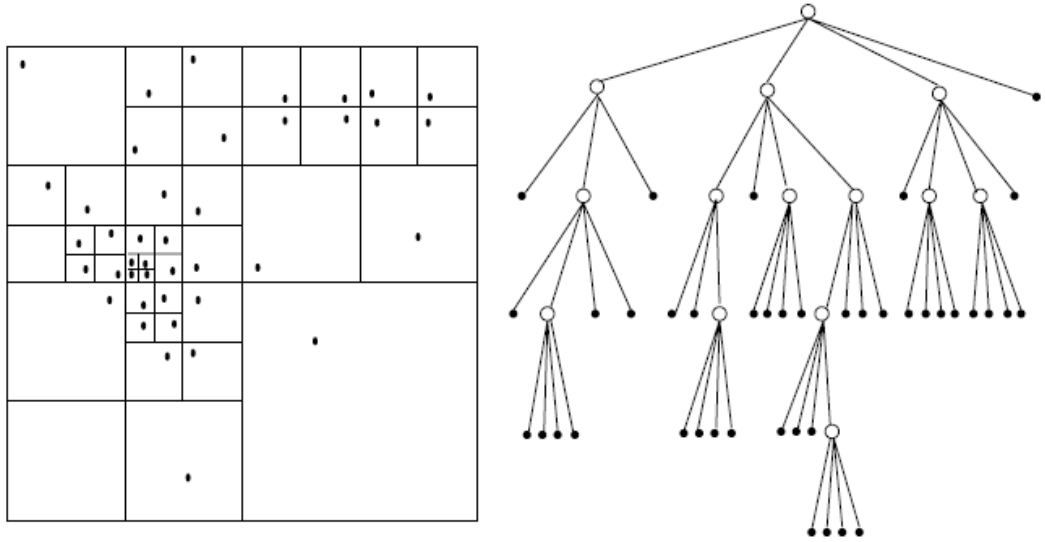




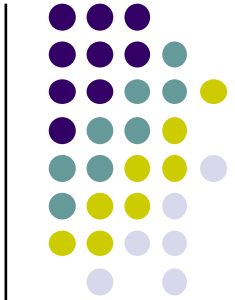
N-Body Methods

Barnes-Hut

- The space is subdivided until there is only one particle per cell.



- The force computation becomes a tree traversal
  - **Close enough particles are directly computed (leave to leave)**
  - **Far away particles are replaced by the root of the their tree**
- Parallelism
  - **On force computation (the tree does not change during computation)**
  - **Tree reorganization**



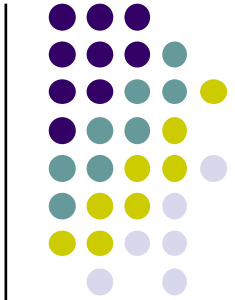
Métodos *N-Body* Hierárquicos

Sobrecargas de paralelismo

- Computações sequenciais
- Trabalho redundante
- Gestão do paralelismo
- Sincronização
- Comunicação
- Má distribuição da carga

Compromisso entre distribuição da carga e a localidade de acesso aos dados

- A unidade de paralelismo por célula (ou partícula) é irregular (depende da distribuição da partículas)
  - **Partições com o mesmo número de partículas não têm a mesma carga de trabalho**
  - **Solução: atribuir uma função de custo aos cálculos associados a cada partícula e utilizar esse custo para distribuir a carga**
- A comunicação é de grão-fino, global e não estruturada
- A distribuição das partículas varia durante a execução (lentamente)
  - **Partição estática não garante a localidade física (partículas próximas no espaço devem ser atribuídas ao mesmo processador)**
  - **Solução: efectuar a partição da árvore**



PL lecture 2

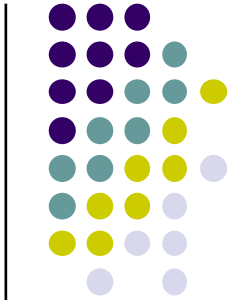
Parallelize the given MD simulation code

- **Shared Memory: Use OpenMP & gcc 5.3.0 (or gcc 7.2.0)**
- **Use input of size of 2** (program correctness can be verified with size 0 or 1)

Suggestion

- Profile the code execution and identify the function responsible by the force computation
- Identify the three lines responsible by the 3rd newton law optimization inside the force computation
- Start with a naive OpenMP particle-based parallelisation taking care of the data races
- Optimize the parallel implementation (optional)

Particle methods



Skeleton of the provided code

DataTypes.h

```
typedef struct MD{
    int ITERS,size,interactions, mdsize, move, mm;
    int irep,istop ,iprint ,movemx;

    double LENGTH,m,mu,kb,TSIM,deltat,epot,vir;
    double l, rcoff, rcoffs, side, sideh, hsq, hsq2;
    double a, r, tscale, sc,ek, ts;
    double den,tref,h;

    double vaver, vaverh, rand2;
    double etot, temp, pres, rp;
    double sum,ekin,count,vel;
  }MD;

typedef struct PARTICLE{
    double *posX,*posY,*posZ; // Posições
    double *vX,  *vY,  *vZ;   // Velocidades
    double *fX,  *fY,  *fZ;   // Forças
 } Particles;
```

MD.C

```
void runiters(MD *md, Particles *part){

    for (md->move = 0; md->move < md->movemx; md->move++) {

       cicleDoMove         (md,part);  // Calcular o movimento
       cicleForces         (md,part);  // Calcular a força
       cicleMkekin         (md,part);  // Scale forces, update velocities
       cicleVelavg         (md,part);  // calcular a velocidade
       scale_temperature   (md,part);  // temperature scale if required
       get_full_potential_energy(md);       // sum to get full potential energy and virial
    }
}

/* move the particles and update velocities */
void cicleDoMove(MD *md, Particles *part){
    for (int i = 0; i < md->mdsize; i++)       domove(md->side,part,i);
}

/* compute forces */
void cicleForces(MD *md,Particles *part){

    md->epot = md->vir = 0.0;

     for (int i = 0; i < md->mdsize; i++)
            force(md,part,i);
}
```