Particle Based Domain Decomposition

Matthias Müller
University of Stuttgart
High Performance Computing Center Stuttgart (HLRS)
www.hlrs.de

Particle Methods

- Molecular Dynamics (MD)
- Direct Simulation Monte Carlo (DSMC)
- Event Driven / Hard Sphere MD (ED)
- Smoothed Particle Hydrodynamics (SPH)
- Lattice Gas Automata (LGA)
- Boltzmann Lattice Gas Automata (BLGA)
- Contact Dynamics
- .... and many more
Differences to Grid based algorithms

- Equation of motion:
  \[ \frac{\partial^2}{\partial t^2} x_i = F(x_1, \ldots, x_N) \]
- One possibility: treat with standard methods for ordinary differential equation
- BUT: the Grid is moving every timesteps:
  - Grid/particle distribution should be cheap
  - possible load imbalance problems
  - physics may help to find efficient algorithm

General questions of the physical approach

- Type of interaction
  - pair interaction (superposition of forces)
  - multi-particle interaction
- Range of interaction
  - long range (gravitation) (Hierarchical methods, fast multipole methods)
  - short range
- Distribution of particles in space
  - homogeneous/inhomogeneous
  - bounded/unbounded
  - regular/irregular shape of domain
- Necessary number of particles
Case study: short range Molecular Dynamics

- N particles interact with each other

$$m_i \frac{\partial}{\partial t} v_i = F_i = \sum_{j=1}^{N} F_{ij} = \sum_{j=1}^{N} F(r_i, r_j)$$

- Due to the short range interaction the Force Matrix $F$ is a sparse Matrix

Approach 1: Particle Decomposition

- Every processor gets $N/P$ particles
- Every processor calculates the forces on its particles and keeps track of their position
- To calculate the forces, every processor needs to know all other particle positions (all-to-all communication)
Approach 1: Particle Decomposition

+ No load imbalance if workload linear to particle number
+ Also works for long range interaction

- Lot of communication
- Huge memory demand on every processor

Approach 2: Force Decomposition

• Every Processor calculates a part of the Force Matrix
• Only Knowledge of 2*N/sqrt(P) particles needed
Advantages and Disadvantages of Approach 2

+ Only Knowledge of $2^N \sqrt{P}$ particles needed, less memory consumption
+ Good load balance if interaction between particles is “random”

- Does not scale with N/P

Approach 3: Domain Decomposition

- Every Processor is responsible for a domain in real space
- It calculates the force on the particles in this domain
- Only N/P particle positions are stored
Approach 3: Domain Decomposition

- Force Matrix in case of Domain Decomposition:

```
1 | x | x | x | x | x | x | x | x | x | x | x | x | x | x |
2 | x | x | x | x | x | x | x | x | x | x | x | x | x | x |
3 | x | x | x | x | x | x | x | x | x | x | x | x | x | x |
4 | x | x | x | x | x | x | x | x | x | x | x | x | x | x |
```

Advantages and Disadvantages of Approach 3

+ Scales with N/P
+ Little memory consumption, all information is distributed across CPUs

- Possible load imbalance if particles distribution is not homogeneous in space
- Algorithm is more complicated to implement
Comparison of Different Approaches

![Graph showing CPU time vs. Particle Number for different approaches: Particle, Force, Domain.](image)

Molecular Dynamics with Verlet Neighbor Lists

- Basic idea: limit the calculation of forces to particles within interaction range

- Problem: creation of verlet neighbor lists is still of order $N^2$
Molecular Dynamics with Linked Cell Algorithm

- Calculation of forces with linked cell algorithm
  - cells with length >= cut-off-length of interaction
  - interaction limited to cell and its direct neighbors
  - use of actio=reactio
  - results in O(N) algorithm

Parallel Molecular Dynamics

- Grid of linked cell algorithm is useful for parallel algorithm
- The information about particles in cells at processor boundaries needs to be communicated
Details of Communication: General Pattern

- Solution 1: Direct Communication
  Send all data directly to the neighbors
  (27 in three Dimension, 9 in two)
- Solution 2: Plimpton scheme:

  ![Diagram showing communication pattern]

  Advantage: only 6 communications (4 for two dimension)
  Disadvantage: second stage can only start after first has finished

Details of Communication: actio=reactio across CPUs

- Approach 1: calculate once, communicate results to all CPUs where
  this particle is in ghost cells.
- Approach 2: don’t use actio=reactio across CPU boundaries.
  Calculate forces with ghost particles twice (or several times in
  corners).

Which approach is better depends on your communication and
computation speed. In other words: how many data your particle
contains and how expensive your force calculation is.
The general rule is, that communication is more expensive than
computation.
Details of Communication: Problem of buffer space

- Problem: the exact amount of data is unknown at the receiver
  
  ```
  MPI_Send(buffer, count, my_type, dest, tag, comm);
  MPI_Recv(buffer, max_count, my_type, ...);
  ```

- Solutions:
  - `max_count` might be known
  - Use `MPI_Probe` to determine size of pending message
  - Send two messages, the first announces the amount of data
  - Use `MPI_Pack` and use the first item to distinguish whether the message is a request to increase buffer at the receiver. Manage receiver buffer size also at the sender.

Load Balancing

- For inhomogeneous density you will need static or dynamic load balancing.
- General load balancing will change your topology between processors.

![Load Balancing Diagram](image-url)
Simple Load Balancing

- Keep Topology unchanged
- Results in good load balancing for quasi-one-dimensional density gradients.
- You have to check your physics, whether this holds

Summary

- Which approach is the best depends on your specific problem
- Some basic principles and problems are general
  - unknown particle numbers in communication
  - general scaling, but crossover will depend on interaction
  - load balancing problems
- Even a simple algorithm gets complicated when it comes to the details