Parallelization of Molecular Dynamics
(with focus on Gromacs)
Outline

• A few words on MD applications and the GROMACS package
• The main work in an MD simulation
• Parallelization
• Stream computing
Supercomputers

Lindgren (PDC)
Cray XE6
36 000 AMD cores
305 teraflops

K computer (Riken, JP)
705 024 SPARC64 cores
10 petaflops
(faster than the next 5 computers together)

Cray XK computer (just on market)
Nodes:
2 AMD 16-core Interlagos processors
2 NVidia Tesla X2090 GPUs
Challenges for exascale computing

Issues:

• Power usage (K computer: 10 MW, cooling doubles this)
• Reliability, chance of components failing increases
• Interconnect speed needs to keep up with millions of cores
• Data handling
• Software does not scale!

Question:

• Do we really need an exaflop computer?
• Does that get us better science?
Two general computing trends

Over the past decades:
• Moore’s law transistor count doubles every two years
• Processor speed doubles every 18 months
• Memory speed does not increase this fast
• Result: memory access relatively expensive!

Second important development:
• Past two decades: slow increase of parallel computing
• This decade: no MHz increase, instead core count increase
• Arrival of GPGPUs: even mores cores!
• Result: parallel algorithms very important
Other “typical” HPC applications, e.g. turbulence:
  • Weak scaling: scale problem size with computer size
  • Calculation times stay the same

Molecular dynamics:
  • Little increase in problem size
  • Time per step decrease (sub millisecond)
  • Need lots of computing power, but use of supercomputers is challenging
  • Algorithms need to be re-designed
How to write efficient code for MD?

Hardware and software:
- Multi-core CPUs: C (SIMD intrinsics?) or Fortran + MPI + OpenMP
- GPUs: CUDA or OpenCL
- Intel Xeon 5 (Larabee, MIC, ...): C or something else?
- FPGAs?
- OpenACC for everything: much less work

Molecular dynamics need sub-millisecond iterations times: we need low-level optimization: C + SIMD intrinsics + CUDA

A lot of work, but it might be worth it
### Classical molecular simulation

#### Common applications:

- **“simple” liquids**: $10^3 - 10^4$ atoms, ns - $\mu$s
- **peptide/protein folding**: $10^4 - 10^5$ atoms, $\mu$s - s
- **protein functional motions**: $10^5 - 10^6$ atoms, $\mu$s - s
- **polymers**: $10^4 - 10^6$ atoms, $\mu$s - ?
- **materials science**: $10^4 - 10^8$ atoms, ns - ?
Simulations of bio-molecules

- Proteins, DNA, RNA
- Fixed system sizes
- Functional motions
- Overdamped dynamics
- Stochastic kinetics
- Need to sample many events
- The time scales often increase exponentially with system size
Simulations in (Chemical) Physics

Simulations of biomass for bio-ethanol
(millions of atoms)

Understanding and controlling interactions of droplets on surfaces
(100 million particles)
Molecular Dynamics

Basically solving Newton’s equation of motion:

\[ m_i \frac{d^2 x_i}{dt^2} = -\nabla_i V(x) = F_i(x) \quad i = 1, \ldots, N_{\text{atoms}} \]

Symplectic leap-frog integrator:

\[ v_i(n + \frac{1}{2}) = v_i(n - \frac{1}{2}) + \frac{\Delta t}{m_i} F_i(x(n)) \]

\[ x_i(n + 1) = x_i(n) + \Delta t \, v_i(n + \frac{1}{2}) \]

\[ V(x) = \sum_{\text{bondeds}} V_b(x) + \sum_{i<j} A_{ij} r_{ij}^{-12} - B_{ij} r_{ij}^{-6} + \frac{q_i q_j}{r_{ij}} \]

Computational cost:

- integration: \( c N \)
- force calculation: \( C N^2 \)
Setting up simulations requires a lot of physical/chemical/biological knowledge

The force field (parameters) is critical:
the results are as good/bad as the force field

Developing algorithms for simulations is a completely different business, but you need to know the application needs
All forces are short ranged, except for the Coulomb forces.

Thus use a (smooth) cut-off for the particle-particle interactions and do the remaining part of the Coulomb interaction on a grid.

Particle Mesh Ewald electrostatics:

Solve the Poisson equation in reciprocal space using a 3D FFT.
## Computational breakdown

<table>
<thead>
<tr>
<th>computation</th>
<th>cost</th>
<th>communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>bonded forces</td>
<td>$c O(N)$</td>
<td>along with nbf</td>
</tr>
<tr>
<td>non-bonded forces</td>
<td>$c M \times N$</td>
<td>local</td>
</tr>
<tr>
<td>spread charge</td>
<td>$C N$</td>
<td>local</td>
</tr>
<tr>
<td>3D FFT</td>
<td>$c N \log N$</td>
<td>global</td>
</tr>
<tr>
<td>gather forces</td>
<td>$C N$</td>
<td>local</td>
</tr>
<tr>
<td>integration</td>
<td>$c N$</td>
<td></td>
</tr>
<tr>
<td>constraints</td>
<td>$c N$</td>
<td>local</td>
</tr>
<tr>
<td>virtual sites</td>
<td>$c N$</td>
<td>local</td>
</tr>
</tbody>
</table>

Time step: 2 - 5 fs, simulation length $10^7$ to $10^9$ steps.

A step takes 1 - 100 milliseconds.
How many pair interactions per second?

Intel Core i7, 2.67 GHz
4 cores + Hyperthreading

NVidia GTX580, 1.5 GHz
512 CUDA Cores
CPU vs GPU

How many pair interactions per second?

Intel Core i7, 2.67 GHz
4 cores + Hyperthreading

192 million per core
960 million total

NVidia GTX580, 1.5 GHz
512 CUDA Cores

3200 million total
For many applications we would like orders of magnitude more sampling

Speed up simulations through:

• Increasing the time step (constraints, virtual interaction sites)
• Use less particles (triclinic, more spherical periodic cells)
• Use more processors: parallelize
• Use stream computing

Parallelization nowadays is MPI + threads
Scaling limits

Strong scaling limited by:
- all communication latencies/bandwith and load imbalance

Weak scaling limited by:
- electrostatics global communication

Electrostatics solvers:
- PME/PPPM: $O(N \log N)$, small pre-factor
- Multi level methods: $O(N)$, large pre-factor
- Fast multipole: $O(N)$, discontinuous gradient

Note:
(nearly) embarrassing parallelizm, run 1 - 1000 simulations in parallel
GROMACS simulation package

Started at the University of Groningen in the early 1990s.
Now all main developers are in Sweden (Stockholm/Uppsala)

Thousands of users world wide
A million users through Folding@home

- Open source (GPL), www.gromacs.org
- Support for all major biomolecular force fields
- Highly optimized algorithms and code
- Advanced algorithms for increasing the time step
- Language: C and essential parts in assembly: SSE2/Altivec/CUDA/...
- Efficient single processor performance
- Since three years efficient parallelization

### Domain decomposition methods

#### (a) (b) (c)

<table>
<thead>
<tr>
<th>comm.</th>
<th>cut-off</th>
<th>Half Shell</th>
<th>Eighth Shell</th>
<th>Midpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>#cells</td>
<td>$r_c &lt; L_d$</td>
<td>13</td>
<td>7</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>$r_c &lt; 2L_d$</td>
<td>62</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>volume</td>
<td>$r_c = \frac{1}{2}L_d$</td>
<td>$2.94 \ L^3_d$</td>
<td>$2.15 \ L^3_d$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$r_c = L_d$</td>
<td>$9.81 \ L^3_d$</td>
<td>$5.88 \ L^3_d$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$r_c \to \infty$</td>
<td>$\frac{1}{2} \ \text{sphere}$</td>
<td>$\frac{1}{8} \ \text{sphere}$</td>
<td></td>
</tr>
</tbody>
</table>


Midpoint: Bowers, Dror, Shaw; JCP 124, 184109 (2006)
Load imbalance can occur due to three reasons:

- imhomogeneous particle distribution
- inhomogeneous interaction cost distribution (charged/uncharged, water/non-water due to GROMACS water innerloops)
- statistical fluctuation (only with small particle numbers)
Load imbalance can occur due to three reasons:

- imhomogeneous particle distribution
- inhomogeneous interaction cost distribution (charged/uncharged, water/non-water due to GROMACS water innerloops)
- statistical fluctuation (only with small particle numbers)

So we need a dynamic load balancing algorithm where the volume of each domain decomposition cell can be adjusted independently.
Triclinic unit cells with load balancing
Dynamic load balancing in action
Multiple-program, multiple data PME

**SP–MD**

- PP
- PME

**MP–MD**

- PP
- PME

**Further advantage:** 4 to 16 times less MPI messages

**Enables 4x further scaling!**

- Latency 4/3 lower
- Bandwidth 4/3 higher
- Latency 4 x lower
Flow chart, CPU only

PP process
- send x to PME
  - comm. x
    - calculate f
      - reduce f
        - receive f from PME
          - integrate eq. of m.
            - apply constraints

PME mesh process
- receive x from PP
  - redistribute x
    - spread on grid
      - 3D FFT
        - solve
          - 3D FFT
            - spread force
              - redistribute f
                - send f to PP
2D PME (pencil) decomposition

cellulose + lignocellulose + water
2.2 million atoms
Cray XT5

![Graph showing performance (ns/day) vs. #cores for 1D PME decomp., 2D PME decomp., and 2D PME, with "cube".](image)
How far can we scale?

100 million atoms
half protein half water
no PME (!)

Jaguar Cray XT5
at Oak Ridge

Issues:
• a $\mu$s is far too short for a 100 million atom system
• no full electrostatics
How far can we scale?

100 million atoms
half protein half water
no PME (!)

Jaguar Cray XT5
at Oak Ridge

Issues:
• a $\mu$s is far too short for a 100 million atom system
• no full electrostatics

Solutions:
• combine thread and MPI parallelization (a lot of work)
• develop electrostatics methods with less communication
Stream calculations

Single CPU-code: focus on functions

Stream calculations: focus on data streams, operations done in parallel

Common stream type architectures:

- SSE: operations on 4 floats at once (or 2 doubles)
- AVX: operations on 8 floats at once (Intel Sandy Bridge, AMD Bulldozer)
- CUDA (NVidia GPUs): operations of 32 floats/doubles at once

So 4 to 32 times speedup, assuming the same #cores and frequency
Hybrid acceleration

- Simulation box
- Compute node
- GPU only does non-bonded
  Re-use most of the CPU code
  Use the GPU for what it’s good at
Cut-off schemes in Gromacs

Gromacs $\leq 4.5$ “group” cut-off scheme, based on (ancient) charge-groups

Gromacs 4.6 “Verlet” buffered cut-off scheme optional

Gromacs 5.0 “Verlet” cut-off scheme default

Group cut-off scheme:

- Fast, since we process groups of atoms at once; water is a group $\Rightarrow$ very fast
- No Verlet list buffer by default, but optional: energy drift

Verlet cut-off scheme:

- Fast on wide SIMD and GPUs
- Verlet list buffer by default
- Supports OpenMP parallelization (+MPI)
Later today

- What is stream computing (SIMD, GPUs, CUDA)
- How to use it efficiently in MD