N-Body Simulation of High-Density Magneto-Optical Traps (MOT)

Documentation & Algorithm Analysis

November 26, 2025

Abstract

This document describes the physics, algorithms, and usage of the `mot_simulation.py` code. The software simulates the dynamics of a large cloud of ^{87}Rb atoms ($N \sim 10^7 - 10^9$) within a Magneto-Optical Trap. To overcome the computational bottleneck of $O(N^2)$ interactions, the simulation employs a **Superparticle** approach combined with advanced mesh-based algorithms: a **Vectorized Sorting method** for beam attenuation and a **Particle-Mesh (FFT) method** for radiative rescattering forces. This allows for the study of density limits, instabilities, and the transition from the temperature-limited regime to the multiple-scattering regime.

Contents

1 Physics and Purpose

1.1 Purpose of the Simulation

Magneto-Optical Traps (MOTs) operate in two distinct regimes. At low atom numbers, the cloud size is determined by the balance between the trapping force and thermal diffusion (Temperature Limited). As the number of atoms increases, collective effects—specifically **Shadowing (Attenuation)** and **Rescattering**—become dominant.

- **Shadowing (Compressive):** Outer atoms absorb laser light, casting shadows on inner atoms. This creates an imbalance that pushes atoms inward.
- **Rescattering (Repulsive):** Atoms re-emit absorbed photons. Nearby atoms absorb these scattered photons, resulting in a repulsive, Coulomb-like force that expands the cloud.

This simulation is designed to test the **Density Limit** (where repulsion balances compression) and verify thermodynamic stability (Virial Theorem) in these high-density regimes.

1.2 Physical Model

1.2.1 The Superparticle Approximation

Simulating 10^9 real atoms is computationally infeasible. We use the **Superparticle** concept:

- We simulate $N_{sim} \sim 10^4$ superparticles.
- Each superparticle represents N_{super} (e.g., 7000) real atoms.
- **Dynamics:** They move according to the mass and cross-section of a single atom.
- **Collective Effects:** They radiate and absorb light with the power of N_{super} atoms.

1.2.2 Trapping and Diffusion Forces

The code implements a 3D Doppler cooling model on the $F = 0 \to F' = 1$ transition (approximate).

$$\mathbf{F}_{trap} = \sum_{\text{beams } \alpha} \frac{\hbar k \Gamma}{2} \frac{s_\alpha}{1 + s_{tot} + 4(\delta_\alpha/\Gamma)^2} \hat{k}_\alpha \tag{1}$$

The detuning δ_α includes the laser detuning δ, the Doppler shift $\vec{k}\cdot\vec{v}$, and the position-dependent Zeeman shift $\mu B(r)$.

Heating due to spontaneous emission is modeled via a Langevin force (Diffusion):

$$\mathbf{F}_{diff} = \sqrt{\frac{2D_{tot}}{\Delta t}} \cdot \mathcal{N}(0,1) \tag{2}$$

where D_{tot} is the momentum diffusion coefficient derived from the total saturation parameter.

2 Algorithms and Complexity

The simulation uses three distinct algorithmic strategies to handle forces efficiently.

2.1 1. Time Integration: Velocity Verlet

Complexity: $O(N)$
The equations of motion are solved using the Velocity Verlet algorithm. This is a symplectic integrator, offering better energy conservation and stability than Euler methods for conservative-like systems.

$$\vec{r}(t+\Delta t) = \vec{r}(t) + \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2 \tag{3}$$

$$\vec{v}(t+\Delta t) = \vec{v}(t) + \frac{1}{2}\left(\vec{a}(t) + \vec{a}(t+\Delta t)\right)\Delta t \tag{4}$$

2.2 2. Beam Attenuation: Vectorized Sorting (Grid/Tube)

Complexity: $O(N \log N)$
Calculating the shadow cast by every atom on every other atom is naively $O(N^2)$. We reduce this by approximating the cloud as a bundle of parallel 1D tubes aligned with the laser axes.

Algorithm:

1. **Binning:** Atoms are assigned to transverse spatial bins (u, v) perpendicular to the beam axis (w).
2. **Sorting:** Within each bin (tube), atoms are sorted by their longitudinal position along the beam.
3. **Cumulative Sum:** We calculate the cumulative optical depth (OD) along the sorted tube.
4. **Beer's Law:** Intensity is reduced as $I(z) = I_0 e^{-\text{OD}(z)}$.

The bottleneck is the sorting step, making the complexity $O(N \log N)$. This is orders of magnitude faster than $O(N^2)$ for $N > 10^3$.

2.3 3. Rescattering: Particle-Mesh (FFT)

Complexity: $O(N + M \log M)$
The rescattering force follows an inverse-square law ($1/r^2$), mathematically identical to the Coulomb force in electrostatics.

$$\mathbf{F}_{resc} \propto \sum_{j \neq i} \frac{P_j(\mathbf{r}_i - \mathbf{r}_j)}{|\mathbf{r}_i - \mathbf{r}_j|^3} \implies \nabla \cdot \mathbf{F}_{resc} \propto \rho_{\text{light}} \tag{5}$$

Instead of summing pairs ($O(N^2)$), we solve the corresponding Poisson equation using the Particle-Mesh method.

Algorithm:

1. **Charge Assignment (Binning):** The scattered power P of each atom is deposited onto a 3D grid (M cells) to create a density field $\rho(\vec{r})$.
2. **FFT Poisson Solver:**
   - Compute $\hat{\rho}(\mathbf{k}) = \text{FFT}(\rho)$.
   - Multiply by the Green's function in Fourier space: $\hat{\Phi}(\mathbf{k}) = \hat{\rho}(\mathbf{k})/k^2$.
   - Compute Field gradients: $\hat{E}(\mathbf{k}) = -i\mathbf{k}\hat{\Phi}(\mathbf{k})$.
   - Compute $\mathbf{E}(\mathbf{r}) = \text{IFFT}(\hat{E})$.

3. **Interpolation:** The force field on the grid is interpolated back to the specific atom positions.

This decouples the complexity from the number of atoms N. It depends only on the grid size M. For 10^4 atoms, this is effectively instantaneous compared to the brute-force approach.

3 Usage Guide

3.1 Running the Simulation

The script is executed from the command line. It requires 4 arguments.

Syntax:

```
python mot_simulation.py [N_atoms] [B_gradient] [T_init_uK] [R_init_m]
```

Example: To simulate 10,000 atoms, with a gradient of 0.1 T/m, starting at 1000 μK and a radius of 0.5 mm:

```
python mot_simulation.py 10000 0.1 1000 5e-4
```

3.2 Output Structure

The simulation creates a directory named according to the parameters: `Results/res_N=1.0e+04_B=1.0e-01.../`

- **parameters.txt**: A human-readable log of all constants (Physics, Laser, Simulation settings) used.
- **mot_data_full.npz**: Compressed NumPy archive containing:
  - `time`: Array of time stamps.
  - `positions`: Shape $(N_{frames}, N_{atoms}, 3)$.
  - `velocities`: Shape $(N_{frames}, N_{atoms}, 3)$.

3.3 Post-Processing and Visualization

The simulation suite includes a dedicated post-processing script, `mot_postprocessing.py`, which automates data analysis and visualization. This script loads the simulation history, computes derived quantities (RMS radius, Temperature, Kinetic Energy), and generates diagnostic plots and animations.

3.3.1 Execution

The script is executed with the same parameters used for the simulation to locate the correct results folder. An optional `download` flag can be used to fetch data from a remote cluster via SCP before analysis.

Syntax:

```
python mot_postprocessing.py [N_atoms] [B_gradient] [T_init_uK] [R_init_m] [optional: download
```

Example:

```
python mot_postprocessing.py 10000 0.1 1000 5e-4
```

3.3.2 Generated Outputs

The script generates the following files in the results directory:

- **figure1_scalar_history.png:** Time evolution of RMS Radius (Confinement), RMS Velocity (Cooling), and Total Kinetic Energy.
- **figure2_spatial_confinement.png:** Final 2D spatial distribution (xz-plane), dimensional RMS radii bar chart, and 1D density profiles.
- **figure3_phase_space.png:** Phase space scatter plot (x vs v_x) and velocity distribution histograms.
- **mot_evolution.gif:** An animated GIF showing the time evolution of the cloud in the xz and yz planes, including real-time Temperature display.
- **Virial.png:** A stability check comparing Kinetic Energy T against the Virial of the Trapping Force W. Stability is indicated by a ratio $-W/2T \approx 1$.