Project Overview

Goal: analyzing macroscopic properties of atoms in a system using MD trajectories

Task 1: Computing the fluctuations

Sought: replace the ensemble average by a time average

Based on: Ergodic theory

$$\langle x \rangle = \bar{x} = \frac{1}{N_x} \sum_{i=1}^{N_x} x_i$$

Implementation: compute the averages and the ***fluctuations*** during each time step while running the MD program

Algorithm 1.1: Inaccurate computation [formula 42]

$$\sigma_x^2 = \frac{1}{N_x - 1} \left(\sum_{i=1}^{N_x} x_i^2 - \frac{1}{N_x} \left(\sum_{i=1}^{N_x} x_i \right)^2 \right)$$

Algorithm 1.2: accurate computation [formula 43, 45]

$$\sigma_x^2 = \frac{Q_{N_x}}{N_x - 1} \qquad \text{For } i = 2, 3, \ldots N_x :$$

$$Q_i = Q_{i-1} + \frac{(S_{i-1} - (i-1)x_i)^2}{i(i-1)}$$

$$S_i = S_{i-1} + x_i$$

Task 2: Computation of correlation functions

Sought: discretization of the correlation function

Based on: Convolution theorem, linearity of FFT

Implementation: obtain the correlations calculated at each time frame by post-processing the wanted property (i.e. velocity)

Algorithm 2.1: Direct method [formula 49]

$$C(n\Delta t) = (N_{\mathrm{MD}} - n)^{-1} \sum_{k=0}^{N_{\mathrm{MD}}-n-1} x(k\Delta t) x((k+n)\Delta t)$$

Algorithm 2.2: FFT method with zero padding [formula 53, 54]

$$C(n\Delta t) = \frac{1}{2N_{\mathrm{MD}}(N_{\mathrm{MD}} - n)} \sum_{m=0}^{2N_{\mathrm{MD}}-1} \hat{x}(m\Delta\omega)^* \hat{x}(m\Delta\omega) e^{-im\Delta\omega n\Delta t}$$

$$\Delta\omega = \frac{2\pi}{2N_{\mathrm{MD}}\Delta t}$$

Implementation

Task 1: Computing the fluctuations

```cpp
void Calculator::inaccurate_fluctuation(double x_i, int m){
    S(m) += x_i;
    Ssq(m) += std::pow(x_i, 2);
}

void Calculator::accurate_fluctuation(double x_i, int m){

    Q(m) += (S(m)-(index-1)*x_i) * (S(m)-(index-1)*x_i) /(index*(index-1));

    S(m) += x_i;
}


// getter
double Calculator::getFluctuation_inaccurate(int m) const{
    double var = (Ssq(m)-S(m)*S(m)/numMDSteps)/(numMDSteps-1);
    return std::sqrt(var);
}

double Calculator::getFluctuation_accurate(int m) const{
    double var = Q(m)/(numMDSteps-1);
    return std::sqrt(var);
}
```

```cpp
void MDRun::run(std::vector<double> &x, std::vector<double> &v) {
    forces.resize(x.size());
    synchronizedPositions.resize(x.size());
    radialDistribution.setZero();

    initializeVariables();
    initializeTemperature(v);

    output.printInitialTemperature(properties[1] / fac);
    output.printIterationStart();

    /* dynamics step */
    Calculator calculator(par.numberMDSteps, numberProperties);
    calculator.set_mode(false); // is_accurate = true

    double time = par.initialTime;
    for (int nstep = 0; nstep < par.numberMDSteps; nstep++) {
        performStep(x, v, nstep, time, calculator);
        time += par.timeStep;
    }

    printAverages(time, calculator);
}
```

Task 2: Computation of correlation functions

```cpp
// formula 49
void CorrelationCalculator::computeCorrelation_direct(){

    double S;

    // looping through all steps
    for(int n = 0; n < numMDSteps; ++n){
        S = 0.;

        for(int k = 0; k < numMDSteps - n; ++k){
            S += (Mat.col(k)).dot(Mat.col(k + n));
        }

        C_direct(n) = S / ( numMDSteps - n);
    }

    C_direct /= C_direct(0);   // normalization
}
```

$$\text{Mat} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

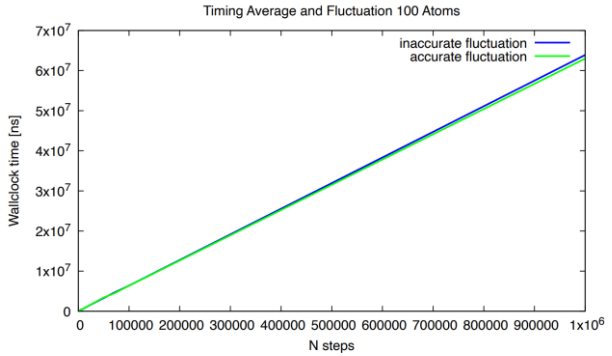
Data structure used for storing velocities from the MDprogram

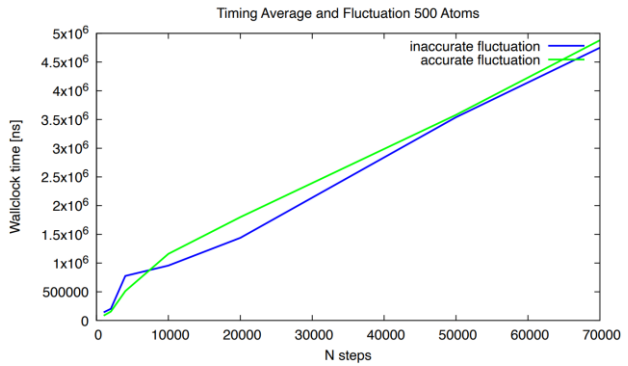
```cpp
// formula 53
Eigen::VectorXd CorrelationCalculator::getC_i(const Eigen::VectorXd& v) {

    int size = v.size();

    // zero padding
    Eigen::VectorXcd vec_padded;
    vec_padded.setZero(size * 2);
    vec_padded.head(size) = v;

    // FFT
    Eigen::FFT<double> fft;
    auto temp = fft.fwd(vec_padded);
    // squared modulus
    auto transformed = temp.cwiseProduct(temp.conjugate()).eval();

    // IFFT
    return fft.inv(transformed).real().head(size);
}


void CorrelationCalculator::computeCorrelation_FFT(){

    // looping through all atoms
    for(int k = 0; k < numberAtoms; ++k){
        C_FFT += getC_i(Mat.row(k).transpose());
    }
        C_FFT = C_FFT.cwiseQuotient(divisor);
        C_FFT /= C_FFT(0);   // normalization
}
```

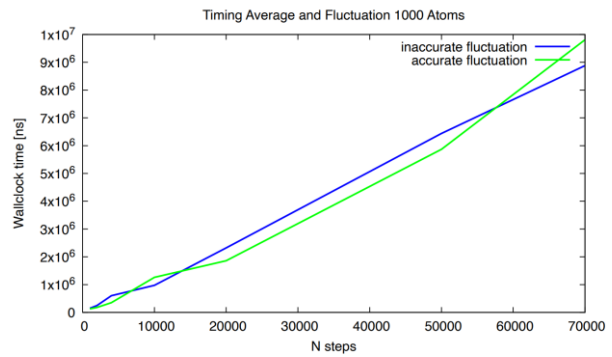
GitHub

Results & Conclusion

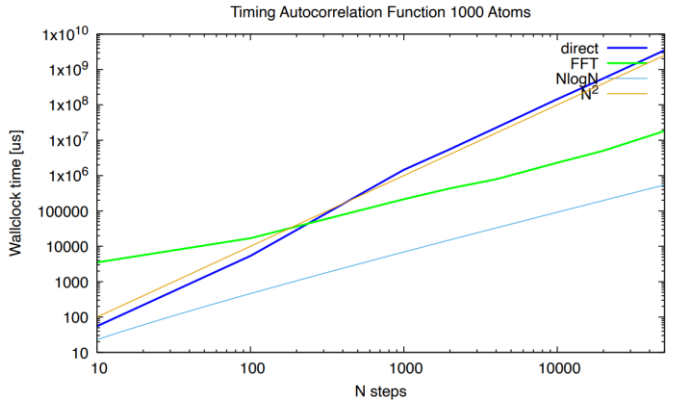
Performance of fluctuation computation:







Performance of VAC:



Results of VAC: