Calculation of Velocity Auto-Correlation Function Using Parallel Algorithms

Parth Shah, Shubhadeep Nag and Subramanian Yashonath

(Dated: 15 January 2020)

I. INTRODUCTION

To study the detailed nature of a system of particles, it is important to understand the correlation of their properties. There are several types of correlation present in a system , like (a) *Mean Square Displacement, it is the temporal correlation of the positions of the particles with the evolution in time, (b) Velocity Auto-correlation function is also a temporal correlation which shows the correlation in the velocities of the particles, whereas (c) Radial distribution function is the spatial correlation which gives the idea of the phase of the system and is analogous to x-ray spectrum of the system of particles. Other than statistical mechanics, correlation is a very important quantity to study in various branches of science, e.g. quantum field theory, astronomy etc. In definition, a correlation function is a function that measures the statistical correlation between two random variables. These correlations can be on the both spatial or temporal distance between those variables.*

The Velocity Auto-Correlation Function (VACF) is a time dependant correlation function. It is used to calculate properties of a system such as the coefficient of diffusion[1]*. Systems such as liquid Argon*[2]*, two component ionic liquids*[3]*, symmetric molten salts and sodium chloride*[3] *are studied using Molecular Dynamics (MD) Simulations. Interesting results are inferred from the nature of the VACF function in each of the preceding papers. VACF is defined as*

$$C(t) = \Big\langle v_i(\tau) \cdot v_i(\tau + t) \Big\rangle_{\tau,N} \tag{1}$$

where v_i represents the velocity of the i^{th} particle and $C(t)$ is the correlation of the velocities at time t and $t + \tau$.
The computational complexity is

$$\begin{aligned} &\sim O(N \cdot \prod_{i=1}^{C} (T - i)) \\ &\sim O(N \cdot T^{C}) \end{aligned} \tag{2}$$

where C is the number of correlations, T is the total number of trajectory points stored and N is the number of particles. VACF is used to compute the Co-efficient of Diffusion. The data required to calculate the VACF is the velocities of all particles for the simulation length. This *can be extracted from a dump file obtained from MD Simulation Packages like LAMMPS*[4] *or DL_POLY*[5].

II. METHODS

We write this section with reference to the computation of VACF but recognize that these algorithms can be adopted to solve other correlation problems. The parallel calculation of VACF for a system is straightforward. The data is represented by two dimensional vectors with the particles spanning the rows and the time-steps spanning the columns. This representation was chosen after inspecting data access patterns and data distribution strategies. Three algorithms have been developed which are detailed below.
Continuous attempts to increase efficiency were pursued. Writing or reading object files instead of ASCII files did not show any significant improvement. All algorithms are for a shared memory model and use MPICH, an implementation of the Message Passing Interface (MPI) and are of a Single Instruction Multiple Data (SIMD) type. These algorithms have been tested against a serial implementation for accuracy and bench-marking purposes.

1. Correlation Decomposition

This algorithm shall be referred to as algorithm 1. The complexity is

$$\sim O(N \cdot T^{\frac{C}{p}})$$

where p is the number of processors. Algorithm 1 divides the correlations amongst all the processors. Each processor has data for the entire simulation length and all particles. Each processor calculates the assigned correlations. We recognized that the initial correlations take longer to compute than the later correlations due to the nature of the function and hence load balanced.

a. ***Massively Parallel Simulation Length Decomposition***

Another algorithm was developed to suit large simulation lengths of the order of 10^6 *femtoseconds, where the velocities of all particles are stored at a frequency of 10 femtoseconds.*

This will be referred to as Algorithm 1.1. Algorithm 1.1 divides the simulation length amongst the processors according to the amount of free memory. Each processor then reads their respective time-steps and an additional number of time-steps that is equal to the number of correlations. Each processor proceeds to calculate all correlations for every particle but for their subset of time-steps locally. The results are then transferred to the root processor where they are added. This is repeated for the entire simulation length. We have observed that changing the number of time-steps per processor does not impact performance. Algorithm 1.1 was not pursued further as initial bench-marking against Algorithm 1.2 (detailed later) showed non-satisfactory results. It is almost 5 times slower for large number of correlations and large simulation lengths, 10^5 *and* 10^5 *respectively. However, for smaller numbers, it out performs algorithm 2.1 if the time taken to initialize is taken into account.*

2. Particle Decomposition

This algorithm shall be referred to as algorithm 2. The complexity is

$$\sim O(\frac{N}{p} \cdot T^C)$$

where p is the number of processors. Algorithm 2 divides the particles amongst all the processors. Each processor has data for the entire simulation length but a subset of particles. Each processor proceeds to calculate for all correlations but for their subset of particles locally. The results are then transferred to the root processor where they are added.

a. Massively Parallel Particle Decomposition

Another algorithm was developed to suit large simulation lengths of the order of 10^6 *and will be referred to as algorithm 2.1. Algorithm 2.1 initially divides the large input file into multiple files equal to the number of particles. Each file has data corresponding to that particle for the entire simulation length. Particles are now distributed amongst the processors according to the amount of free memory. Each processor proceeds to calculate all correlations but for their subset of particles locally. The results are then transferred to the root processor where they are added. This is repeated till all particles are completed. We have observed that changing the number of particles per processor or the distribution of particles does not* *impact performance. Algorithm 2.1 has a large initialization time ie.* 10^4 *seconds. However, for large number of correlations and large simulation lengths,* 10^5 *and* 10^5 *respectively, this algorithm is optimal.*

3. Double Decomposition

This algorithm shall be referred to as algorithm 3. The complexity is

$$\sim O(\frac{N}{b} \cdot T^{\frac{Cb}{p}})$$

where b is the number of processors per batch, p is the number of processors. Algorithm 3 divides both, the correlations and the particles amongst all processors. Processors are first divided into batches. Each batch is assigned a subset of correlations. Each processor in each batch is assigned a subset of particles. Each processor has data for the entire simulation length and a subset of particles. Each processor then proceeds to calculate for the assigned correlations and particles. The results are then transferred to the root processor of the batch where they are added.

III. RESULTS

We carried out an MD simulation of liquid Argon particles to test the correctness of our algorithms. LAMMPS was used to perform the simulations. A system of 864 particles at 94.4K and with number density $1.374g/cm^3$ *was simulated. The simulation parameters were taken from A. Rahman's seminal paper*[6]. *We computed the VACF (FIG. 1) from the stored velocities of all the particles after the simulation. We also calculated the coefficient of diffusion (D)*[2], *given by*

$$D = \frac{1}{3} \int_0^\infty \Big\langle v(0) \cdot v(\tau) \Big\rangle d\tau \tag{3}$$

The diffusivity calculated from our serial algorithm was $2.05 \times 10^{-5} cm^2/s$ *whereas, a value of* $2.43 \times 10^{-5} cm^2/s$ *was reported by A. Rahman. All runs were performed on an Intel Xeon E5 cluster with 24 processors per node, each operating at 2.30 GHz.*

All runs were performed with 24, 48 and 72 processors to observe scalability.

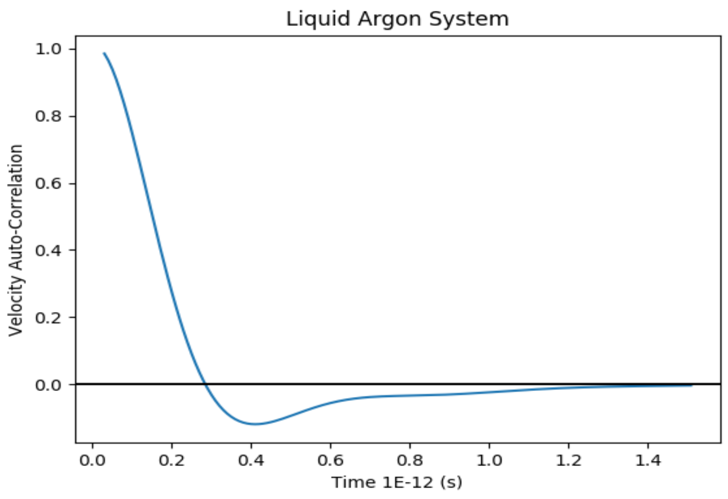

FIG. 1: Velocity Auto-Correlation of liquid Argon, 864 Particles at 94.4K and with number density $1.374 g/cm^3$.

Algorithm 3 maintains an overall low execution time. In addition, it demonstrates parallel speedup for larger number of correlations. 100K correlations are not calculated for a simulation length of 5E4 as the correlations are greater than the total simulation length.

TABLE I: Execution Time in seconds for 500 Particles.

Correlations	No. of Processors	Algorithm 1		Algorithm 2		Algorithm 3	
		T=5E4	T=2E5	T=5E4	T=2E5	T=5E4	T=2E5
1K	24	101	398	62	247	52	201
	48	57	219	55	218	44	163
	72	118	459	88	349	40	162
5K	24	272	1077	175	741	129	534
	48	155	631	137	585	82	331
	72	173	714	297	1235	66	265
10K	24	478	2035	307	1369	223	938
	48	262	1121	229	1033	129	534
	72	188	787	534	2325	98	400
100K	24	-	16055	-	10358	-	7598
	48	-	7774	-	7579	-	4084
	72	-	5279	-	18373	-	2798

As the number of correlations increase, as expected, Algorithm 1 scales efficiently.

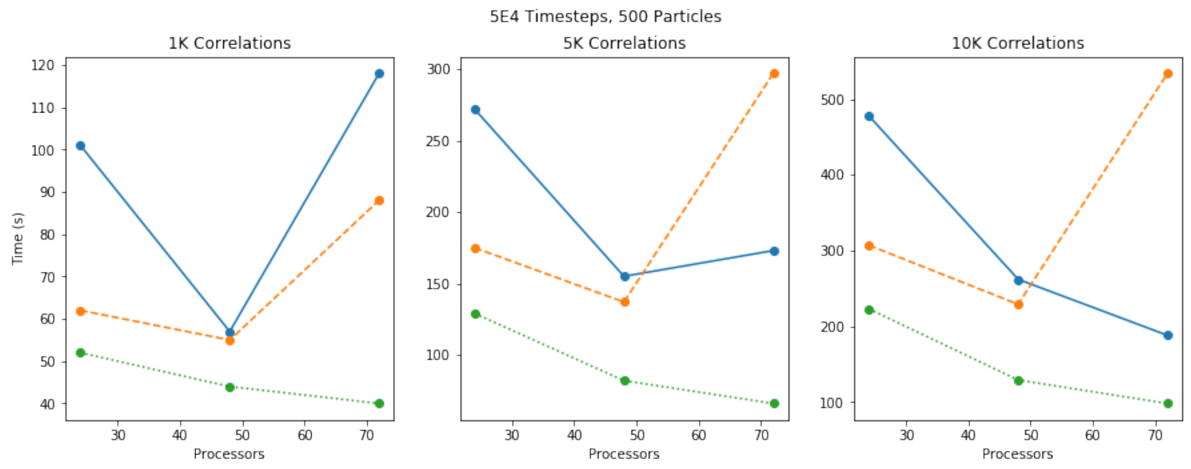

FIG. 2: Execution Time for Algorithms 1 (blue), 2 (orange), 3 (green) for Simulations of 5E4 Timesteps, 500 Particles

The same can be observed here. Algorithm 3 continues to performs optimally. Overall, the execution time taken by Algorithm 3 is the lowest.

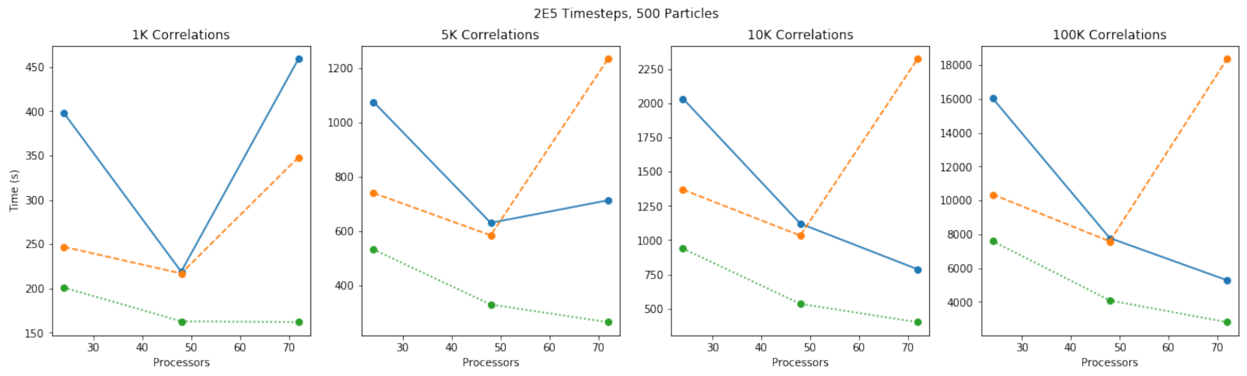

FIG. 3: Execution Time for Algorithms 1 (blue), 2 (orange), 3 (green) for Simulations of 2E5 Timesteps, 500 Particles

Algorithm 2.1 demonstrates massive parallel speedup. Although, it requires a significant time for setup (not included below) equal to 11786 seconds. For 100K correlations, on 24 and 48 processors, the execution time was extrapolated on account of the same being large.

For larger simulation runs, more particles, Algorithm 2.1 performs optimally. The scaling is almost linear.

TABLE II: Execution Time in seconds for 6912 Particles.

Correlations	No. of Processors	Algorithm 2.1	
		T=2E5	T=1E6
1K	24	1308	6500
	48	710	3269
	72	388	1702
5K	24	6167	32053
	48	3136	16019
	72	1589	7828
10K	24	12069	63784
	48	6095	31881
	72	3068	15394
100K	24	91732	(599847)
	48	46067	(299799)
	72	22968	144743

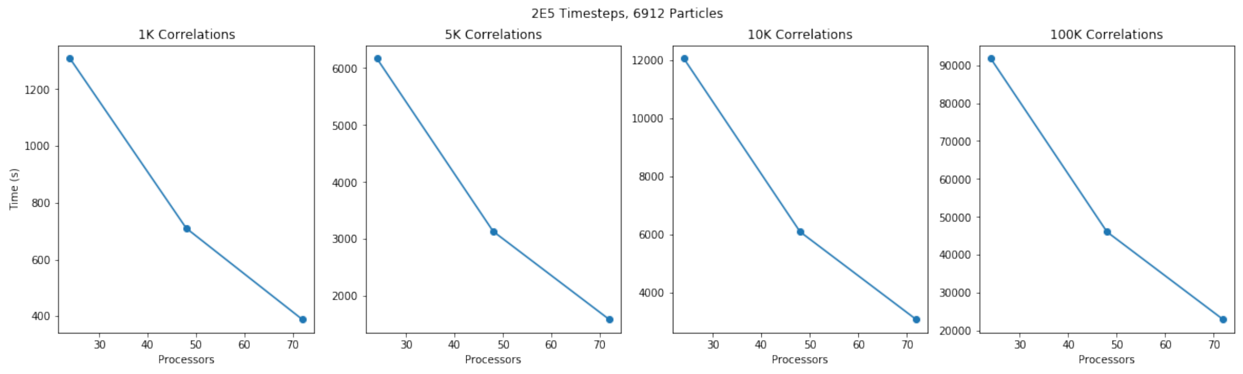

FIG. 4: Execution Time for Algorithm 2.1 (blue) for Simulations of 2E5 Timesteps, 6912 Particles

The execution time for 100K correlations on 24 and 48 processors was approximated from the preceding data.

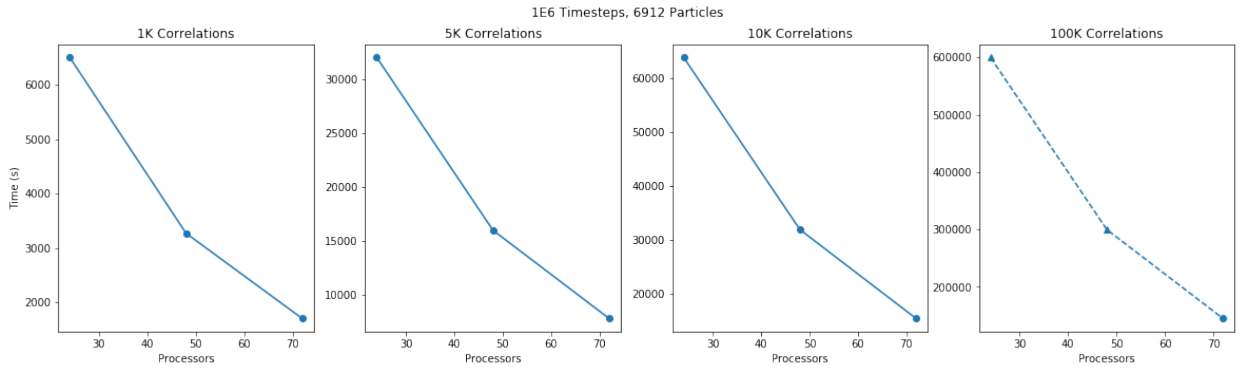

FIG. 5: Execution Time for Algorithm 2.1 (blue) for Simulations of 1E6 Timesteps, 6912 Particles

IV. CONCLUSION

The nature of the VACF function makes it highly parallelizable. A combination of correlation and particle decomposition scales most efficiently. Calculations for larger simulation lengths, particles scales efficiently too.

V. APPENDIX

1. Prediction of run time

The amount of time the optimal parallel algorithm would take to calculate the VACF, given a system is predicted from the collected data. A simple Artificial Neural Network (ANN) is used to achieve this. ANN learning methods provide a robust approach to approximating real-value, discrete-valued and vector-valued target functions. They are amongst the most effective learning methods currently known. However, the ability of humans to understand the learned target function is lost.
The ANN is based on a unit called a perceptron. It takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise. A network of many interconnected units learn an unknown function. A method of learning an acceptable weight vector is to begin with random weights, then iteratively apply the network to each training example, modifying the weights whenever it misclassifies an example. Multi-layer networks learned by the back-propagation algorithm are capable of expressing a rich variety of non-linear decision surfaces. The back-propagation

algorithm learns the weights for a multi-layer network, given a network with a fixed set of units and interconnections. It employs gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs. Over-fitting may occur because the weights are being tuned to fit idiosyncrasies of the training examples that are not representative of the general distribution of examples.[7]

The structure of the ANN used to predict the run times is as follows. The ANN has 3 hidden layers, an input layer with 3 inputs, an output layer with 1 output. The inputs are, the number of time-steps, the number of particles, the number of processors. The output is the predicted run time. The hidden layers have 64 nodes each. The training set has 24 instances. The activation function in all the layers is Rectified Linear except the last layer which uses a Linear function as the output values are continuous. The loss is calculated using absolute error and the optimizer used is Adam[8].

REFERENCES

[1] *J.P. Boon and S. Yip.* Molecular Hydrodynamics. *Dover books on physics. Dover Publications, 1991.*

[2] *Donald E. O'Reilly. Velocity autocorrelation function and selfdiffusion in liquids.* The Journal of Chemical Physics, *55(6):2876–2884, 1971.*

[3] *Toyonori Munakata. Velocity field and velocity autocorrelation function in ionic liquids.* The Journal of Chemical Physics, *78(12):7290–7295, 1983.*

[4] *Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics.* Journal of Computational Physics, *117(1):1 – 19, 1995.*

[5] *W. Smith, C.W. Yong, and P.M. Rodger. Dl_poly: Application to molecular simulation.* Molecular Simulation, *28(5):385–471, 2002.*

[6] *A. Rahman. Correlations in the motion of atoms in liquid argon.* Phys. Rev., *136:A405–A411, 1964.*

[7] *Tom M. Mitchell.* Machine Learning. *McGraw-Hill, 1997.*

[8] *Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.*