Queen Mary
University of London

Calculating system properties on the fly in DL_POLY

CoSeC Conference 2023
Manchester Central

HL. Devereux[1], AM. Elena[2], IT Todorov[2], K. Trachenko[1]

[1] School of Physics and Astronomy, Queen Mary University of London

[2] Daresbury Laboratory STFC UKRI, Scientific Computing Department

- DL_POLY – a brief overview
- On-the-fly correlations
  - The big picture
  - Theory
  - Software design
  - User workflow
- Case studies
- Conclusions

DL_POLY – a brief overview

- Classical molecular dynamics (MD) software
- Developed at the Daresbury Lab continuously since 1994 [1] designed for large scale CPU parallelism [2]
- Currently open-source at Gitlab - https://gitlab.com/ccp5/dl-poly
- Fortran (90, with some OOP via 2003+)
- Python meta package, dlpoly-py https://pypi.org/project/dlpoly-py/ also open source on Gitlab

Code: DL_POLY

dlpoly-py code

https://gitlab.com/ccp5/dl-poly

https://gitlab.com/drFaustroll/dlpoly-py

[1] Todorov, I.T. and Smith, W., 2004. DL_POLY_3: the CCP5 national UK code for molecular–dynamics simulations. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, *362*(1822), pp.1835-1852.
[2] Todorov, I.T., Smith, W., Trachenko, K. and Dove, M.T., 2006. DL_POLY_3: new dimensions in molecular dynamics simulations via massive parallelism. *Journal of Materials Chemistry*, *16*(20), pp.1911-1918.

DL_POLY – a brief overview

Current developments include:

- On-the-fly correlation functions (this talk!)
- Machine learned potentials (Via OpenKIM)
- GPU acceleration
- Data provenance, and reproducible science
- Atomic Simulation Environment (ASE) integration
- dlpoly-py, Python meta-package
- Automatically detecting regions of interest

Code: DL_POLY

dlpoly-py code

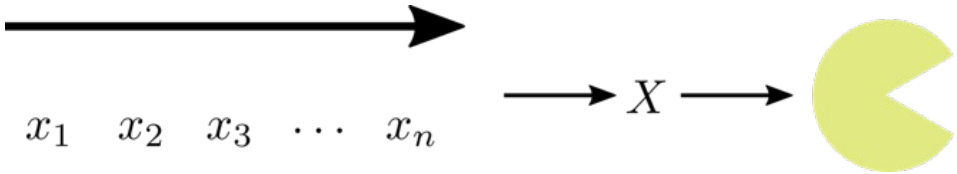
https://gitlab.com/ccp5/dl-poly

https://gitlab.com/drFaustroll/dlpoly-py

Correlations on the fly – the big picture

- Analysis *can* be done by storing a simulation trajectory (atom/molecule configurations over time).

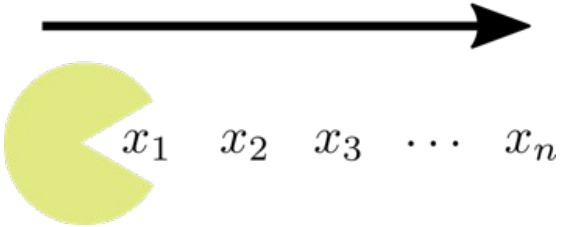
- This creates *infeasible* storage space requirements (and I/O time penalties)



Correlations on the fly – the big picture

- Aim to analyse a simulation *at runtime* resulting in *no trajectory storage.*

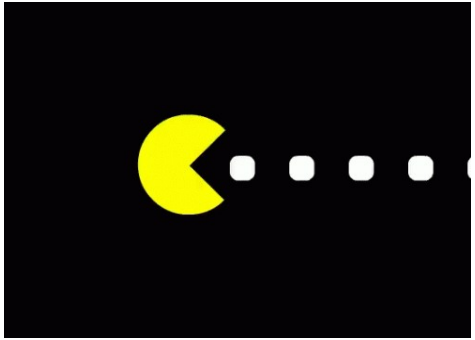
- Previous successes with collision cascades applied to radiation damage [3-4]



[3] Diver, A., Dicks, O., Elena, A.M., Todorov, I.T. and Trachenko, K., 2020. Evolution of amorphous structure under irradiation: zircon case study. *Journal of Physics: Condensed Matter*, *32*(41), p.415703.
[4] Diver, A., Dicks, O., Elena, A. M., Todorov, I. T., Trachenko. K. “Radiation damage effects in amorphous zirconolite.” Journal of Nuclear Materials 544 (2021): 152654

Correlations on the fly – the big picture

- On-the-fly (also online) algorithms consume a sequence of data, updating immediately on each input [5]
- Related to (but not the same as) streaming algorithms applied to manage "Big data streams" [6] and "fast data" [7]



[5] Karp, R.M., 1992, July. On-line algorithms versus off-line algorithms: How much. In *Algorithms, Software, Architecture: Information Processing 92: Proceedings of the IFIP 12th World Computer Congress* (Vol. 1, p. 416).
[6] Krempl, G., Žliobaite, I., Brzeziński, D., Hüllermeier, E., Last, M., Lemaire, V., Noack, T., Shaker, A., Sievi, S., Spiliopoulou, M. and Stefanowski, J., 2014. Open challenges for data stream mining research. *ACM SIGKDD explorations newsletter*, *16*(1), pp.1-10.
[7] Lam, W., Liu, L., Prasad, S.T.S., Rajaraman, A., Vacheri, Z. and Doan, A., 2012. Muppet: Mapreduce-style processing of fast data. *arXiv preprint arXiv:1208.4175*.

Correlations on the fly – Theory

- Online (running) average is a simple, useful, example often used in MD.
- Only needs three numbers, the current average and count (state), and the current data point

$$\mu_n = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\mu_{n+1} = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i$$

$$\Rightarrow \mu_{n+1} = \frac{n\mu_n + x_{n+1}}{n+1}$$

Correlations on the fly – Theory

- Not necessarily offline ⊆ online
- Different computations can have error implications

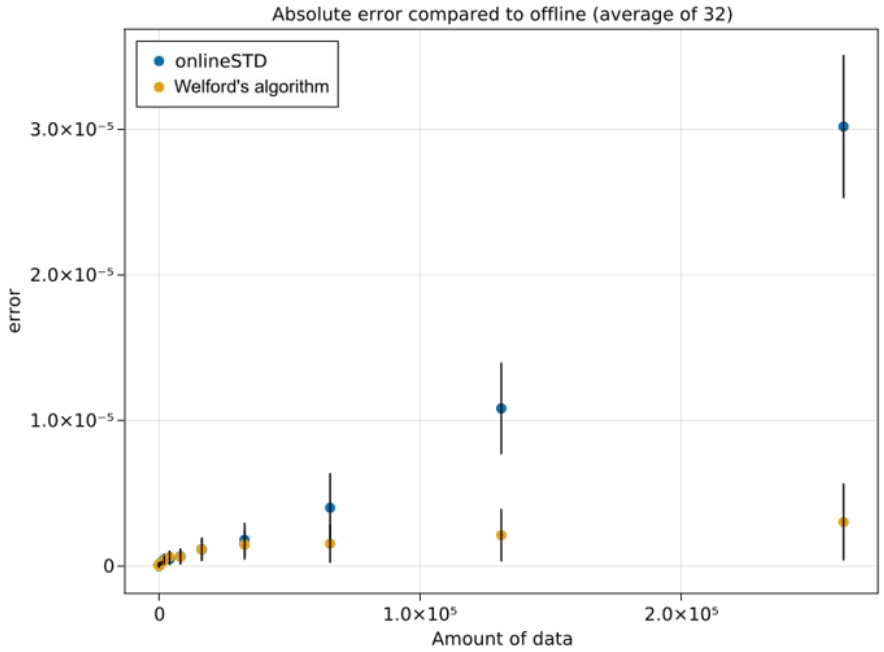
- Online variance

$$s_n^2 = s_{n-1}^2 + \frac{(x_n - \overline{x}_{n-1})^2}{n} - \frac{s_{n-1}^2}{n-1}$$

- Welford's algorithm

$$M_n = M_{n-1} + (x_n - \overline{x}_{n-1})(x_n - \overline{x}_n)$$

$$s_n^2 = \frac{M_n}{n-1}$$



[7] Welford, B.P., 1962. Note on a method for calculating corrected sums of squares and products. *Technometrics*, *4*(3), pp.419-420.

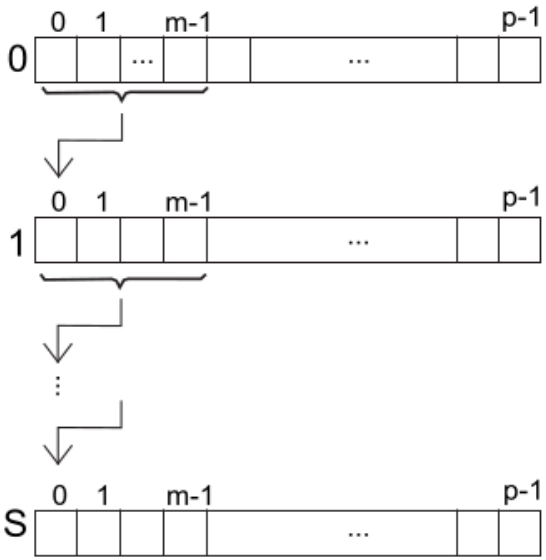
Correlations on the fly – Theory

$$C_{ij}(\mathbf{X}(t), \mathbf{Y}(t+\tau))) = \langle \mathbf{X}_i(t) \cdot \mathbf{Y}_j(t+\tau) \rangle$$

- How to do the same with correlations
- Can we use online means? Or is there a better way...

Correlations on the fly – Theory

- The multi-tau algorithm calculates on-the-fly correlations [7], following earlier online correlators [8-9].
- Stores data in hierarchical block averages.
- Data is passed down blocks, averaged over m points.



Memory outline of multi tau correlator blocks [7]. Each block stores p data points, and passes down data averaged over m points.

[8] Ramírez, J., Sukumaran, S.K., Vorselaars, B. and Likhtman, A.E., 2010. Efficient on the fly calculation of time correlation functions in computer simulations. *The Journal of chemical physics*, *133*(15).

[9] Frenkel, D. and Smit, B., 2002. *Understanding molecular simulation: from algorithms to applications*. San Diego: Academic Press.

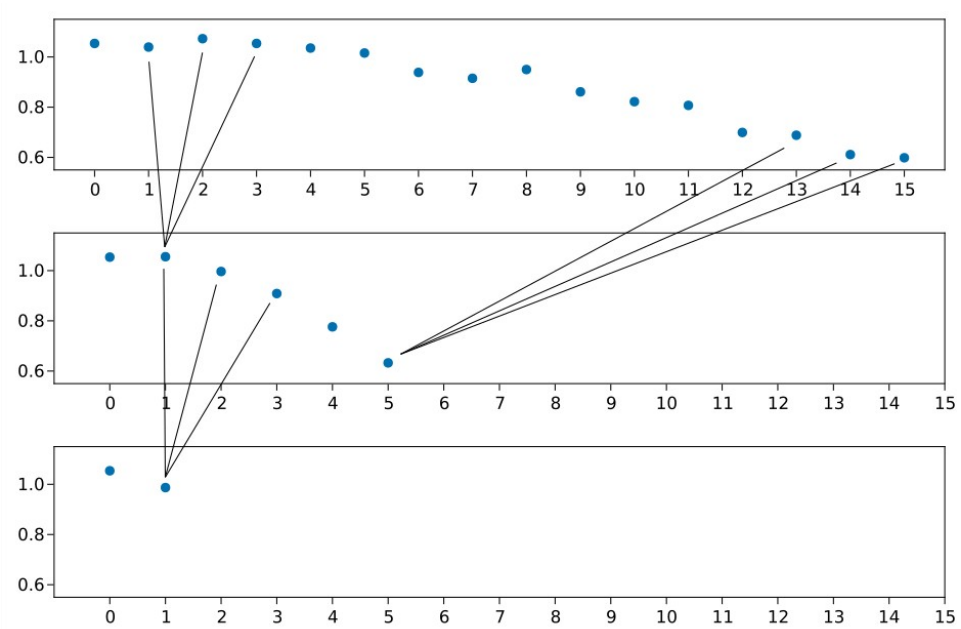
[10] Schätzel, K., Drewel, M. and Stimac, S., 1988. Photon correlation measurements at large lag times: improving statistical accuracy. *Journal of Modern Optics*, *35*(4), pp.711-718.

Correlations on the fly – Theory

Data

Block 1

Block 2



m = 3
p = 16

- Fixed sized each block
- Lag time unwraps as:

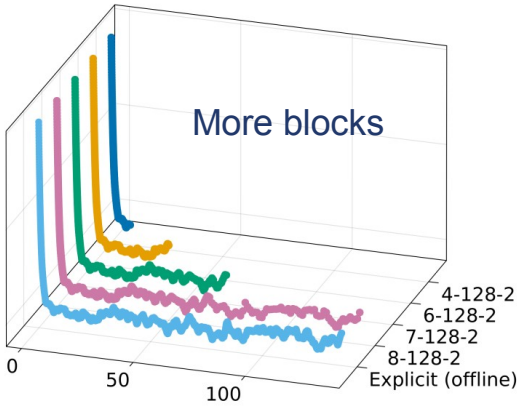
$$t = il^m$$

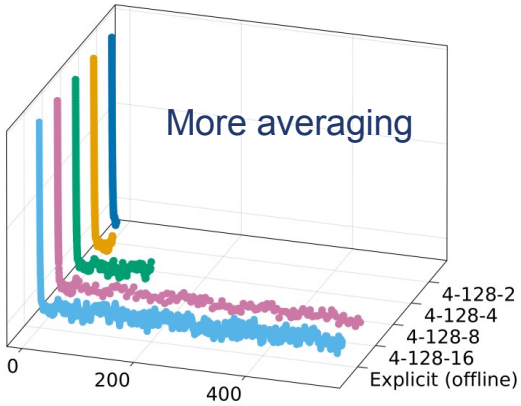
- Correlation updated each step

- As blocks fill, they are "refilled" - sliding window

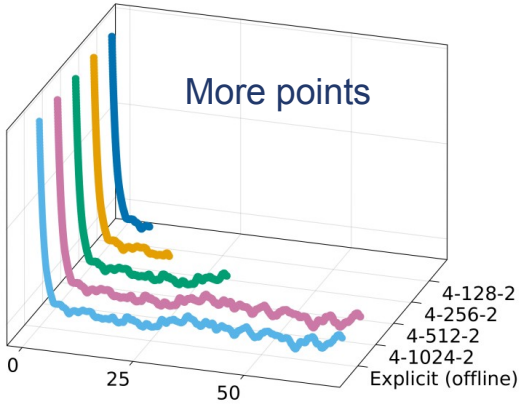
Correlations on the fly – Theory

- Velocity autocorrelation for single Langevin particle
- Blocks – Points (p) - Averaging (m)





$$M\frac{d^2x(t))}{dt^2} + \eta\frac{dx(t)}{dt} = \sqrt{2D/\Delta t}W(t)$$



Correlations on the fly – *Software design*

$$C_{ij}(\mathbf{X}(t), \mathbf{Y}(t+\tau))) = \langle \mathbf{X}_i(t) \cdot \mathbf{Y}_j(t+\tau) \rangle$$

- "Kernelise", observables (X, Y) into a common abstract interface

- A new observable only requires a new kernel definition;
- <u>no tinkering in the main simulation/statistics loop</u>

- Correlators *only correlate* data, the statistics module tracks what the
- data represent

- Special cases: per-atom, spatial, cross-atom correlations

```
for t in 1, t do
  forces(state)
  integrate(state)
  ...

  for cor in correlations do
    cor.update
    (
        cor.a.get(state),
        cor.b.get(state)
    )
  end
end
```

Correlations on the fly – *User workflow*

Input

- Request correlations by juxtaposing observables

- Standardised output – YAML
  - Per species data
  - derived quantities

```
title DL_POLY Argon

correlation_observable [s-s velocity-velocity heatflux-stress]
correlation_blocks [500 100 1000]
temperature 400.0 K
```

```yaml
correlations:
    - name: [stress-stress, global]
      parameters:
            points_per_block: 300
            number_of_blocks: 1
            window_size: 1
      derived:
            viscosity:
                  value:    2.6237850
                  units: Katm ps
            kinematic-viscosity:
                  value:    2.3895092
                  units: Katm ps / (amu / Ang^3)
      lags: [0.0000000, 0.10000000E-02,  ...]
      components:
           stress_xx-stress_xx: [34.869077, 34.886824, ...]
           ...
```

Case studies

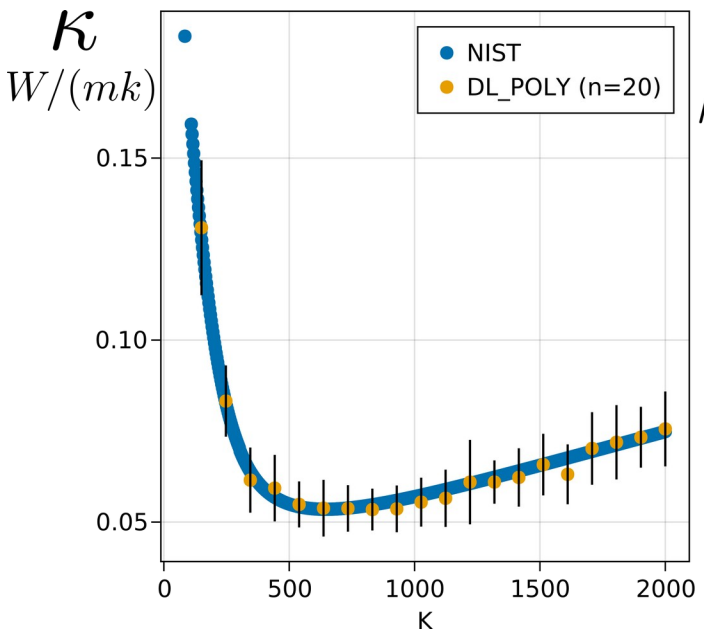
Argon – viscosity and thermal conductivity

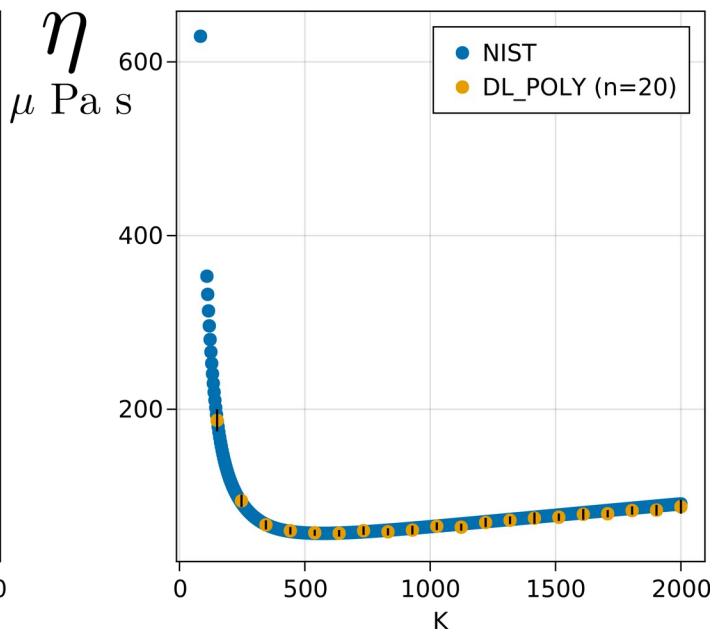
- Compare with NIST experimental data for Argon [9], and previous results using post-processing [10]
- Simple Leonard-Jones model: (eps) 0.01032 eV, (sigma) 3.40 Ang
- N = 16,384

[10] National Institute of Standards and Technology database, see https://webbook.nist.gov/chemistry/fluid.
[11] Cockrell, C., Brazhkin, V.V. and Trachenko, K., 2021. Universal interrelation between dynamics and thermodynamics and a dynamically driven “c” transition in fluids. Physical Review E, 104(3), p.034108.

Case studies

Argon – viscosity and thermal conductivity





Agreement with experimental data. Here for the Argon (N=16,384) system. Similar results with much smaller systems at (N~500).
Error bars, 1 SD of n=20 replicates

Case studies

Argon – workflow

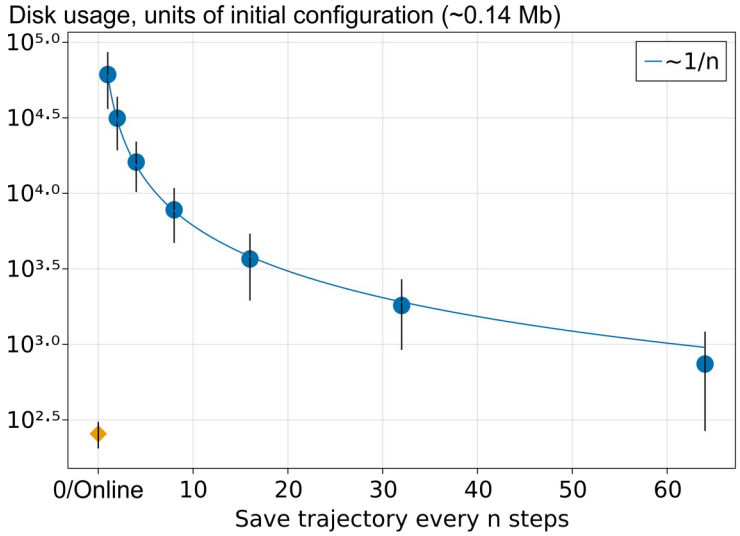
- dlpoly-py provides many convenience features to aid reproducible science
- Separate jobs can be setup and run via simple Python code
- Inputs can be generated and manipulated, Outputs can be queried easily through Python

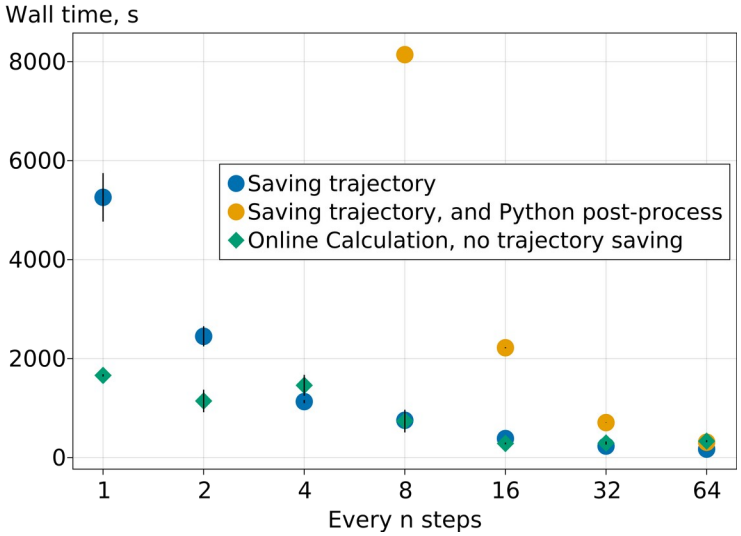
```python
# given temperature T, and replicate r, generate inputs
dlPoly = DLPoly(exe=args.exe,
                control=f"{args.dir}CONTROL-prod",
                config=f"data/T{T}/CONFIG-eq",
                field=f"{args.dir}FIELD",
                workdir="data/T{}-{}-prod".format(T,r),
                output="data/OUTPUT-{}-{}".format(T,r)
               )
    # manipulate CONTROL files cleanly
    dlPoly.control['temperature'] = (T,'K')
    dlPoly.control['random_seed'] = (r,1,round(time()*1000))
    # simulate
    dlPoly.run(numProcs = args.np)
    # obtain correlations
    dlPoly.load_correlations()

    # obtain viscosity and thermal-conductivity measurements!
    for d in dlPoly.correlations.derived:
        if 'viscosity' in d.keys():
            viscosity[r] = d['viscosity']['value']
        elif 'thermal-conductivity' in d.keys():
            thermalCond[r] = d['thermal-conductivity']['value']
```

Snippet from viscosity and thermal-conductivity workflow run from dlpoly-py. Each run a separate DIR is created with generated inputs and containing outputs. Both can be manipulated and queried all through a Python script or Jupyter notebook

Case studies

Argon – velocity auto-correlation





Disk space saving and runtime for an N=500 Argon system. Data for n=0 includes calculation of the velocity auto-correlation function, data for n>0 (blue) does not. Yellow scatter points include VAF calculation time in Python. (from n=4). Disk space averaged over simulation time across three simulations, data is compressed between simulations.

Run on SCARF HPC with 8 MPI processes

Conclusions

- On-the-fly correlation functions successfully implemented in DL_POLY
- Little runtime overhead, and an improvement compared to storing trajectories (VAF)
- Proof of concept, and a platform for other system properties

Acknowledgments, and next Steps
Thanks for Listening!

Suggestions for analysis modes and use cases welcome!

People

- Alin Elena[1,2,3], Jacob-Wilkins[1,2], Cillian Cockrell[3], Illian Todorov[1,3], and Kostya Trachenko[1,3] for getting me up to speed with [1]DL_POLY, [2]dlpoly-py and related [3]science.
- Elliott Kasoar, co-working on dlpoly-py

Silicon (Apocrita@QMUL, Sluis Tier 2, and SCARF@STFC)

- Computing resources provided by STFC Scientific Computing Department's SCARF cluster
- This research utilised Queen Mary's Apocrita HPC facility, supported by QMUL Research-IT. http://doi.org/10.5281/zenodo.438045
- Calculations were performed using the Sulis Tier 2 HPC platform hosted by the Scientific Computing Research Technology Platform at the University of Warwick. Sulis is funded by EPSRC Grant EP/T022108/1 and the HPC Midlands+ consortium.

Coming next...

- Large scale simulations on ARCHER2 (thanks to the Materials Chemistry Consortium, MCC)
- Elastic constants
- More derived quantities (vibrational density of states, dynamical structure factor, ...)

Science and Technology Facilities Council

Engineering and Physical Sciences Research Council

EP/W029006/1

Daresbury Laboratory

CoSeC Computational Science Centre for Research Communities

HEC MCC UK

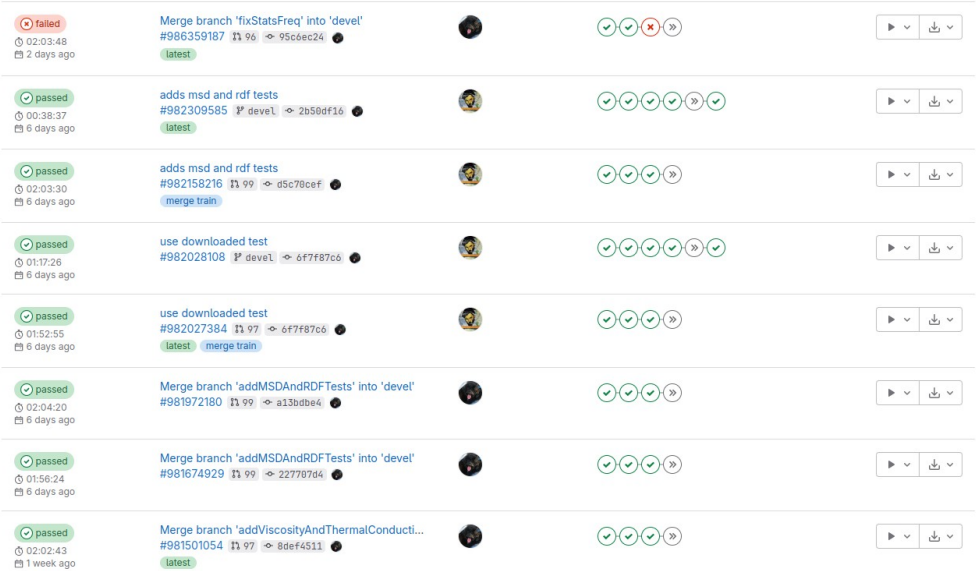
DL_POLY

https://gitlab.com/ccp5/dl-poly

https://gitlab.com/drFaustroll/dlpoly-py

Software engineering

- Updated DL_POLY regression tests with on-the-fly correlation tests
- Including also unit tests in the pipeline
- All via gitlab automated CI, custom runners



Algorithm and implementation detail

The Multi-tau correlator

- Correlator implements multi-tau for n dimensional vectors

- MPI deport/ receive for case of "per-atom" correlations (e.g. velocity)

```fortran
Type, Public :: correlator
                                          ! blocks x points x left_dim x right_dim
  Real(Kind=wp),      Allocatable :: correlation(:,:,:,:)
  Integer(Kind=wi),   Allocatable :: count_correlated(:,:)
                                          ! mutli-tau parameters
  Integer(Kind=wi)                :: number_of_blocks = 0, &
                                     window_size      = 0, &
                                     points_per_block = 0, &
                                     ! correland dimenions
                                     left_dim         = 0, &
                                     right_dim        = 0

  ! various bookkeeping data/ parameters
  Real(Kind=wp),      Allocatable :: left_accumulator(:,:)
  Real(Kind=wp),      Allocatable :: right_accumulator(:,:)
  Integer(Kind=wi),   Allocatable :: count_accumulated(:)
  Real(Kind=wp),      Allocatable :: left_shift(:,:,:)
  Logical,            Allocatable :: shift_not_null(:,:,:)
  Real(Kind=wp),      Allocatable :: right_shift(:,:,:)
  Integer(Kind=wi),   Allocatable :: shift_index(:)
  Integer(Kind=wi)                :: max_block_used = 0, &
                                     min_dist       = 0, &
                                     buffer_size    = 0, &
                                     count_updated  = 0

Contains
  Private

  Procedure, Public               :: init
  Procedure, Public               :: update          ! submit data
  Procedure, Private              :: add             ! recursively update blocks
  Procedure, Public               :: get_correlation ! return correlation
  Procedure, Public               :: deport_buffer   ! (MPI) deport
  Procedure, Public               :: recieve_buffer  ! (MPI) recieve
  Final                           :: cleanup

End Type
```

Algorithm and implementation detail

The Multi-tau correlator

- A correlator's only concern is to receive data (update) and update its correlation (add, recursive), or return the full correlation

- DL_POLY is responsible for making sure a correlator tracks the correct data

- When an atom moves process, receive and deport allow for transferring packed correlator states

```fortran
Type, Public :: correlator
                                          ! blocks x points x left_dim x right_dim
  Real(Kind=wp),      Allocatable :: correlation(:,:,:,:)
  Integer(Kind=wi),   Allocatable :: count_correlated(:,:)
                                          ! mutli-tau parameters
  Integer(Kind=wi)                :: number_of_blocks = 0, &
                                     window_size      = 0, &
                                     points_per_block = 0, &
                                     ! correland dimenions
                                     left_dim         = 0, &
                                     right_dim        = 0

  ! various bookkeeping data/ parameters
  Real(Kind=wp),      Allocatable :: left_accumulator(:,:)
  Real(Kind=wp),      Allocatable :: right_accumulator(:,:)
  Integer(Kind=wi),   Allocatable :: count_accumulated(:)
  Real(Kind=wp),      Allocatable :: left_shift(:,:,:)
  Logical,            Allocatable :: shift_not_null(:,:,:)
  Real(Kind=wp),      Allocatable :: right_shift(:,:,:)
  Integer(Kind=wi),   Allocatable :: shift_index(:)
  Integer(Kind=wi)                :: max_block_used = 0, &
                                     min_dist       = 0, &
                                     buffer_size    = 0, &
                                     count_updated  = 0

Contains
  Private

  Procedure, Public               :: init
  Procedure, Public               :: update          ! submit data
  Procedure, Private              :: add             ! recursively update blocks
  Procedure, Public               :: get_correlation ! return correlation
  Procedure, Public               :: deport_buffer   ! (MPI) deport
  Procedure, Public               :: recieve_buffer  ! (MPI) recieve
  Final                           :: cleanup

End Type
```

Algorithm and implementation detail
Abstract machinery

- An Observable must supply certain Interface functions
- Most functionality via "value"
- Correlation stores a pair of observables, possibly tracking one atom

```
Type, Abstract, Public :: observable
  Contains
    Procedure(get_value),     Deferred :: value
    Procedure(get_dimension), Deferred :: dimension
    Procedure(get_name),      Deferred :: name
    Procedure(get_id)  ,      Deferred :: id
    Procedure(is_per_atom),   Deferred :: per_atom
End Type observable

Type, Public :: correlation
  Class(observable), Allocatable :: A
  Class(observable), Allocatable :: B
  ! 0 indicates not tracking an atom but a global property
  ! >= 1 indicates a local atom's index, or global resp.
  Integer                        :: atom, atom_global
End Type
```

Algorithm and implementation detail
Abstract machinery

- The kernel must take certain state containers and spit out some data (v)

```
Abstract Interface
    ! Kernal for selecting data for correlation
    Subroutine get_value(t, config, stats, v, atom)
        Import observable, configuration_type, stats_type, wp
        Class(observable),          Intent(In    ) :: t
        Type(configuration_type),   Intent(InOut) :: config
        Type(stats_type),           Intent(InOut) :: stats
        Real(Kind=wp), Allocatable, Intent(InOut) :: v(:)
        Integer,       Optional,    Intent(In    ) :: atom
    End Subroutine get_value

    ...

End Interface
```

Algorithm and implementation detail

Abstract machinery

```fortran
Subroutine stress_value(t, config, stats, v, atom)
  Class(observable_stress),      Intent(In    ) :: t
  Type(configuration_type),      Intent(InOut) :: config
  Type(stats_type),              Intent(InOut) :: stats
  Real(Kind=wp), Allocatable,    Intent(InOut) :: v(:)
  Integer, Optional,             Intent(In    ) :: atom

  Integer                                      :: d, i

  Call stress_dimension(t,d)

  Allocate(v(1:d))

  Do i = 1, 9
    v(i) = stats%strtot(i) * prsunt / stats%stpvol
  End Do
```

Kernel for the stress observable, flattening to 9 entries

```fortran
Subroutine velocity_value(t, config, stats, v, atom)
  Class(observable_velocity),    Intent(In    ) :: t
  Type(configuration_type),      Intent(InOut) :: config
  Type(stats_type),              Intent(InOut) :: stats
  Real(Kind=wp), Allocatable,    Intent(InOut) :: v(:)
  Integer, Optional,             Intent(In    ) :: atom

  Integer                                      :: d

  Call velocity_dimension(t,d)

  Allocate(v(1:d))

  If (Present(atom)) Then
    v(1) = config%vxx(atom)
    v(2) = config%vyy(atom)
    v(3) = config%vzz(atom)
  Else
    Call error(0,message="no atom index specified")
  End If
```

Kernel for the atom velocity observable