DL_POLY_4: Multiple Time Stepping Development Support

Ian Bush (Oxford University) and Ilian Todorov (STFC Daresbury Laboratory)

Abstract

The RESPA symplectic multiple time stepping scheme has been implemented in DL_POLY_4. As opposed to an earlier implementation all ensembles and force fields are now covered. Use of the new RESPA implementation is shown to consistently bring a 15-25% improvement in performance at fixed process count, and to also noticeably improve the scaling of the code. The new implementation carefully separates the application dependent and independent layers, so allowing re-use in other molecular dynamics codes.

Introduction

DL_POLY_4[1] is a general purpose classical Molecular Dynamics (MD) simulation software package which has been developed at STFC Daresbury Laboratory[2] by I.T. Todorov. The package is used to model the atomistic evolution of the full spectrum of models commonly employed in the materials science, solid state chemistry, biological simulation and soft condensed-matter communities. It has been developed under the auspices of CCP5[3] and MCC[4] and is widely used throughout the UK as well as world-wide, with over 3500 licence holders. It is a fully data distributed code, employing methodologies such as spatial domain decomposition, link-cells, Verlet neighbour list[5] and the 3D Daresbury Fourier Transform (DaFT)[6]. The code demonstrates excellent performance and has been shown to scale to large numbers of processors.

This proposal was for the implementation of a symplectic multiple time stepping schemes for less frequent calculations of expensive operations such as long-ranged Ewald and short-ranged inter-molecular evaluations. The core of the necessary work had already been partially implemented through a previous dCSE project under the HECToR CSE support mechanism[7]. Though successful it was found during the work that the large number of options and integrators DL_POLY supports made it possible only to implement this scheme for subset of the systems that DL_POLY can simulate, mainly because more restructuring of the integrators was required than initially envisaged. We therefore asked for a small amount of extra time to finish the work for systems that require barostats and rigid bodies, and also a short period of time to merge the new code into the main code base. In this report we highlight the results of this work.

Multiple Time Stepping

Multiple time stepping is a feature especially popular in, although not limited to, the biochemical area of computational chemistry. The idea of symplectic multiple time stepping, RESPA, was first suggested by

[1] http://www.ccp5.ac.uk/DL_POLY/
[2] http://www.sci-techdaresbury.com/properties/daresbury-laboratory/
[3] http://www.ccp5.ac.uk/
[4] http://www.ucl.ac.uk/klmc/mcc/
[5] I.T. Todorov, W. Smith, K. Trachenko & M.T.Dove, J. Mater. Chem., 16, 1911-1918 (2006)
[6] I.J. Bush, I.T. Todorov and W. Smith, Comp. Phys. Commun., 175, 323-329 (2006)
[7] http://www.hector.ac.uk/cse/distributedcse/reports/DL_POLY05/

Tuckerman, Berne and Martyna[8] and elegantly researched by Humphreys, Freisner and Berne[9]. The idea behind the methodology is to exploit the fact that different terms in the force field evolve on different time scales. For instance the long-ranged contribution to the Ewald (electrostatic) terms changes only very slowly when compared to terms involving intra-molecular motion, such as vibrations. Now, as is usual with the numerical integration of differential equations, the fastest motion determines what the timestep must be to ensure numerical stability of the solution (c.f. the Courant-Fredrichs-Lewy[10] condition). However, in standard molecular dynamics every term in the force field is evaluated at every timestep. Thus if a very short timestep is required to correctly integrate molecular vibrations, it will also force the evaluation of the long range Ewald potential at every timestep, despite the later hardly varying at all on the time scale being examined. This can be very expensive, and instead the RESPA method provides a way to evaluate the slowly varying terms in the user specified force field only when required, whilst both keeping the short timestep required for the more quickly varying terms, and only causing a small degradation in the quality of the results. Thus as certain terms are only evaluated infrequently rather than at every step one would hope for an improvement in time to solution.

The mathematics of RESPA ultimately depend on the Trotter expansion of the Liouville propagator. The terms in the MD force field are split into a number of classes and the fastest varying terms (class 1) are evaluated every timestep. Every n_1 timesteps the slower varying Class 2 terms are evaluated, and in turn Class 3 terms are evaluated every n_2 times Class 2 is calculated (and so n_1xn_2 times Class 1 is evaluated). This scheme is iterated until all the classes have been evaluated. In Humpherys *et al.* a 4 level RESPA scheme was outlined, and this was followed in the original implementation within DL_POLY. In that work the following breakdown of the force terms was adopted:

Class 1 (i.e. fastest varying): Core-shell, tethered atoms, harmonic bonds

Class 2: Tersoff, three body, four body, angles, dihedral, inversions

Class 3: Metal, Van der Waals, "Short range" electrostatic terms (including Ewald)

Class 4: (i.e. slowest) "Long range" Ewald

It can be seen that physically classes 1 and 2 are intra-molecular terms, 1 corresponding to (extremely!) approximate electronic effects and hard vibrational modes, class 2 to essentially soft vibrational modes, while 3 and 4 deal with inter-molecular effects. Please also note that each of the above short descriptions may cover more than one routine within the code, as each type of term may be modelled in more than one way.

Implementation

In fact as part of this work the *whole* RESPA scheme was re-implemented within DL_POLY. It was decided that for software sustainability grounds that it was better to re-engineer the whole scheme. The main driver behind this was ultimately the reason behind this proposal: the number of required intervention points to support RESPA within DL_POLY was too high to be long term sustainable in an actively developed code. Instead a reorganisation of the scheme described by Humphreys allowed the vast majority of the changes to be restricted

[8] Tuckerman, Berne and Martyna, J. Chem. Phys., 97 (3), 1990-2001 (1992)

[9] Humphreys, Freisner and Berne, J. Phys. Chem., 98, 6885-6892 (1994)

[10] Courant, R.; Friedrichs, K.; Lewy, H. (1928), "Über die partiellen Differenzengleichungen der mathematischen Physik", *Mathematische Annalen* **100** (1): 32–74

to very high levels of the code, and many of the base force term evaluators and integrators required no modification at all. The essence of this is as follows. Consider a two class scheme within the velocity Verlet integration scheme, which one might outline as follows (again following Humphreys). In the below ***F*** represents a force term, ***v*** a velocity, ***r*** a position and *m* a mass

```
Evaluate force class 2 ($\mathbf{F}_2$)                                    Evaluate
Evaluate force class 1 ($\mathbf{F}_1$)                                    Evaluate
Do N = 1, n_time_steps
    $\mathbf{v}_j = \mathbf{v}_j + 0.5 * n_1 * \delta t * \mathbf{F}_{2j} / m_j$      for all j      Integrate
    Do $i_1$ = 1, $n_1$
        $\mathbf{v}_j = \mathbf{v}_j + 0.5 * \delta t * \mathbf{F}_{1j} / m_j$         for all j      Integrate
        $\mathbf{r}_j = \mathbf{r}_j + \delta t * \mathbf{v}_j$                                      Integrate
        Evaluate force class 1 ($\mathbf{F}_1$)                            Evaluate
        $\mathbf{v}_j = \mathbf{v}_j + 0.5 * \delta t * \mathbf{F}_{1j} / m_j$         for all j      Integrate
     End Do
     Evaluate force class 2 ($\mathbf{F}_2$)                               Evaluate
     $\mathbf{v}_j = \mathbf{v}_j + 0.5 * n_1 * \delta t * \mathbf{F}_{2j} / m_j$     for all j      Integrate
End Do
```

In the scheme we have indicated whether each step is an evaluation of forces or a step in the integration of the equations of motion. While simple this has a number of practical drawbacks for implementation in a large, general purpose MD code such as DL_POLY.

1. It can be seen that each and every stage requires knowledge of what class is being evaluated. In the integration steps this is due to the presence of n_1 when updating the velocities with the class 2 forces
2. If new force terms are added it is necessary that they are made "RESPA aware" *a priori* as otherwise they will be evaluated at multiple levels, and thus the equations will be incorrectly integrated

However, it is simple to rewrite the above in a form, which obviates much of these deficiencies. First let us introduce a weighted force of class C: $\mathbf{F}_{Cj}{}^{\mathbf{W}}=\sum_{k\leq C} n_{k-1}\mathbf{F}_{kj}$. Thus the weighted force of class C is simply the sum of all forces for classes up to and including class C, each term being weighted by the appropriate repeat number n_C. For the purposes of this we set n_0=1. Using this we may rewrite the above 2 class scheme as

```
Evaluate weighted force class 2 ($\mathbf{F}^W=\mathbf{F}_2{}^W$)                     Evaluate
Do N = 1, n_time_steps
    Do $i_1$ = 1, $n_1$
        $\mathbf{v}_j = \mathbf{v}_j + 0.5 * \delta t * \mathbf{F}_j{}^W / m_j$         for all j      Integrate
        $\mathbf{r}_j = \mathbf{r}_j + \delta t * \mathbf{v}_j$                                      Integrate
        If( $i_1$ != $n_1$) Then
            Evaluate weighted force class 1 ($\mathbf{F}^W=\mathbf{F}_1{}^W$)      Evaluate
        Else
            Evaluate weighted force class 2 ($\mathbf{F}^W=\mathbf{F}_2{}^W$)      Evaluate
        End If
        $\mathbf{v}_j = \mathbf{v}_j + 0.5 * \delta t * \mathbf{F}_j^W / m_j$        for all j        ***Integrate***
    End Do
End Do
```

This rewriting means that

1. Only the force terms require major modification, and then only those terms which are NOT class 1. The integrator terms now need no knowledge of the RESPA repeat counts.
2. Any force terms that are not made RESPA aware effectively default to class 1, and thus will be dealt with correctly (if not as efficiently as possible)

Further it is easily extended to any number of classes – only the last iteration of each class needs to be modified to include the class(es) above it appropriately weighted.

This is the basic idea behind the new implementation. Over and above this it also provides facilities for arbitrary depth RESPA, as opposed to hardwired to 4 as in the original implementation. This is covered by provision of an iterator datatype and methods which act upon it which simulate the arbitrary depth nested loops that are required. Further the new implementation is very simple to switch on and off during the simulation as opposed the earlier method where this was more difficult. This is useful, for example, to ease handling "special" timesteps where one would prefer to take a normal, non-RESPA, step. An example of this is the very first step in the simulation, where avoiding the use of RESPA eases the implementation of restart runs, and for related reasons steps where restart data need be written are most easily handled by a non-RESPA step as this preserves backward compatibility with file formats for earlier versions of DL_POLY. A second use might be avoiding using RESPA during equilibration, as during that period the system may be in non-physical states which makes the integration of the equations of motion more sensitive. Another feature is that facilities are provided so that the class of each term can be changed dynamically, again as opposed to being hardwired in like before. Currently this is not exposed to the user, but could be in the future if required (and coupled with the arbitrary depth support could provide a huge degree of control for knowledgeable users), and also looks forward to work to be performed by Ian Bush during his EPSRC RSE Fellowship.

In terms of the code base the main parts of the implementation are in 3 modules. One provides the arbitrary depth iterator type, one implements a general RESPA type scheme on top of that, and the final one interfaces to DL_POLY in particular, and also holds any DL_POLY specific data. We hope that this structure will facilitate the use of this code in other MD applications since the vast majority is NOT DL_POLY specific.

Results

All results presented are for DL_POLY compiled with the gnu compiler collection, with simply –O3 used for optimisation.

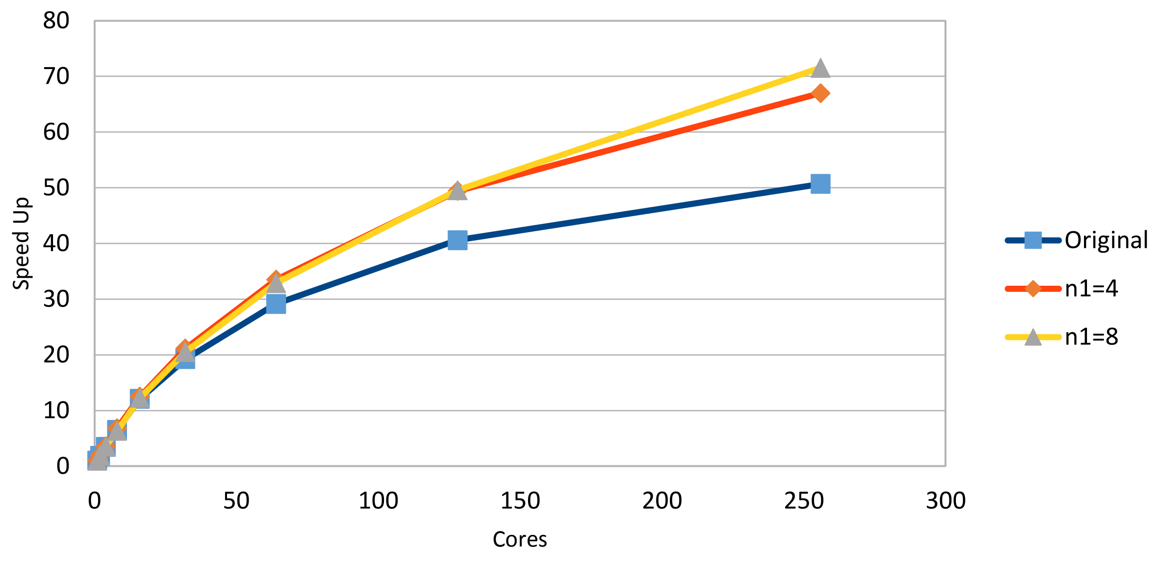
The proposal addressed 2 major areas, isobaric ensembles and the use of rigid bodies within DL_POLY. However, as we have re-implemented the whole of RESPA within DL_POLY we will start by presenting results for the simplest case, NVE dynamics; the shorthand here refers to the system being simulated with a constant number of particles (N), at a constant volume (V) and with a constant internal energy (E). The case chosen is a small NaCl simulation, consisting of 27,000 particles. The simulation was carefully equilibrated and then averages where taken over a final 80ps run. This case is chosen as it is relatively simple for RESPA, there are only 2 classes and so only 1 parameter to vary (n_l in the above scheme). Given the simulation is within a constant energy ensemble one would expect the use of RESPA to decrease the run time at the cost of poorer energy conservation as a function of n_l.

The first table shows the results, the runs being on 32 cores.

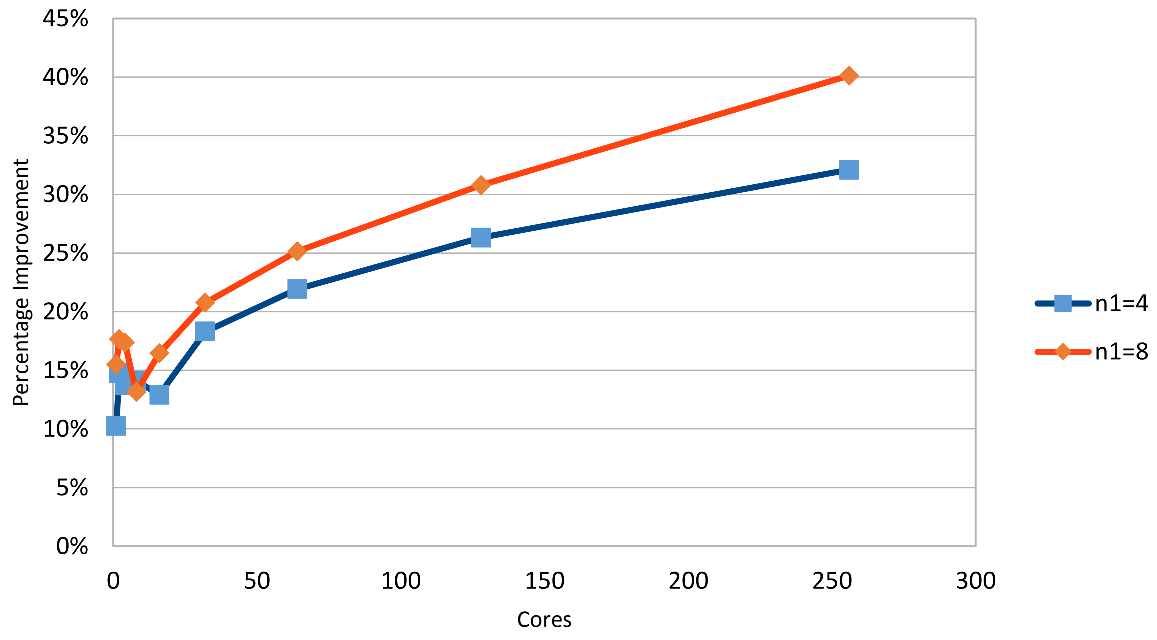
n_l	Average Energy	RMS Energy Fluctuations	Time to solution	Percentage Improvement
Original Code	-9.6757E+008	5.9435E+003	1986.664	0.00%
No RESPA	-9.6757E+008	5.9435E+003	1985.851	0.04%
1	-9.6757E+008	5.9435E+003	2013.498	-1.33%
2	-9.6757E+008	6.1770E+003	1753.202	13.32%
4	-9.6757E+008	6.1494E+003	1648.449	20.52%
6	-9.6756E+008	6.5640E+003	1564.91	26.95%
8	-9.6756E+008	7.4027E+003	1546.719	28.44%
12	-9.6755E+008	1.1688E+004	1516.386	31.01%
16	-9.6753E+008	2.1880E+004	1498.641	32.56%

The results bear out the expectations. The first 3 rows are “null results” showing that the introduction of RESPA has not broken the base code, and that RESPA with each class taking the same amount of iterations again does not affect the integration of the equations of motion. However, it can be seen that it does have a small negative affect on the run time. The reason for this is the structure of the loops in the short-ranged coulomb term evaluation stage, which as currently structured means the test for this being a RESPA run is made many times more than required. This is easily fixed and will be in the full release. On increasing n_l it can be seen that for low values the same energy is returned and that the run time markedly decreases at the cost of slowly increasing fluctuations in the energy. Higher values of n_l eventually affect the energy measured. Pragmatically looking at the above results one might use a value of n_l of 2-8 depending upon the accuracy required, with a nett gain of around 15-25% in performance.

RESPA also helps scaling. This is because the higher level classes describe interactions between larger groups of atoms, and thus due to DL_POLY’s domain decomposed data distribution require more communication of atomic data between processes. Hence if they are evaluated less often the reduction in communications overhead improves the scaling of the code. This is shown in the figure below:



The second figure shows the percentage speed improvement. It can be seen that the benefit of RESPA is greatest at the highest core accounts due to the improvements in scalability that it enables:

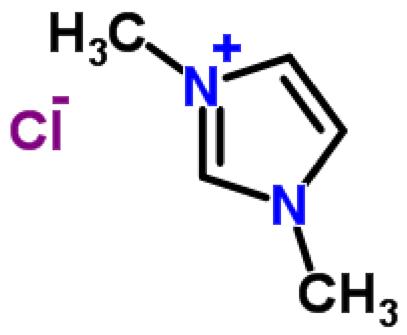


Thus the simplest case, constant number of particles (N), constant volume (V) and constant energy (E) is correct and brings computational benefits to the user. Next we consider one of the focuses of the new work, constant temperature and pressure simulations by a NPT simulation of the system above; everything is the same as the previous case, only the ensemble has been changed from NVE to NPT. DL_POLY has many ways to implement a NPT ensemble, here we use Hoover's method, one of the more popular techniques. The temperature was set to 500 K and the pressure to 1 atm, as this closely corresponds to the state point being simulated in the NVE simulations performed above.

n_l	Average Energy	Energy Fluctuations	Average Temperature	Average Pressure	Time	Improvement
1	-1.0126E+009	1.1281E+002	5.0000E+002	-9.9988E-001	2248.445	0.00%
2	-1.0126E+009	3.8674E+002	4.9998E+002	-1.0080E+000	2041.01	10.16%
4	-1.0126E+009	7.6880E+002	4.9991E+002	-1.0121E+000	1916.473	17.32%
6	-1.0126E+009	1.3541E+003	4.9977E+002	-1.0135E+000	1821.454	23.44%
8	-1.0126E+009	2.5619E+003	4.9958E+002	-1.0142E+000	1790.221	25.60%
12	-1.0125E+009	6.6238E+003	4.9901E+002	-1.01490E+000	1801.473	24.81%
16	-1.0125E+009	1.3427E+004	4.9817E+002	-1.01530E+000	1778.23	26.44%

Again the runs presented are using 32 processes. A similar story to the NVE ensemble can be seen. Increasing n_l reduces the run time (in fact in the above n_l=1 is the original code before modification which is used as the reference) at the cost of increasing errors in the integration of the equations of motion, as can be seen in the constants drifting from their appropriate values, and the fluctuations in the energy increasing. The gain in performance is similar though slightly less than in the NVE case, possibly reflecting the more expensive integrator being used here, while the range of appropriate values for n_l is roughly as before.

The other main focus of this project was including rigid bodies in the RESPA scheme. This is illustrated in the results below which is a NVE simulation of the ionic liquid 1,3-dimethylimidazolium chloride for 96 ps. Ionic liquids are an interesting class of compounds which find increasing use as highly polar solvents in a number of chemical processes. The simulation consisted of 44,352 ions, with the 1,3-dimethylimidazolium ions (pictured below) modelled as rigid particles.

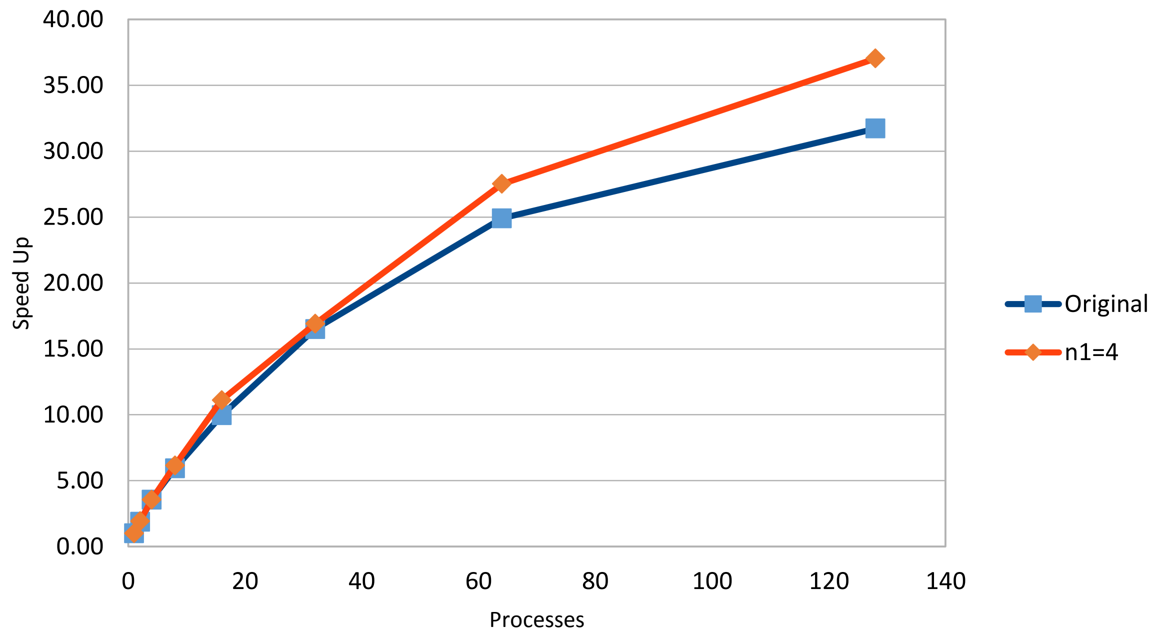


We again have a 2 class system, making comparison with the results above easy. As usual the simulation was carefully equilibrated before any results were obtained. These results are shown in the table below, and again 32 cores were used:

n_l	Average Energy	Energy Fluctuation	Time	Percentage Improvement
Original Code	-1.9829E+006	9.4554E+000	1455.087	0.00%
2	-1.9829E+006	9.5713E+000	1300.287	11.91%
4	-1.9829E+006	1.0014E+001	1219.921	19.28%
6	-1.9829E+006	1.4197E+001	1181.498	23.16%
8	-1.9829E+006	2.1610E+001	1150.537	26.47%
12	-1.9827E+006	5.9599E+001	1132.658	28.47%
16	-1.9825E+006	1.1234E+002	1128.462	28.94%

Again we have a similar set of results which correspond with the expectations of the method, and again a 10-25% improvement can be obtained with the RESPA parameter in the range 2-8 without a too marked degradation of the quality of the results.

For this case the scaling with process count was also investigated, and a similar picture to above was obtained. This is shown below



Again the newly implemented RESPA scheme achieves a noticeable improvement in the speed up.

Current Status

Unfortunately the modifications just missed the code freeze for the last release of DL_POLY, which occurred in March 2016. However the code is under the CCPForge DL_POLY git repository, and will be included in the next (non bug-fix) release of DL_POLY.

Conclusions and Summary

The work to extend the RESPA scheme to the whole of DL_POLY has been demonstrated. It has consistently been shown to give at least a 15-25% improvement in performance at fixed process count, and has also been shown to give noticeable improvements to the scalability of DL_POLY.

Acknowledgements

Ian Bush would like to acknowledge the usefulness of the original implementation written by Dr Asimina Manioupolou which taught me many lessons about the pitfalls of implementing RESPA within a large code like DL_POLY. The devil really is in the detail here!
This work was funded under the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk)