DL_POLY_3: new dimensions in molecular dynamics simulations *via* massive parallelism

Ilian T. Todorov,*[ab] William Smith,[a] Kostya Trachenko[b] and Martin T. Dove[b]

Received 19th December 2005, Accepted 15th March 2006
First published as an Advance Article on the web 4th April 2006
DOI: 10.1039/b517931a

DL_POLY_3 is a general-purpose massively parallel molecular dynamics simulation package embedding a highly efficient set of methods and algorithms such as: Domain Decomposition (DD), Linked Cells (LC), Daresbury Advanced Fourier Transform (DAFT), Trotter derived Velocity Verlet (VV) integration and RATTLE. Written to support academic research, it has a wide range of applications and can run on a wide range of computers; from single processor workstations to multi-processor computers. The code development has placed particular emphasis on the efficient utilization of multi-processor power by optimised memory workload and distribution, which makes it possible to simulate systems of the order of tens of millions of particles and beyond. In this paper we discuss the new DL_POLY_3 design, and report on the performance, capability and scalability. We also discuss new features implemented to simulate highly non-equilibrium processes of radiation damage and analyse the structural damage during such processes.

Introduction

The molecular dynamics (MD) simulation technique has developed rapidly in recent years to utilise emerging advancements in hardware by better scalable numerical algorithms based on novel mathematical concepts[1–9] and better engineered software solutions.[10–12] Researchers have been provided with the power to simulate systems of sizes unthinkable just a few years ago with a higher degree of realism and great interaction complexity such as bio-molecules, phase-to-phase interfaces, surface growth and deposition, and clusters.

In this article we present the DL_POLY_3 program that blends a number of modern numerical techniques[1–7,11,12] into a program of great power and sophistication. The article describes the design, new functionality, performance, and capability and scalability limits of DL_POLY_3. The following section outlines the origin and availability of the package. Next described are the new code design and underlining concepts with emphasis on the parallelisation strategy. The subsequent section describes the new functionality available in the simulation package. Then follows a section reporting the simulation performance with three model systems on an IBM SP4 cluster with discussion on performance and limitations.

The DL_POLY project

The project originated in 1994 by the effort of the UK's CCP5[13] to create an MD package to meet the needs of the UK academic community for a general-purpose MD code. The objective was to develop a new community code to exploit the emerging parallel computers to the fullest advantage, which supports a wide range of applications: for example macromolecules (both biological and synthetic), complex fluids and ionic materials of high complexity, and permits verification and extension by the user (meaning that the package should be available in the form of source code—"open" software policy). The DL_POLY_3 program is one of the outcomes of these requirements.†

The DL_POLY package is available free of cost to academic scientists pursuing research of non-commercial nature and has been applied in a broad range of scientific studies since its first official release in 1996[14] as DL_POLY_2. Over 1400 user licences have been taken out worldwide since then. The original code, DL_POLY_2, was based on a Replicated Data (RD)[14] parallelisation strategy, but recent developments have introduced the Domain Decomposition (DD)[10] version, DL_POLY_3, to permit simulation of systems of the order of tens of millions of atoms and beyond. As we shall see in the Performance and discussion section, DL_POLY_3's inherent parallelism allows close to perfect parallelisation up to impressively high processor counts.

Design and concepts

Introduction

Nowadays, we more often speak of machine computing power in terms of number of processors (CPUs) rather than a single processor power. Although computer technology evolves rapidly and new more powerful CPU solutions are continuously appearing, it has long been recognised that the

[a]*Computational Science & Engineering Department, CCLRC Daresbury Laboratory, Daresbury, Warrington, UK WA4 4AD. E-mail: I.T.Todorov@dl.ac.uk*
[b]*Department of Earth Sciences, University of Cambridge, Downing Street, Cambridge, UK CB2 3EQ*

† There is also the original package DL_POLY_2 and a number of spin-offs such as DL_MULTI, DL_PROTEIN and DL_MESO. For more information visit: http://www.cse.clrc.ac.uk/msi/software/DL_POLY/index.shtml

way to speed up computational applications is to harness the power of multi-CPU assemblies simultaneously. This has led to the appearance of power clusters of dedicated (multi-)CPU nodes as well as development of technologies for utilizing the collective power of non-dedicated single-CPU platforms such as Condor pools.[15] It is the former that has also driven the development in network and network carrier solutions to provide faster message passing interfaces (MPI). Computer languages suitable for high-precision numerical calculations: Fortran, C/C++, *etc.*, have been supplied with extension libraries providing means to deal with inter-CPU awareness and communications. Currently, there are two mainstream technologies, PVM and MPI,[16,17] that provide these and only Fortran and C/C++ are fully supported as they are the most commonly used in coding ''number-crunching'' algorithms. Many simulation codes have been upgraded to implement parallelism in various ways in order to make it possible to address problems with higher degrees of realism (larger sizes and increased complexity) at minimal additional cost, which was not an option when they were only available in serial form.

Programming choices

We have chosen Fortran 90 as a standard for our new code. The language offers all the functionality needed for numerical coding and avoids the scripting and multi-concept error-proneness of C/C++. We have found that software development for scientific and engineering computing in Fortran 90 is preferable as well as compiler optimisation of code is more reliable than that in C[18,19] due to its rigid, numerically orientated syntax and limited variety of concepts. The Fortran 90 modularisation concept is fully employed to logically separate and distribute common (science-, maths- and semantics-wise) sets of variable declarations, methods and initialisations in modules. Adopting modularisation allows a lego-like build of further enhancements and new implementations such as force fields, scientific methodologies and numerical algorithms. It can also provide various safety features similar to those that object oriented programming (OOP) languages provide, such as data encapsulation, overloading, *etc*. Other scientific codes, such as CASTEP,[20] have taken a similar engineering approach to build a safe, reliable, and easy to maintain and develop further code infrastructure.

We have approached the modularisation of DL_POLY_3 in the following manner:

1. kinds module—defining globally the bit precision for real and integer parameters and variables at compile time

2. communication module—containing global communication routines and functions

3. setup module—defining global constants at compile time as well as run time specific parameters used as array bounds to all local arrays

4. domains module—containing a domain decomposition manager and fundamental domain decomposition mapping arrays

5. parse module—containing generic tools for parsing textual input

6. site module—containing global configuration and configuration related data

7. configuration module—containing domain localised configuration and configuration related data

8. interaction modules (14)—defining global as well as local parameters and variables for each specific interaction (*e.g.*: van der Waals, metal, Tersoff, three-body, four-body; core–shell, constraints, potential of mean force, tethers; chemical bonds, bond angles, dihedral angles, inversion angles; external force field)

9. statistics module—containing statistical arrays and variables

10. defects module—containing parameters and variables needed for defect detection algorithms and subroutines.

Fortran 90 also provides other safety features from which we have benefited and which we have set as conventions for the programming style in our code:

1. explicit type declaration and immediate initialisation of all types of variables

2. explicit declaration of modular variables and methods used in functions and subroutines

3. explicit declaration of intent for all arguments in calling sequences

4. explicit declaration of privacy status of variables and methods in modules

The adopted style has very few overheads in terms of code size and it pays off in implementation and testing stages when problem detection starts at compilation time.

The inter-CPU communication is implemented using MPI. Most of the communication in the code is implicit, based on dedicated functions and subroutines developed as methods in a Fortran 90 module (comms_module). However, there are a few subroutines that use explicit MPI calling, which implement either non-parallelisable tasks such as reading and writing to hard disk (HD), or somewhat intricate tasks of exchanging domain boundary data (see below). The rest of the subroutines incorporate parallelism automatically (see below). It is important to note that the code can also run on *serial* computers without any modification.

Parallelisation

DL_POLY_3 DD parallelisation[10] is generally an extension of the *link-cell* (LC) method[1–3] in which the simulation cell is divided into subcells, Fig. 1a, the width of which is not less than the radius of the cut-off applied in the potential energy and force calculations. Thus, by construction, an atom can only interact with atoms in the same subcell or in a subcell that is an immediate neighbour. This allows an inexpensive building of a *link list* for fast location of interacting atoms, leading to an overall algorithmic scaling of $\approx O(N)$, where N is the number of atoms. The parallel adaptation of this algorithm requires *a priori* partitioning of the simulation cell into geometric domains, each of which is allocated onto a processor of the parallel machine. Although the mapping of the domains on the array of processors can be a non-trivial problem in general, an elegant solution for machines like hypercubes exists. This solution imposes the partitioning into spatially identical domains. The exchange of link-cell contents from the borders of each domain (the *halo data,* Fig. 1b) between neighbouring processors establishes the contiguity of the

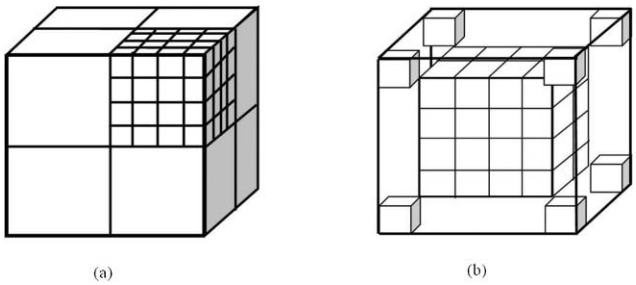

Fig. 1 Sketch of DL_POLY_3's link cell and domain decomposition. Sketch (a) represents the division of the simulation cell into geometric domains (shown by larger cubes dividing the volume) and the division of domains into link cells (shown by smallest cubes dividing a domain). Sketch (b) represents the construction of halo data around a domain. The central cube is a domain divided into link cells. The surrounding cube incorporates atomic data from neighbouring domains and is composed of the link cells from the edges of those domains.

global system and simultaneously incorporates any periodic boundary conditions. It is worth noting that for successful building of domain halos, the boundary link-cell data are only passed in successively complementary directions (in 3D: north–south and back, east–west and back and up–down and back) as at each pass (*i.e.* north–south and back) the exchanged data must be re-sorted before the next pass. This is necessary to ensure the corner and edge link-cell data are correctly exchanged between domains sharing edges and corners, rather than faces.

The exchange of halo data is the key communication step in DL_POLY_3 as no further communication between the nodes is necessary until after the equations of motion have been integrated. However, if bond constraints are present in the system, the equations of motions are modified to include constraint solvers, RATTLE[6] for velocity Verlet (VV[4,5]) integrators and SHAKE[21] for leapfrog Verlet (LFV[22–24]) integrators. The constraint solvers are iterative algorithms that add incremental corrections to the positions, velocities and forces of constrained particles until the bond lengths for all constraints in the system equal the corresponding predefined constraint bond lengths to within a given tolerance. In RATTLE there is a second iterative stage which adds incremental corrections to the velocities and forces of constrained particles so that the relative velocity of the two particles in the constraint is zero along the constraint vector within a tolerance.

Constraint algorithms involve extra communication at each iteration when constraint bonds cross domains. A domain crossing constraint bond is present in two domains as each domain contains one of the constrained particles and the other in its halo. As constraint particles change their positions, velocities and forces at each iteration and each domain updates these only for particles present in it, it is crucial to update these for constraint particles that lie in their halo by additional communication. This is why systems with constraints are bound to have lower parallelisation efficiency than systems without constraints. After the equations of motion have been integrated the particles which have moved out of their original domain must be reallocated to a new domain.

Although the DD algorithm is designed by construction to treat only systems with short-range forces, the treatment of coulombic forces (as well as any other long-ranged forces) can also be fitted in easily. For the evaluation of the coulombic interactions in molecular simulations, DL_POLY_3 incorporates an adaptation of the Smoothed Particle Mesh Ewald method (SPME)[7] as a means of enhancing the performance of the traditional Ewald sum calculations.[22] This adaptation gains in performance over the traditional Ewald sum implementations through the use of (complex) three dimensional Fast Fourier Transforms (3D FFTs)[25] when computing *the reciprocal space* contributions to the sum. DL_POLY_3 uses the Daresbury Advanced Fourier Transform (DAFT),[11] which is a novel, fully memory distributed, parallel implementation of the 3D FFT that conforms to the DD concept.

Portability

DL_POLY_3 is highly portable as compilation requires only a Fortran 90 compiler and complementary MPI libraries to handle communications. Although it is designed for memory distributed parallel computers the package can also be used on single processor machines (no MPI needed). The code has been ported on a large variety of platforms including propriety HPC such as Cray T3E, IBM SP4, Sun Fire 880, *etc.* as well as clusters of dedicated CPUs such as Beowulf Linux clusters.

Functionality

DL_POLY_3 is a general purpose package with continuously increasing functionality.[10,26] It will suffice here to outline only some recently developed functionality in DL_POLY_3 such as Tersoff interactions,[27] relaxed shell model,[28] Brownian dynamics *via* Langevin thermostat[29] for LFV integration and Langevin impulse[30] for VV integration, and potential of mean force integration.[31]

Features driven by radiation damage research

Radiation damage (RD) research has long been required to build theories, to explain experiments, and improve understanding of how the chemical nature of materials defines the resistance to amorphisation by radiation damage and the processes of damage recovery, and, ultimately, make predictions of prospective materials for use in nuclear waste encapsulation. Theoretical studies[32–36] involve highly non-equilibrium MD simulations—RD cascades of high-energy ions propagating in crystalline and damaged structures. Such simulations have not only benefited from the exceptional capabilities of DL_POLY_3 to handle huge systems[34–36] required by the high energies involved but also driven its development. This development has resulted into the implementation of *variable timestep*, *pseudo* and *defect detection* algorithms.

The *variable timestep* option requires the user to specify an initial guess for a reasonable timestep for the system (in picoseconds). The simulation is unlikely to retain this as the operational timestep however, as the latter may change in response to the dynamics of the system. The option is used in

conjunction with the default values of *maxdis* (default 0.03 Å) and *mindis* (default 0.10 Å), which can also be optionally altered if used as directives (note the rule *maxdis* $\geqslant$ 2.5 *mindis* applies). These distances serve as control values in the variable timestep algorithm, which calculates the greatest distance a particle has travelled in any timestep during the simulation. If the maximum distance is exceeded, the timestep variable is halved and the step repeated. If the greatest move is less than the minimum allowed, the timestep variable is doubled and the step repeated. In this way the integration timestep self-adjusts in response to the dynamics of the system; the simulation slows down to account accurately for the dynamics during the most violent stages in radiation damage cascades and then speeds up to utilise effectively CPU time when the system cools down.

The *pseudo* option attaches a Langevin (stochastic) thermostat[22,30,30] at the MD cell boundaries. It requires the user to specify the width of the thermostat buffer, d, compliant with the rule 2 Å $\leqslant d \leqslant$ *width*/4, where *width* is the minimum MD cell width. Every particle within the buffer is coupled to a viscous background and a stochastic heat bath, such that

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} r_i(t) &= v_i(t) \\ \frac{\mathrm{d}}{\mathrm{d}t} v_i(t) &= -\lambda(t) v_i(t) + \frac{1}{m_i}[f_i(t) + R_i(t)] \end{aligned} \tag{1}$$

where $\lambda(t)$ is a friction parameter and $R(t)$ is the stochastic force with zero mean that satisfies the fluctuation–dissipation theorem:

$$<R_i^{\alpha}(t) R_j^{\beta}(t')> = 2\lambda(t) m_i k_B T \delta_{ij} \delta_{\alpha\beta} \delta(t - t') \tag{2}$$

where superscripts denote Cartesian indices, subscripts particle indices, m_i is the mass of particle i, k_B the Boltzmann constant and T the system target temperature. The algorithm is implemented in two stages. First, random forces are generated for all particles within the thermostat buffer. Here, care must be exercised to prevent introduction of non-zero net force when the random forces are added to those which arise from the force field of the system. The second stage is to rescale the kinetic energy of the thermostat bath so that particles within have Gaussian distributed kinetic energy with respect to the target temperature at the end of each MD step and determine the pseudo thermostat friction

$$\lambda(t) = \mathrm{Max}\left(0, \sum_i [f_i(t) + R_i(t)] \cdot v_i(t) \Big/ \sum_i m_i v_i^2(t)\right) \tag{3}$$

for the first stage. Care must be exercised to prevent introduction of non-zero net momentum. The effect of this algorithm is to relax the buffer region of the system on a local scale and to effectively dissipate the incoming excess kinetic energy from the rest of the system, thus emulating a pseudo-infinite environment surrounding the MD cell. Stochastic boundary thermostats previously used in studies of chemical reactions and other localised processes[37,38] have now been re-employed in radiation damage simulations[39,40] to provide effective energy dissipation and faster temperature relaxation which help speed up simulations as well as simulate higher energy impacts.

The *defect detection* tool uses an algorithm that compares the simulated MD cell to a reference MD cell. The former defines the actual positions of the particles and their atom types and the latter is taken here to be the structure of the undamaged lattice. If a particle, p, is located in the vicinity of a site, s, defined by a sphere with its centre at this site and a user defined radius, 0.3 Å $\leqslant R_{def} \leqslant$ 1.3 Å (default value 0.75 Å), then the particles is a *first hand* claimee of s, and the site is not vacant. Otherwise the site is presumed vacant and the particle is presumed a general interstitial. If a site, s, is claimed and another particle, p', is located within the sphere around it, then p' becomes an interstitial associated with s. After all particles and all sites are considered, it is clear which sites are vacancies. Finally, for every claimed site, distances between the site and its *first hand* claimee and interstitials are compared and the particle with the shortest one becomes the *real* claimee. If a *first hand* claimee of s is not the *real* claimee it becomes an interstitial associated with s. At this stage it is clear which particles are interstitials. The sum of interstitials and vacancies gives the total number of defects in the simulated MD cell. Note that the algorithm *cannot* be applied safely if R_{def} is larger than half the shortest interatomic distance within the reference MD cell since a particle may: (i) claim more than one site, (ii) be an interstitial associated with more than one site, or both (i) and (ii). Low values of R_{def} are likely to lead to slight overestimation of the number of defects. If the simulation and reference MD cells have the same number of atoms then the total number of interstitials is always equal to the total number of defects. It is worth noting that the implementation of this algorithm makes extensive use of LC and DD methods to achieve an overall algorithmic scaling of $\approx O(N)$ rather than $\approx O(N^2)$ which is not acceptable for large scale simulations ($\approx 10^6$ particles) as is the case in modern radiation damage research.[35,36]

Performance and discussion

DL_POLY_3 performance was evaluated by a set of test MD simulations performed on the HPC*x* (IBM SP4 cluster—http://www.hpcx.ac.uk) super-cluster at Daresbury Laboratory (the UK's 3rd and world's 28th fastest‡) by exclusive use of its resources. The tests were based on three model systems, at conditions as outlined in Table 1: (i) solid Ar with Lennard-Jones interactions between Ar–Ar, (ii) NaCl with van der Waals interactions between Na^+–Na^+, Na^+–Cl^- and Cl^-–Cl^-, and Coulomb forces between the ions, and (iii) SPC (single point charge) water with Lennard-Jones interactions between O^-–O^-, Coulomb forces between the ions and three constraints per water molecule: O–H1, O–H2 and H1–H2. All systems were generated with perfect crystal structures (apart from SPC water) and lattice parameters with values obtained from previously equilibrated structures at the same simulation conditions as shown in Table 1.

The size-per-CPU and cutoff values for each system were so chosen as to ensure all systems have the same domain halo volume on average, so that relatively the same volume of MPI messaging for domains boundary data exchange is required

‡ At the time of writing!

Table 1 The model systems simulated using DL_POLY

System	Size per CPU [particles]	Ensemble [type]	Cutoff/Å	Temperature/K	Pressure/k bar
Solid Ar	32 000	NVE	9	4.2	0.001
NaCl	27 000	NVE	12	500	0.001
SPC water	20 736	NPT Berendsen 0.5 0.75	8	300	0.001

Table 2 DL_POLY_3.04 scaling performance on HPC*x* for solid Ar simulations as described in the text. Time-per-timestep (in seconds, averaged over 10 timesteps), start-up and close-down times are listed as a function of number of CPUs used in parallel. Based on time-per-timestep values per different CPU counts, also listed are values of the speed gain and the relative speed gain

CPUs	Speed gain	Relative speed gain	Time-per-timestep/s	Start-up time/s	Close-down time/s	System size/atoms
1			0.59	1.02	0.73	32 000
2	1.97	1.97	0.60	1.44	1.59	64 000
4	3.87	1.97	0.61	1.92	3.09	128 000
8	7.96	2.06	0.59	2.95	9.84	256 000
16	15.69	1.97	0.60	5.62	10.50	512 000
32	29.96	1.91	0.63	10.37	20.30	1 024 000
64	59.14	1.97	0.64	20.71	43.13	2 048 000
128	116.50	1.97	0.65	43.16	88.34	4 096 000
256	231.86	1.99	0.65	86.08	178.13	8 192 000
512	448.44	1.93	0.67	195.03	361.18	16 384 000
1024	838.36	1.87	0.72	411.25	740.99	32 768 000

Table 3 DL_POLY_3.04 scaling performance on HPC*x* for NaCl simulations as described in the text. Time-per-timestep (in seconds, averaged over 10 timesteps), start-up and close-down times are listed as a function of number of CPUs used in parallel. Based on time-per-timestep values per different CPU counts, also listed are values of the speed gain and the relative speed gain

CPUs	Speed gain	Relative speed gain	Time-per-timestep/s	Start-up time/s	Close-down time/s	System size/ions
1			2.82	3.39	0.66	27 000
2	1.95	1.95	2.89	3.78	1.25	54 000
4	3.80	1.95	2.97	4.68	2.41	108 000
8	7.69	2.02	2.93	5.21	5.07	216 000
16	14.96	1.95	3.01	7.66	9.77	432 000
32	27.79	1.86	3.24	13.04	19.95	864 000
64	54.07	1.95	3.34	20.78	40.17	1 728 000
128	105.12	1.94	3.43	38.20	81.20	3 456 000
256	206.24	1.96	3.50	78.66	166.05	6 912 000
512	388.41	1.88	3.71	168.11	331.87	13 824 000
1024	705.81	1.82	4.09	340.86	664.33	27 648 000

between neighbouring domains for each system at any timestep. Thus the parallelisation performance is purely based on the complexity of the different force fields and on the additional communication the former involved.

To detect and compare the parallelisation efficiency between systems and between different processor counts the following construction was employed. Whenever the number of CPUs was doubled the simulated systems were also doubled in size,§ ensuring that the link-cell algorithms¶ and the domain halo volume for each system remained the same for any processor count (this type of scaling is also referred as weak-scaling). Thus, if parallelism were ideal the simulation time-per-timestep for each system would be the same for any processor count. As the effectiveness of the communication is known to worsen with increasing processor count and increasing volume of messages per processor, it is therefore expected to assume that systems with constraints (requiring additional overhead communication) will show faster decline of parallelisation efficiency with processor count.

Tables 2, 3 and 4 present simulation performance data of the investigated systems, as described above, on HPC*x*. The tables list four main data values: time-per-timestep, start-up time, close-down time and system size as functions of processor count. From the first one, one can recalculate the relative speed gain‖ (RSG) and (absolute) speed gain (SG) as functions of processor count. These are plotted in Fig. 2 and 3. Perfect parallelisation corresponds to a constant relative speed gain factor of 2 and good parallelisation corresponds to a constant factor of 1.75. The results show clearly that the parallelisation is excellent although it is not perfect as the time-per-timestep increases slowly with increasing processor

§ System sizes were doubled cyclically in x, then y and then z directions.

¶ It is crucial to note that increased parallelisation efficiency remains even when the link-cell algorithm is used inefficiently.

‖ The relative speed gain at 2 N CPUs is defined as the ratio of time-per-timestep for 2 N CPUs to that for N CPUs from the simulation on the same system. Since the system size changes in the same fashion as the processor count size, the definition in this case changes to "the ratio of time-per-timestep for 2 N CPUs to that of N CPUs times two". The (absolute) speed gain is a product of the relative speed gains and reflects the simulation speed-up if the processor count increases from 1 to N at constant system size.

Table 4 DL_POLY_3.04 scaling performance on HPC*x* for SPC water simulations as described in the text. Time-per-timestep (in seconds, averaged over 10 timesteps), start-up and close-down times are listed as a function of number of CPUs used in parallel. Based on time-per-timestep values per different CPU counts, also listed are values of the speed gain and the relative speed gain

CPUs	Speed gain	Relative speed gain	Time-per-timestep/s	Start-up time/s	Close-down time/s	System size/ions
1			1.90	1.98	0.53	20 736
2	1.81	1.81	2.10	2.35	1.03	41 472
4	3.36	1.85	2.26	2.87	2.06	82 944
8	7.11	2.12	2.14	3.55	3.81	165 888
16	13.54	1.90	2.24	5.40	7.64	331 776
32	23.53	1.74	2.58	9.28	15.45	663 552
64	43.80	1.86	2.77	16.84	30.48	1 327 104
128	84.24	1.92	2.89	34.62	59.23	2 654 208
256	158.97	1.89	3.06	64.63	122.30	5 308 416
512	285.41	1.80	3.41	133.03	247.98	10 616 832
1024	495.35	1.74	3.93	274.72	506.67	21 233 664

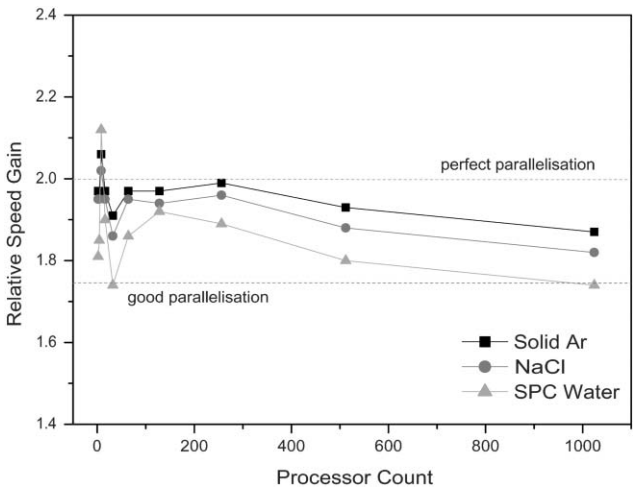

Fig. 2 DL_POLY_3 relative speed gain plotted as a function of processor count from simulation data as shown in Tables 2, 3 and 4. The upper dotted line indicates the parallelisation limit also called perfect or embarrassing parallelisation. The lower dotted line indicates the standard for a good parallelisation.

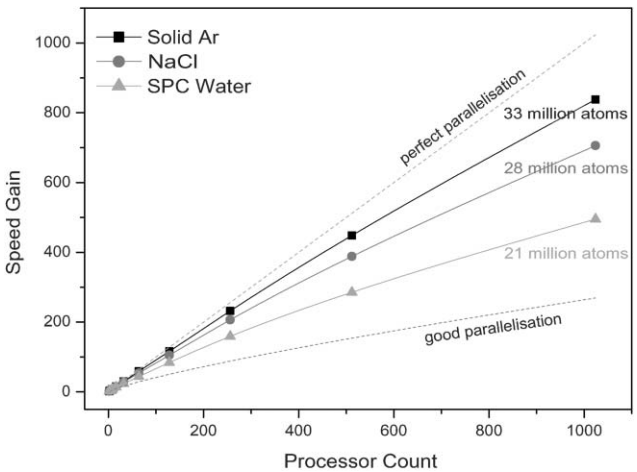

Fig. 3 DL_POLY_3 speed gain plotted as a function of processor count from simulation data as shown in Tables 2, 3 and 4. The upper dotted line indicates the parallelisation limit also called perfect or embarrassing parallelisation. The lower dotted line indicates the standard for a good parallelisation.

count. This is a clear indication that there is time loss due to slow-down in organising local MPI messaging (each-to-each) as well as in collective (global) communication operations.

The SPME summation accuracy (10^{-6})** was kept constant for all NaCl and SPC water simulations together with the constraint tolerance accuracy (10^{-5}) for all SPC water simulations. This imposes some small extra memory overheads as well as more expensive SPME electrostatics due to 3D FFT calculations which scale as $N \log N$ rather than linearly (N) with problem size. This is well shown in Fig. 2 where the performance for NaCl and SPC water is worse than that for solid Ar. We also see from Fig. 2 by the divergence of RSG curves that the performance for the systems with Coulombic forces decreases with increasing system size.

As expected, the SPC water system exhibits the worst parallelisation performance. This is due to increased MPI communications dictated by the increase in the total number of constraint bonds crossing domain boundaries with increasing the number of domains (CPUs). As mentioned in the Parallelisation section, constraint bonds crossing domains need to be refreshed during each iteration cycle of the constraint algorithms.

A careful look at Fig. 2 shows an interesting behaviour of the RSG at low processor count. To understand this we have to take a slightly deeper insight into the HPC*x*'s architecture. The HPC*x* system uses IBM p690+ Regatta H+ frames. Each Regatta frame consists of 32 1.7 GHz POWER 4 CPUs. In POWER 4 architecture, a chip contains two CPUs with 1.5 MB shared L2 cache as each processor has its own 192 kB L1 cache. Four chips (8 CPUs) are integrated in a multi-chip module (MCM) with 128 MB shared L3 cache, as each chip has a 3 × 3 distributed switch for fast L2 communication. An MCM has 8 GB of main memory available as 2 slots by 4 GB shared per pair of chips. Four MCMs comprise a Regatta frame and the total memory of 32 GB is shared between the 32 CPUs of the frame. Each frame runs its own AIX operating system (OS), which effectively decreases the available RAM for running non-OS applications by ≈20%. Inter-frame communication is handled by an IBM high performance "federation" switch.

** This corresponded to a 64 × 64 × 64 grid for the FFT on 1 CPU that increased cyclically by a factor of 2 as the CPU count increased by a factor of 2.

As we see in Fig. 2 the RSG behaviour is dictated by the internal frame architecture. The population of a Regatta frame is done in a round-robin way which tries to satisfy quasi-equal load per MCM. Apparently, the internal communications are suited to perform better on a fully rather than partially loaded MCM due to the distributed switches. Also, the memory distribution is maximised at 1 CPU per MCM (totals to 4 per Regatta frame) and 2 CPUs per MCM but on different chips (totals to 8 per Regatta frame). The last two facts explain why the RSG exhibits the "outstanding" performance on 8 CPUs. However, Fig. 2 also shows that RSG for 16 CPUs is approximately the same as that for 4 CPUs which suggest a cancellation of three competing trends on 16 CPUs: (i) decreasing the availability of L2 and L3 cache memory per CPU, (ii) increasing the usage of MCM internal L2 cache communications *via* distributed switches and (iii) increasing the usage of MCM external communications. However, once a Regatta frame is loaded fully there is a drop in RSG which can be explained by sharing work with the copy of the OS on the frame and increased communication latency due to triggering the inter-frame communication switch. After this point, however, the RSG seems to remain constant up until 256 CPUs (8 frames) after which a slight decline in performance is observed. Nevertheless, the DL_POLY_3 performance is extraordinary. As shown in Fig. 2 the RSG does not decrease below 1.75 even for high processor counts and furthermore the SG, Fig. 3, for the most CPU consuming system (SPC water) displays a performance better that the high standard of sequential 1.75 RSG (lower dashed line).

Maximum load tests involving the three test systems were also run on HPC*x*. We increased the MD simulation size load on one CPU incrementally until execution failed. As mentioned above the available memory per CPU on HPC*x* is $\approx$0.8 GB. The maximum load for the solid Ar system was $\approx$700 000 particles per 1 GB. This on 1024 CPUs on HPC*x* would add up to $\approx$610 million particles, which is well below the limit (2 147 483 647) DL_POLY_3 can handle on conventional 32-bit machines. Even this, of course, can be exceeded with careful effort and system sizes of 9.2×10^{18} made possible but then the package must be compiled in a 64-bit mode and run on a 64-bit (enabled) platform with considerable hard disk space required. For a one million particle system a configuration file (lattice parameters, positions, velocity and forces) in textual format is $\approx$0.25 GB.

The maximum load per CPU for the NaCl and SPC water systems amounted to $\approx$220 000 and $\approx$210 000 ions per 1 GB memory respectively. These cannot be scaled so directly to 1024 CPUs since some extra memory per CPU will be needed for the 3D FFTs in the SPME summations driven by the constant SPME precision. However, our estimate is a 1000 fold load on 1024 CPUs.

The reduced maximum load per CPU for the latter two systems to that of the solid Ar system can be easily explained. The maximum load per CPU is dependent on the available memory per CPU after force field arrays are allocated. The larger the complexity and/or the higher the accuracy of the force field description, the lower the limit of the maximum load per CPU. In DL_POLY_3, the memory demand for force field description as well as the CPU time for force calculations is proportional to the following factors:

1. number of atoms (reflecting inclusion of shells to account for polarisation *via* core–shell models)
2. number of intermolecular interactions
3. number of intramolecular interactions (including core–shell units, bond constraints, PMF constraints and tethers)
4. precision of Ewald summation
5. tolerance for constraint algorithms.

That is also why the time-per-timestep for the NaCl and SPC water systems is much larger than that of the solid Ar one, even though the latter has a larger number of particles per CPU.

It is worth noting that the parallelisation efficiency as defined and discussed above was defined in terms of weak scaling—when the ratio of particles (system size) to processor count remains a constant. Therefore, it does not match the strong scaling parallelisation efficiency defined by the code performance for a fixed size system. In general, we can write the time-per-timestep, t, for a system without constraints and long-range forces as

$$t = \alpha \cdot \left[\Delta \cdot \left(\frac{(M_x+2)\cdot(M_y+2)\cdot(M_z+2)}{M_x \cdot M_y \cdot M_z} - 1 \right) \right]^{\beta \cdot N^{\gamma}} \cdot \left[\rho \cdot (M_x+2)\cdot(M_y+2)\cdot(M_z+2) \right]^{l} \cdot \left(\rho \cdot M_x \cdot M_y \cdot M_z \right)^{v+f} \quad (4)$$

where M is a link cell dimension per domain in a designated direction (x,y,z), N is the processor count, ρ the particle density per link cell volume, f a constant dependent on the force-filed complexity, v an integration constant dependent on the integration complexity, l a link cell algorithm constant, Δ the network latency, and α, β and γ constants dependent on the platform. It is clear that for a fixed size system when N increases M decreases and therefore the dominant contribution to t is the overheads in communication at high processor counts and/or at low values of link cell dimensions per domain. In practice, the parallelisation efficiency of strong scaling could be considered to be very close to that of weak scaling if it was guaranteed that the link cell decomposition of the system in question does not drop below 4 link cells per dimension at any high processor count.

It is important to stress that there is some time penalty associated with loading up the simulation (start-up time) and dumping down the simulation (close-down time) since reading and writing to hard disk as well as inter-CPU communication one-to-all and all-to-one are serial tasks. These are problem-size dependent: system-size (number of particles) and domain count (number of CPUs). It is clear from Fig. 4 that they are linearly proportional to both, as indicated by the constant, but different slopes for different systems, of the start-up and close-down time plots. However, these are "one-off" operations during simulation and in real production runs (thousands of steps) their contributions can be neglected.

Last but not least we note that our tests to compare performance between DL_POLY versions 3.03 and 3.05 on the same three systems showed that the newer version was $\approx$25% faster and allowed $\approx$50% larger maximum load per CPU than the older one. In this way the design transition of

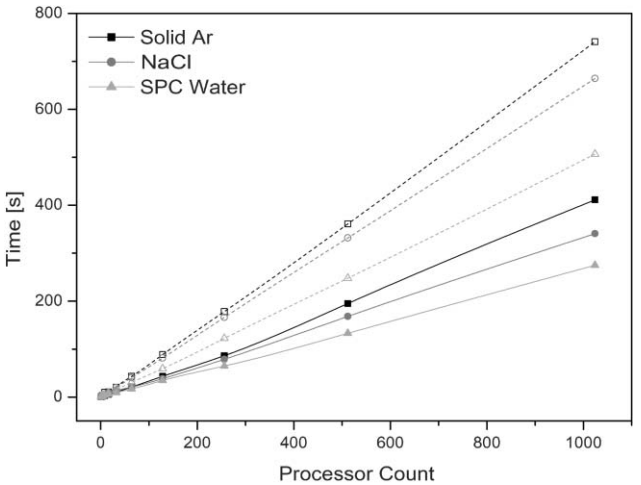

Fig. 4 DL_POLY_3 start-up (solid lines) and close-down (dashed lines) times plotted as a function of processor count from simulation data as shown in Tables 2, 3 and 4.

DL_POLY_3 code structure from functional Fortran 77 to modularised Fortran 90 was proved beneficial.

Conclusions

DL_POLY_3 is a modern molecular simulation package incorporating a wide range of functionality allowing for a vast potential range of applications.[41] Based on the forward domain decomposition concept the code exhibits excellent parallelism to thousands of processors and makes it possible to simulate systems of tens of millions of particles. In the new adaptation of DL_POLY_3 the code is couched in a modular, free-format Fortran 90 manner allowing for easier support and development as well as better compiler optimisation and completely self-contained, shedding any previous dependence on vendor libraries. It also includes new optimised memory allocation and domain decomposition mapping management, and additional functionality to help highly non-equilibrium simulations. This new adaptation excels in performance over any previous non-modularised versions of DL_POLY_3 by increased speed of calculations ($\approx$25%) and data handling loads ($\approx$50%).

Acknowledgements

This project has been financially supported by grants from the EPSRC, the Computational Science Initiative and the NERC. Daresbury Laboratory is thanked for time on various parallel computers. The advice and encouragement from the HPC*x* terascaling team, the *e*Minerals and the EPSRC Materials Chemistry Consortium is gratefully acknowledged.

References

1 M. R. S. Pinches, D. Tildesley and W. Smith, *Mol. Simul.*, 1991, **6**, 51.
2 R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*, McGraw-Hill, New York, 1981.
3 D. Rapaport, *Comput. Phys. Commun.*, 1991, **62**, 217.
4 M. E. Tuckerman, B. J. Berne and G. J. Martyna, *J. Chem. Phys.*, 1992, **97**, 3, 1990.
5 G. J. Martyna, M. E. Tuckerman, J. T. Douglas and M. L. Klein, *Mol. Phys.*, 1996, **87**, 5, 1117.
6 H. C. Andersen, *J. Comput. Phys.*, 1983, **52**, 24.
7 U. Essmann, L. Perera, M. L. Berkowtz, T. Darden, H. Lee and L. G. Pedersen, *J. Chem. Phys.*, 1995, **103**, 8577.
8 D. E. Shaw, *J. Comput. Chem.*, 2005, **26**, 13, 1318.
9 X. L. Cao and Z. Y. Mo, *Lect. Notes Comput. Sci.*, 2004, **3358**, 757.
10 I. T. Todorov and W. Smith, *Philos. Trans. R. Soc. London, Ser. A*, 2004, **362**, 1835.
11 I. J. Bush, *The Daresbury Advanced Fourier Transform*, Daresbury Laboratory, 1999.
12 I. J. Bush, I. T. Todorov and W. Smith, A DAFT DL_POLY Distributed Memory adaptation of the Smoothed Particle Mesh Ewald Method, unpublished work.
13 W. Smith, *Mol. Graphics*, 1987, **5**, 71.
14 W. Smith and T. R. Forester, *J. Mol. Graphics*, 1996, **14**, 136.
15 http://www.cs.wisc.edu/condor/.
16 http://www.csm.ornl.gov/pvm/.
17 http://www-unix.mcs.anl.gov/mpi/.
18 B. Mösli, *A Comparison of C++, FORTRAN 90 and Oberon-2 for Scientific Programming*, GISI 95, ed. F. Huber-Wäschle, H. Schauer and P. Widmayer, Springer, Berlin, 1995, pp. 740–748.
19 http://www.ibiblio.org/pub/languages/fortran/ch1-2.html.
20 M. D. Segall, P. J. D. Lindan, M. J. Probert, C. J. Pickard, P. J. Hasnip, S. J. Clark and M. C. Payne1, *J. Phys.: Condens. Matter*, 2002, **14**, 2717.
21 J. P. Ryckaert, G. Ciccotti and H. J. C. Berendsen, *J. Comput. Phys.*, 1977, **23**, 327.
22 M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Clarendon Press, Oxford, 2002.
23 C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.
24 W. C. Swope, H. C. Andersen, P. H. Berens and K. R. Wilson, *J. Chem. Phys.*, 1982, **76**, 63.
25 E. O. Brigham, *The Fast Fourier Transform and its Applications*, Prentice Hall, Singapore, 1988.
26 I. T. Todorov and W. Smith, *The DL_POLY_3 User Manual*, version 3.06, March 2006, http://www.cse.clrc.ac.uk/msi/software/DL_POLY/.
27 J. Tersoff, *Phys. Rev. B*, 1989, **39**, 5566.
28 P. J. D. Lindan and M. J. Gillan, *J. Phys.: Condens. Matter*, 1993, **5**, 1019.
29 S. A. Adelman and J. Doll, *J. Chem. Phys.*, 1976, **64**, 2375.
30 J. A. Izaguirre, *NATO Sci. Ser., Ser. III*, 2001, **117**, 34.
31 J. A. McCammon and S. C. Harvey, *Dynamics of Proteins and Nucleic Acids*, Cambridge University Press, Cambridge, 1987.
32 K. Trachenko, M. T. Dove, T. G. W. Geisler, I. T. Todorov and W. Smith, *J. Phys.: Condens. Matter*, 2004, **16**, 27, S2623.
33 K. Trachenko, M. T. Dove, E. K. H. Salje, I. T. Todorov, W. Smith, M. Pruneda and E. Artacho, *Mol. Simul.*, 2005, **31**, 355.
34 K. Trachenko, I. T. Todorov, M. T. Dove and W. Smith, *Phys. Rev. B*, submitted.
35 I. T. Todorov, J. A. Purton, N. L. Allan and M. T. Dove, *J. Phys.: Condens. Matter*, 2006, **18**, 2217.
36 K. Trachenko, I. T. Todorov, M. T. Dove and W. Smith, "Resistance to radiation damage: insights from massive parallel molecular dynamics simulations", writing in progress.
37 M. Berkowitz and J. A. McCammon, *Chem. Phys. Lett.*, 1982, **90**, 215.
38 C. L. Brooks, III and M. Karplus, *J. Chem. Phys.*, 1983, **79**, 6312.
39 R. Smith, *Atomic and ion collisions in solids and at surfaces*, Cambridge University Press, Cambridge, 1994.
40 D. J. Bacon and T. D. de la Rubia, *J. Nucl. Mater.*, 1994, **216**, 275.
41 W. Smith, C. W. Yong and P. M. Rodger, *Mol. Simul.*, 2002, **28**, 5, 385.

