計算機アーキテクチャ 137－5
ハイパフォーマンス
コンピューティング 80－5
（2000．3．2）

# Some Optimisations of DL_POLY Molecular Dynamics Simulation Code on the Fujitsu VPP700

KHOLMIRZO KHOLMURODOV[a,+], WILLIAM SMITH[b], KENJI YASUOKA[c] AND TOSHIKAZU EBISUZAKI[a]

[a] *Computational Science Division, Advanced Computing Center, The Institute of Physical and Chemical Research (RIKEN), Hirosawa 2-1, Wako, Saitama 351-0198, Japan*
*tel: +81-48-467-9415, fax: +81-48-467-4078*
*e-mail: mirzo@atlas.riken.go.jp*

[b] *Daresbury Laboratory, Daresbury, Warrington, Cheshire, UK, WA4 4AD*

[c] *Department of Mechanical Engineering, Keio University, Yokohama 223-8522, Japan*

[+] *Permanent address: Laboratory of Computing Techniques and Automation, Joint Institute for Nuclear Research, Dubna, Moscow region, 141980, Russia*

## ABSTRACT

This work is devoted to the optimisation of the molecular dynamics (MD) simulation code DL_POLY on a vector computer. DL_POLY is a FORTRAN package of subroutines and data files, designed for the MD study of a wide range of physico-chemical and biological systems on a distributed memory parallel computer. It was written at Daresbury Laboratory by W.Smith and T.R.Forester under the auspices of the Engineering and Physical Sciences Research Council (EPSRC) in the United Kingdom for the EPSRC's Collaborative Computational Project (CCP5) for the computer simulation of condensed phases. DL_POLY is a general purpose MD simulation package, but it was not designed specifically to run on vector machines, and consequently the efficiency of the code on such machines may be enhanced. Hence, the main purpose of this reported optimisation of the DL_POLY package was to increase the performance of the code on vector computer.

# 1. Introduction

In this work we consider some optimisations of DL_POLY molecular dynamics (MD) simulation code on a vector computer. DL_POLY is a general purpose FORTRAN package, designed for the MD study of a wide range of physico-chemical and biological systems on distributed memory parallel computers [1]. It was written at Daresbury Laboratory by W.Smith and T.R.Forester under the auspices of Collaborative Computational Project (CCP5) for the computer simulation of condensed phases. The code is documented in reference [2] and the algorithms employed in the original code are described in references [3–6]. The package was not designed to run on vector machines so that the efficiency of the code on such machines could be enhanced. Hence, the main purpose of this reported optimisation of the DL_POLY package was to increase the performance of the code on vector computer. We have specifically used the Fujitsu VPP700E/128 vector machine at RIKEN and tested the tuned code on several specific benchmark systems. Optimal performance was found to require considerable re-coding. Analysis of the code showed that a significant fraction of the time (about 50%) is spent calculating the forces. Many time consuming loops in the code are not vectorisable in their original form and often the compiler was unable to recognise that certain loops are in fact vectorisable (because the array elements in the loop can be assumed to be non-recursive).

# 2. Optimisation of neighbour list generation

The optimisation work on the DL_POLY package has concentrated on the restructuring of the central "link-cells" MD algorithm for constructing of the neighbour list and improving the procedure for calculating atomic forces. It is worth noting that DL_POLY calculates the nonbonded pair interactions using a Verlet neighbour list [7, 3], which records the indices of all 'secondary' atoms within a certain radius of each 'primary' atom; the radius being the cut-off radius ($r_{cut}$) normally applied to the nonbonded potential function, plus an additional increment ($\Delta r_{cut}$). The neighbour list removes the need to scan over all pairs of atoms in the simulation at every timestep. The larger radius ($r_{cut} + \Delta r_{cut}$) means the same list can be used for several timesteps without requiring an update. The frequency at which the list must be updated depends on the thickness of the region $\Delta r_{cut}$. In DL_POLY the

following two methods are implemented to construct the neighbour list: the first is based on the Brode-Ahlrichs scheme [8] (viz., subroutine "parlst") and is used when $r_{cut}$ is large in comparison with the simulation cell; the second uses the link-cell algorithm [9, 10] (viz., subroutine "parlink") when $r_{cut}$ is relatively small. The performance of the above schemes on vector machines is quite different depending on their degree of vectorisation. In the original version the Brode-Ahlrichs scheme is relatively vectorisable. As regards the "link-cell" scheme, improving its efficiency on vector computers is a challenging task and has been the subject of many studies [11-13]. A vectorised version of the "link-cell" algorithm has been described by Rapaport [11] and practically realized in [12]. A partially vectorised "link-cell" subroutine for DL_POLY package has been proposed by S.Liem in [13]. Nevertheless, a large proportion of time (about 21 percent of CPU time) was still used in the tuned "parlink" subroutine which makes this subroutine a potential target for further enhancement [13].

The subroutines "parlink" and "parlst" in DL_POLY construct the neighbour lists which identify all potential interaction pairs according to the predefined distance criterion. These lists are updated frequently to account for the dynamic nature of the system (density fluctuations, diffusions of atoms, etc.). The poor performance of "parlink" subroutine on a vector computer was basically due to the short vector lengths of most loops. In many cases most of the execution time is consumed by "parlink" and this subroutine in its original form is substantially non-vectorisable. Hence, one of the basic steps for restructuring of previously un-vectorisable loops here has been included in the increasing of the vector lengths for central loops. To vectorise the "parlink" subroutine we have preliminarily constructed the following two new lists:

$$list1(icell, i) = i' \text{ and } list2(icell, j) = j'.$$

It should be noted that the conventional link-cell method [9, 10] goes through the following steps: (1) MD cell is divided into contiguous cubic subcells (*icell*); (2) all atoms are alloted in accordance to their coordinates to the appropriate subcell; (3) the interaction between each included atom and its neighbours in the same subcell or in one of the neighbouring subcells (26 in three dimensions) is calculated. Double counting of interactions is avoided by excluding half of the neighbouring cells from consideration. The first of the above two lists (vis. *list1*) assigns all the atoms to the appropriate

cell (numbered *icell*) in accordance with the "particle-cell" scheme, while the second list (*list2*) defines all the atoms in 13 of the cells neighbouring cell *icell*. From these two lists we can now contruct the final list

$$list(i,j) = j',$$

which contains all potential interacting pairs (i.e. assigns the atoms in "particle-particle" pairs). By defining a maximum atom number over each cell we have increased the vector length in the main loop of "parlink": now in the main two *do* loops over subcells the loop *do* $i = 1, natms$ is inner one.

We have also tuned the "parlst" subroutine, which constructs the neighbour list in accordance with the Brode-Ahlrichs scheme, which originally was vectorisable. By introducing a new logical array, which checks the neighbour list capacity, we have reached a full vectorisation of this subroutine. A summary of the optimisation results on the "parlink" and "parlst" codes is given in the table below. The efficiency of the subroutine "parlink" which uses a significant amount of total CPU time has now been improved and the vectorisation proportion is now at 96 percent.

**Table:** The optimisations of "parlst" and "parlink" codes.

| Subroutine | ORIGINAL(Percent) | OPTIMISED(Percent) |
|---|---|---|
| "parlst.f" | 51 | 99 |
| "parlink.f" | 0.1 | 96 |

The essential modifications have been made in the forces subroutines too, while retaining all the universality possessed by the original code (e.g. the tests for 'frozen' atoms, or for *excluded atoms* for macromolecules, etc.). Originally, the forces subroutines calculated only the interactions for one specific central atom per call. These now contain the conventional double loop in each force subroutine, so all interactions can be evaluated with just one call to the relevant subroutines. The efficiency of the proposed codes has been tested on the Fujitsu VPP700/128E vector computer and for several DL_POLY benchmark systems [14]. The results agree with the original, and thus the optimised codes are verified with respect to the tested systems. In the constructing of neighbour list and calculating of atomic forces our optimised codes are significantly faster than the original one.

# References


[1] Smith, W. and Forester, T.R., 1996, *Molecular Graphics*, **14**, 136.

[2] Smith, W. and Forester, T.R., 1999, *The DL_POLY User Manual. Version 2.11*, Daresbury Laboratory.
See also www.dl.ac.uk/TCS/Software/DL_POLY.

[3] Smith, W. and Forester, T.R., 1994, *Comput. Phys. Commun.*, **79**, 52.

[4] Smith, W. and Forester, T.R., 1994, *Comput. Phys. Commun.*, **79**, 63.

[5] Forester, T.R. and Smith, W., 1994, *Molecular Simulation*, **13**, 195.

[6] Smith, W., 1992, *Comput. Phys. Commun.*, **67**, 392.

[7] Allen, M.P., and Tildesley, D.J. 1989, *Computer Simulation of Liquids.* Oxford: Clarendon Press.

[8] Brode, S., and Ahlrichs, R., 1986, *Comput. Phys. Commun.*, **42**, 51.

[9] Hockney, R. W., and Eastwood, J. W. 1981, *Computer Simulation Using Particles.* McGraw-Hill International.

[10] Heyes, D. M., and Smith, W. CCP5 Quarterly 26, 68 (1987).

[11] Rapaport, D. C., 1988 *Comp. Phys. Rep.*, 9, 1.

[12] Grest, G. S., Dunweg, B., and Kremer, K., private communication.; Kremer, K., Grest, G. S., and Carmesin, I. 1988, *Phys. Rev. Lett.*, 61, 566.

[13] Liem, S., *UMIST-FECIT Collaboration Report*, private communication (1998).

[14] Kholmurodov, K.; Smith, W.; Yasuoka, K; Ebisuzaki, T., 2000, *Comput. Phys. Commun.*, **124**, 1 (to be appear).