

DL_POLY_3 I/O Analysis, Alternatives and Future Strategies

Ilian T. Todorov, Ian J. Bush

STFC Daresbury Laboratory

Daresbury

Cheshire

WA4 4AD

Abstract

We outline the problems associated with I/O when performing large classical Molecular Dynamics runs, and show that it is necessary to use parallel I/O methods when studying large systems.

This is a Technical Report from the HPCx Consortium.

Report available from <http://www.hpcx.ac.uk/research/publications/HPCxTRyynn.pdf>

© HPCx UoE Ltd 2003

Neither HPCx UoE Ltd nor its members separately accept any responsibility for loss or damage arising from the use of information contained in any of their reports or in any communication about their tests or investigations.

1	<i>Introduction</i>	3
2	<i>I/O in DL_POLY</i>	3
2.1	Serial I/O in DL_POLY_3	4
2.2	Parallel I/O in DL_POLY_3	4
3	<i>Results</i>	5
4	<i>Discussion</i>	6

1 Introduction

DL_POLY_3 is a general purpose molecular dynamics (MD) package developed by I.T. Todorov and W. Smith at STFC Daresbury Laboratory to support researchers in the UK academic community¹. This software is designed to address the demand for large scale MD simulations on multi-processor platforms, although it is also available in serial mode. DL_POLY_3 is fully self-contained and written in Fortran 95 in a modularised manner with communications handled by MPI. The standards conformance of the code has been very rigorously checked using the NAGWare95 and FORCHECK95 analysis tools, so guaranteeing exceptional portability. Parallelisation is achieved by equi-spatial domain decomposition distribution which guarantees excellent load balancing and full memory distribution provided the system's particle density is fairly uniform across space². This parallelisation strategy results in mostly point to point communication with very few global operations, and excellent scaling; one might compare it with the halo exchange algorithms employed in computational fluid dynamics.

However, parallelisation of the computation is only part of the story. For DL_POLY_3 to be an effective tool for researchers all parts of the calculation must scale, and this includes the input and output (I/O) stages of the code. Historically, this has most often been performed in an essentially serial manner, not only in DL_POLY_3 but also many other large scale packages. However, with the scale of calculations now possible it is clear that this approach will not scale to the next generation of machines as the time for I/O is becoming prohibitive. In this short technical report we shall describe how DL_POLY_3 has in the past performed I/O, what problems this has resulted in, and a very first effort to address the problems.

2 I/O in DL_POLY

The main I/O in DL_POLY_3 is, as is the case for all classical molecular dynamics codes, reading and writing *configurations*. These are simply lists of the coordinates, velocities and forces acting on the particles that comprise the system. In DL_POLY_3 this has traditionally been performed using formatted I/O for portability; while the MD run itself may be done on the supercomputer the analysis of the results is often done on a workstation at home.

While this is very portable there are a couple of potential problems:

1. Formatted I/O is not very efficient
2. For large systems the files can get very big.

It is the second point that is causing us to re-evaluate our I/O strategy. With top end machines like HPCx, now capable of performing classical MD simulations in millions, or in some cases even billions, of atoms the time taken to write these large files is now beginning to impact on the amount of science that users of DL_POLY_3 can perform.

As a matter of fact, it is not the reading of configurations that is the issue. That is typically done only once to define the initial state of the system. It is the writing of

configurations. Not only need this be done for the final state of the system, but also many times during the simulation of system so that the time evolution can be studied. Thus it is the writing of configurations that need be parallelised, at least initially. In the remainder of this section we shall describe how the writing of configurations has been performed historically, and also our new parallel method.

2.1 Serial I/O in DL_POLY 3

Historically, the writing of configurations has been done very simply in DL_POLY_3, using a master slave method. One processor, the master, receives in turn the coordinates, velocities and forces held by the other processors and writes them to file. While simple, portable and robust this strategy has one obvious drawback, it is inherently serial and so may impact scalability through simple Amdahl's Law effects. For instance, on a large Blue Gene system we have observed one run where while a timestep in the MD took around 0.5s to perform, the dumping of a configuration took 450 seconds. Since a configuration is typically dumped every 1,000-10,000 timesteps this is not a very satisfactory situation! We shall call this the *Swrite* algorithm.

Reading of the initial configuration is the reverse of the above. The master reads chunks of the initial configuration and sends them in turn to the slave processors. As input is not so much of an issue at present, all results given here for reading will use this algorithm.

2.2 Parallel I/O in DL_POLY 3

The obvious problem with the *Swrite* algorithm is that only one processor ever performs the I/O, and thus all the other processors must wait on it. As an alternative we developed a very simple *Pwrite* algorithm where all processors participate in writing to the file.

Given the simple and regular format of the configuration file it is very simple to calculate where the data for a given atom needs to be written; it depends solely on the global index of that atom. Further, given that each item of the data consists of nine real numbers, the components of the position, velocity and force, all the record lengths in the file will be the same. Thus it is very simple to use Fortran direct access files and have each processor writing to the appropriate records for the atoms it holds. So in our *Pwrite* algorithm all the processors write to the file at once, as each processor can calculate where in the direct access file the data need be written.

While this algorithm is simple to implement, it is not robust and will not work on some computers, for instance the Cray T3X series. The reason for this is that it does not conform to the Fortran standard, which essentially assumes that only one process will ever be accessing a file at once. Given the emphasis on standard conformance previously in DL_POLY_3 this is unacceptable for general release. However, it is "good enough" to test the impact of parallel I/O, and much simpler than a full MPI I/O or netCDF implementation.

3 Results

To test the I/O the system we used was an oxygen deficient pyrochlore $Gd_2Zr_2O_7$ (zirconite) with a size of 3,773,000 particles, which corresponds to 1.1 GB configuration dump file. It is worth mentioning that no machine was available for exclusive use while benchmarking which could have contributed to the fluctuations of the observed times at low processor counts.

CPUs	IBM BG/L					IBM P575				
	PWrite		SWrite		Timestep	PWrite		SWrite		Timestep
32	21.5	52.5	228.7	4.9	28.22	11.3	99.7	98.3	11.5	12.02
64	20.8	54.2	244.5	4.6	13.69	9.7	116.1	118.3	9.5	5.95
128	16.6	67.9	242.1	4.7	7.23	9.1	123.7	153.5	7.4	3.03
256	18.4	61.2	238.9	4.7	3.86	13.1	86.0	120.0	9.4	1.58
512	22.5	50.1	248.9	4.5	2.03	13.3	84.7	134.1	8.4	0.87
1024	39.5	28.5	252.2	4.5	1.16	17.1	65.8	148.3	7.6	0.58
2048	58.8	19.2	253.0	4.5	0.77					

Table 1 – Times/s for I/O Transactions (in s) and I/O Bandwidth (in **MByte/s**)

Table 1 show the results for two IBM systems, the Blue Gene system at Daresbury and HPCx. The times for writing a single configuration are given, as is the time to perform a single timestep (in seconds). The values in bold are estimates of the I/O Bandwidth in MByte per second. Both systems have a GPFS file system.

It can be seen that the parallel writing algorithm is markedly superior to performing I/O in serial. Improvements by an order of magnitude can be obtained, and it is clear that if regular writing of configurations is required when studying this system that the parallel strategy will markedly improve the scaling of the whole code, even though the I/O is not scaling especially well itself.

CPUs	Cray XT3 Single Core					Cray XT3 Dual Core				
	PWrite		SWrite		Timestep	PWrite		SWrite		Timestep
64	16.5	68.2	32400	0.0	2.93	15.0	75.1	32400	0.0	3.62
128	16.3	68.2	32400	0.0	1.61	12.5	90.1	32400	0.0	2.30
256	14.9	75.6	32400	0.0	0.87	14.8	76.1	32400	0.0	1.26
512	17.3	65.1	32400	0.0	0.48	19.6	57.4	32400	0.0	0.64
1024						11.5	97.9	32400	0.0	0.44

Table 2 – Times/s for I/O Transactions (in s) and I/O Bandwidth (in **MByte/s**)

As far as the bandwidth is concerned, it is lower than that reported in [3]. This is not too surprising. Here, we are dealing with formatted I/O, and further the lengths of the individual I/O transactions are much shorter than those in [3]. It is difficult to say more, as different machines are being compared, but it does indicate that while the simple strategy adopted here has brought considerable improvement to the code, still better performance is achievable.

Table 2 contains the timings and bandwidth obtained on a Cray XT3 system. Both single and dual core runs were performed, and again exclusive use was not available.

The improvements here are dramatic, though it probably reflects more a problem in the Lustre I/O system used by the XT3 rather than the code changes we have made. Whatever the number of processors the *Write* algorithm took at least 9 hours; remember this is the time to dump **just one configuration!** On the other hand, the *Pwrite* time is comparable to that found on the IBM systems, and show similarly poor scaling. It can also be seen that the timestep calculation is appreciably faster on the XT3 compared to the IBM systems, but that the I/O is generally, but not always a little slower.

4 Discussion

It is clear from the above that for DL_POLY to be able to address larger systems we must move to a parallel I/O strategy. However it is also clear that the current strategy, while simple to implement is not optimal as it

1. is not portable
2. does not scale well
3. does not perform as well as one might hope.

To address the second point a simple method would be to use only a subset of the processors to perform the I/O, rather than all of them. For instance, on the BG/L system it is clear that 128 processors would be a sensible number to use for this particular simulation, and we suspect that similar considerations will apply whatever the form of parallel I/O is used.

For portability there are two issues, the portability of the implementation and the portability of the resulting file. The current implementation fails the first but passes the second. Unfortunately, the most common parallel I/O library, MPI IO, in practice passes the first but fails the second. This is because most current implementations of MPI IO only implement the *native* data representation which is inherently non-portable. As analysis of DL_POLY_3 runs is often performed on a different machine to that which the code originally executed on, this is not acceptable. We, therefore, intend to look at other possible solutions, such as netCDF which uses XDR encoding to ensure portability, to solve the problem outlined in this report. We hope that this will also address the performance problem, so raising the I/O bandwidth close to the values observed in [3], where bandwidths close to 1 GB/s range were recorded on the phase 2 HPCx machine.

-
- [1] Todorov IT and Smith W, 2004, *Phil Trans R Soc Lond, A* **362**, 1835
 - [2] M.R.S. Pinches, D. Tildesley, W. Smith, 1991, *Mol Simulation*, **6**, 51
 - [3] Michael Holden, Elena Breitmoser, Joachim Hein HPCxTR0504 "I/O Performance on the HPCx Phase 2 System"