DL_POLY_4 and Xeon Phi: Lessons Learnt

Alin Marin Elena[1], Christian Lalanne[3], Victor Gamayunov[2], Gilles Civario[3], Michael Lysaght[3], and Ilian Todorov[1]

[1]Scientific Computing Department, STFC, Daresbury Laboratory, UK
[2]Software and Services Group, Intel Corporation, UK
[3]Irish Centre for High-End Computing, Dublin, Ireland

I. Introduction

Molecular dynamics techniques grew rapidly in the last twenty years. The growth was fuelled by development of new scalable mathematical algorithms, availability of powerful hardware and better availability of ready to use software packages. DL_POLY is one of these packages, widely adopted by the computational physics and material science communities.

DL_POLY started its life in 1992 at Daresbury Laboratory, now part of Science & Technology Facilities Council in United Kingdom, with a first public release in 1993. The main developers for the current version are W Smith and IT Todorov. DL_POLY is a general classical molecular dynamics code and was used to simulate macro molecules (both biological and synthetic), complex fluids, materials and ionic liquids. DL_POLY also plays an important role as sandbox for both development of new methods and algorithms for molecular dynamics and testing of emerging hardware technologies[1] and [2]. The core code is written in Fortran 95/2003 standards and optimised for distributed systems using domain decomposition, also OpenMP and CUDA ports exist as contributions to DL_-POLY but not part of the official distribution. DL_POLY is free of use for academics pursuing non-commercial research and available for licensing for the rest.

The Intel Xeon Phi co-processor is a novel accelerator technology that provides few attractive features as: many cores, 60 cores with 240 hardware threads for the mid model, low power consumption, the same set of instructions as an Intel CPU, supports popular and standardised programming models as MPI and OpenMP and a theoretical peak of 1 TFlops in double precision.

In this communication we present the progress made in porting and optimising DL_POLY to Xeon Phi co-processor. The rest of the paper is organised as follows: a short introduction to the methodology used for port and optimisation, OpenMP implementation results in section III-A, synchronous offload ones in section III-B, MPI symmetric running mode in section III-C.

II. Methodology

DL_POLY is extensively parallelised and optimised for distributed systems using MPI 2.0[3].

Being a general purpose tool one can simulate a large variety of systems. However for simplicity we decided to concentrate on systems of interest for two communities: biophysics and material science.

The system of interest for biophysics community chosen is case 7 in DL_POLY testing suite, we shall refer to it from now on as Gramidicin and it consists of 8 Gramidicin A solvated in water. The system has a total of 99120 atoms. The simulation is carried out in an NPT ensemble at room temperature. An equivalent larger system with 792960 is provided, case 8.

The system of interest for material science community is case 31 in DL_POLY testing suite, we shall refer to it fom now on as Iron and it consists of an Iron BCC with a constant lattice of 2.8665 Å with 31250 atoms. The simulation as previously is carried out in an NPT ensemble at room temperature. A larger system with 250000 atoms is provided by test case 32.

With the exception of the large Gramidicin system all the systems comfortably fit in the 8 GiB memory of the Xeon Phi we have available.

All the simulations reported here were carried out on ICHEC's system fionn[4]. The Xeon Phi partition consists of 16 nodes each Host with two sockets Ivy Bridge Xeon, E2660-v2[5], and two Xeon Phi cards, 5110P[6]. The memory available on the host is 64 GiB and for each card 8 GiB.

One indicator of the code performance traditionally used by molecular dynamics communities is average time per molecular dynamics integration step. We will call it *time per step(TPS)*. This metric will be employed by us to assess the performance of DL_POLY, this can be easily transformed into other customary metrics, *e.g.* simulation time per 24 hours of walltime, as employed by GROMACS[7] and NAMD[8].

Preliminary scalability curves for the two systems of interest were carried out, see fig. 1. We used the latest stable version of DL_POLY 4.05.01, with Intel Fortran compiler, version 14.0.1, and Intel MPI implementation, version 4.1.2.040, current stable versions at time of computing. Very good scaling is shown for both systems, Gramidicin in upper right panel, 68% relative efficiency, and Iron in the lower right panel, 73% relative efficiency,

in the case of the Host for 20 MPI processes, for the time step. However, when run on the Xeon Phi co-processor in native mode the relative efficiency for both systems is at around 50% up to 60 processes and then suddenly degrades when more MPI processes are added - oversubscription of cores. For this we used pure MPI mode for both host and Xeon Phi. We considered only the top 3 functional segments of code for each system: two body forces (TBF), linked cell lists (LCL) and shake for gramidicin, and TBF, LCL and metal local density (MLD) for Iron.

The three avenues left to investigate are OpenMP native, offload and MPI symmetric modes. In the results sections of this communication we will present results of our investigations.

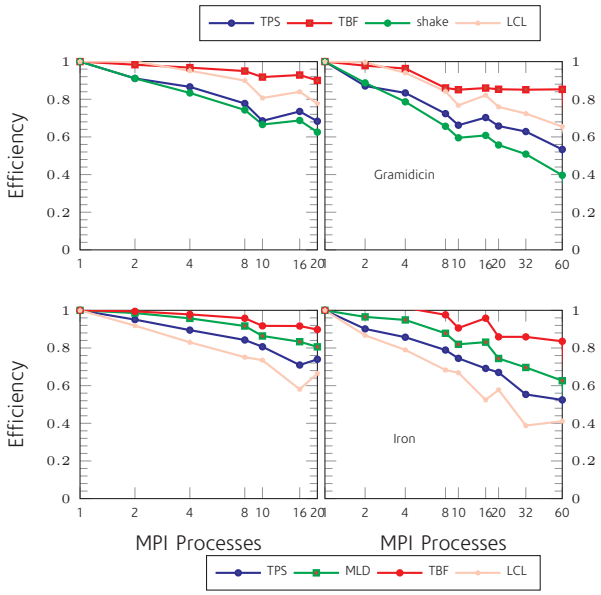

Figure 1. MPI Scalability for two systems Gramidicin (upper panels) and Iron(lower panels). Host scalability is reported in the left hand side and Xeon Phi in the right hand side.

III. Results and Discussion

A. OpenMP Parallelisation

In order to reduce the MPI communication which is a slowing factor on the Xeon Phi we decided to investigate the advantages of shared memory strategies. One popular approach for shared memory programming within the community is the OpenMP model [9]. OpenMP is a mature, standard programming model supported by all major compilers and is currently at version 4.0. OpenMP supports offloading mechanism for the Xeon Phi which makes it an attractive proposal for us.

Starting from the time profiles of the pure MPI code we extracted the three most expensive code segments for each system for up to 20 MPI processes. Combined evaluation of TBF, computing the LCL and applying Shake constraints are the most expensive code segments for Gramidicin, from 79% in the case of one MPI process to 68% for 20 MPI processes. In the case of Iron the most expensive segments of code are, evaluation of TBF, computation of LCL and the computation of the MLD, from 95% for one MPI process to 90% for 20 MPI processes. If one increases the MPI process count one can expect a dramatic change of the profile for code segments which contain MPI collectives dominating, *e.g.* fft and shake.

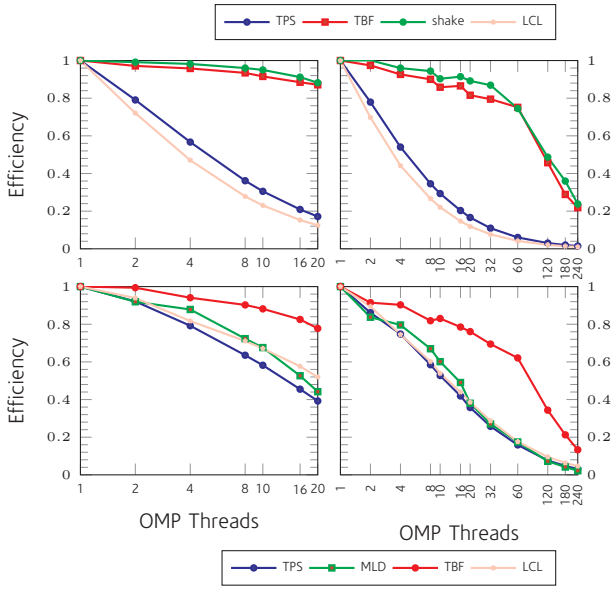

Figure 2. OpenMP scalability with one MPI process for two systems Gramidicin (upper panels) and Iron(lower panels). Host scalability is reported in the left hand side and Xeon Phi in the right hand side.

OpenMP scalability of time per step with one MPI process is poor as shown in the blue curve of fig. 2 on both Host and Xeon Phi for both systems, this is caused mainly by the poor scalability of linked lists for Gramidicin, and, linked lists and metal local density for Iron and the rest of sequential code segments which are around 20%.

Thread affinity plays a major role in the performance on Xeon Phi. We have investigated *compact, scatter,balanced* and *explicit* affinities and we concluded that *compact*[1] provides by far the worst performance for us. The last three offer similar performances with a slight advantage for *scatter*[2], hence all the results reported in this section are obtained using *scatter* affinity. One may argue that treating equally all the threads is not right and one shall use only 60 cores and the supplementary extra hardware threads to be considered separately. Such experiment is considered in the remainder of this section, considering only the time per step for two body forces as performance metric.

[1] assigns the $n+1^{th}$ thread to a free thread context as close as possible to the thread context where the n^{th} thread was placed

[2] distributes the threads as evenly as possible across the entire system

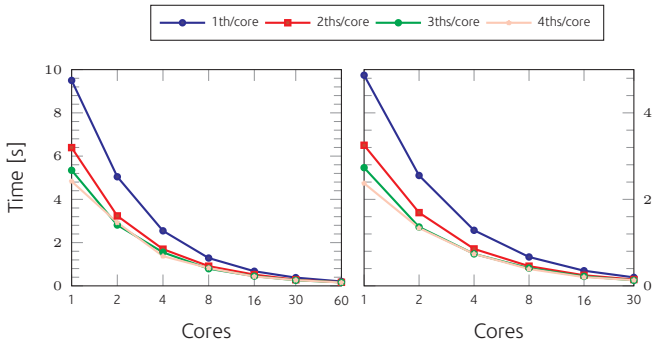

Figure 3. Gramidicin in Water simulation, 99120 atoms. Strong scalability curves when using one MPI process, left panel and two MPI processes, right panel on Xeon Phi.

Gramidicin, the small system, was run using direct evaluation of the kernel for the two body forces rather than the classical tabulated form. The strong scalability for OpenMP is presented in Fig. 3, for both one MPI process run and two MPI processes run. One shall note that each curve represents a strong scalability at constant core load with threads - 1, 2, 3 and 4 threads per core. The OpenMP process placement was done via KMP_PLACE_THREADS. For constant number of threads per core the scalability with core numbers is good for both MPI cases. It varies from around 80% efficiency when only one thread per core is used to 52% when 4 threads per core are used and one MPI process. In the case of 2 MPI processes for 4 threads per core we obtain an efficiency of 60%. Overall speedup on Intel Xeon Phi for 1 MPI is of 63 times, whereas for 2 MPI processes is 37 times, indicating a strong perfomance degradation due to poor MPI communication. We notice that for one MPI process case and one core, increasing the subcription of cores from 1 to 2 threads we gain around 48% while from 1 to 3 threads the gain is 78% and using all 4 threads on the core results in 97% gain. The conclusions for gramidicin are transferable to Iron systems too.

1) Shake: Shake is the name of the algorithm for the treatment of dynamical systems with holonomic constraints [10]. DL_POLY uses a modified version of the original algorithm as described in [11]. Shake is originally a sequential iterative algorithm, the version implemented in DL_POLY is modified to allow distribution over MPI processes. Each iteration involves synchronization of the coordinates of atoms in all the MPI processes which contain constraints. This makes the OpenMP parallelisation a challenging task.

We parallelised for each shake iteration the most intensive loops, unfortunatelly the last loop of the algorithm in which the atomic positions are corrected contains six atomic operations which cannot be removed due to the fact that the parallelisation is made over constraints and one atom can belong to more than one constraint. By profiling with vtune we discovered that the division operation is atypical expensive on the Xeon Phi hence we transformed some expressions wich contained it.

OpenMP scaling for Shake is excellent within the Host with an efficiency of 88% for 20 threads and on the Xeon Phi we achieved good scalability with a relative efficiency up to 74% with 60 threads, as shown by the green curve in the upper panels of fig. 2. Increasing further the thread count on Xeon Phi the efficiency degrades. This is due the fact that the algorithm does not benefit from the usage of the extra threads on each core. Further investigations are required in order to increase the usage of the extra threads. In terms of absolute times the pure OpenMP implementation of Shake on the Xeon Phi is 77% slower than the pure MPI version when 10 MPI processes are used and are around 3 times slower compared with a full Host.

On the Xeon Phi the OpenMP implementation with one MPI process is 66% faster than the best time for shake with pure MPI for the Gramidicin system.

Incrementing the number of MPI processes decreases the OpenMP efficiency of Shake, this is due to the MPI synchronization of data which needs to be done every iterative step of Shake.

2) Two Body Forces: Two body forces (TBF) is the code segment that performs the computation of interatomic forces between a pair of atoms. Depending on the system that it is computed, Iron or Gramidicin, different contributions are evaluated. If the Iron system is executed, metal components of energies and forces are calculated additionally to the computation of Van der Waals and Coulombic forces and energies.

Two body forces is executed for every time step of the simulation and it was originally implemented sequentially for every MPI process. The sequential implementation basically consists of a loop over all the atoms that belong to a MPI process, this loop was parallelised with OpenMP obtaining the results shown by the red curve in fig. 2, this parallelisation forced us to introduce one more dimension to several of the arrays used in the code *e.g.* forces and distance differences, in this way avoiding atomic operations for force updates. Although we increased the memory footprint of the code, with these changes, we were able to parallelise the code improving its performance.

To improve the performance of our first parallel OpenMP implementation we applied two optimisations. The first one was based on [12] and consisted of avoiding OpenMP reductions over multi-dimensional arrays, writing our own reductions for arrays allowed us to improve the performance of the parallel loop. The second improvement was to replace all the Fortran allocatable arrays involved in the computation of two body forces by automatic arrays.

All the procedures called from the parallel region needed an interface, which was implemented by creating Fortran modules for each file that needed them.

OpenMP parallelisation achieved very good scalability for Gramidicin, with an efficiency of 87% for 20 threads

on the Host and 75% with 60 threads on the Xeon Phi. For Iron the OpenMP parallelisation achieved 78% efficiency with 20 threads on the Host and 62% efficiency with 60 threads on the Xeon Phi, this is shown by the red curve in fig. 2. For the Xeon Phi, with a thread count over 60 threads the efficiency shows a steep decrease reaching 45% for Gramidicin and 34% for Iron. The differences in efficiency between Gramidicin and Iron and the poor efficiency of two body forces on the Xeon Phi after 60 threads on the Xeon Phi need further investigations.

In terms of absolute times, best cases of the OpenMP parallelisation on the Xeon Phi are 52% slower for Gramidicin (30 MPI processes and 8 threads) and 97% for Iron (60 MPI processes and 4 threads) than the case when 10 MPI processes are used on the Host with the original version of the code. The OpenMP parallelisation using 20 threads achieved a speedup of 4% over the MPI base implementation for Gramidicin and its performance decreases 18% for Iron using 20 MPI processes on the Host. Comparing the best executions on the Intel Xeon Phi, the OpenMP parallelisations using 120 threads achieved similar results than the MPI only implementation of the code, the OpenMP version of the code was 22% for Iron and 15% for Gramidicin slower than the MPI original code using 120 MPI processes.

3) Linked Lists: Verlet neighbour lists, or as they are commonly known linked lists, are a common method in molecular dynamics codes to reduce drastically the number of force evaluations during the time propagation[13]. DL_POLY implements a modern version of these lists[14–16]. The main idea is the partitioning of the entire simulation space into cells such that in order to determine all the atoms with which one atom interacts one needs to check only the neighbouring cells. The version implemented in DL_POLY is efficient with respect to the MPI processes, see fig. 1 pink curve, over 70% for both Gramidicin and Iron on Host. This picture is not replicated in the case of the Xeon Phi for Iron where efficiency is as low as 40%.

OpenMP parallelisation of the most time consuming loop involved an extensive refactoring of the original implementation. Instead of looping over all the cells and then over all the atoms in a particular cell we loop over all the atoms. In this way we increas the amount of work that can be done in parallel. The results, presented in fig. 2, show a good scalability for small thread counts in the case of Iron on both Host and Xeon Phi. The poor scalability for high thread counts can be attributed to the multiple branching within the OpenMP region and the role of the sequential code from this segment.

4) Metal Forces: This code segment is executed for every time step of the simulation(only in the case of the Iron system) and it is used to compute local densities in metals, this is performed using the verlet neighbour list and Finnis-Sinclair type potential [17]. These local densities are computed sequentially using two loops over the atoms of every MPI process.

The parallelisation of this algorithm was performed using OpenMP over these two loops, the results of this parallelisation are shown by the green curve on the bottom panels of fig. 2. This parallelisation does not achieve good efficiency, with 44% with 20 threads on the Host and 17% with 60 threads on the Xeon Phi. The main reasons for this poor performance is an MPI process synchronization segment at the end of the computation of metal densities and some sequential code still present in the computation.

We noticed during the parallelisation of this code that some calculations performed are also computed in two body forces *eg.* calculation of the differences between atom positions. To optimise this, we stored these values to not perform the same computation twice. We did not obtain good results with this optimisation, due to the increase of memory foot print that the implementation of this optimisation introduced, also the code added in two body forces to allow this optimisation degraded the performance in the case of the Gramidicin system.

B. Synchronous/Asynchronous Offload Implementation

Based on our previous results in Native Mode it is obvious that our OpenMP implementation for MD does not make efficient use of all the threads available on the Xeon Phi, one way to tackle this issue is to use the co-processor in offload mode. From the code segments identified earlier the best candidates for offloading is two body forces due to its good OpenMP scaling on Xeon Phi. Shake is the only code segment which due to the MPI synchronization at each iteration is not suitable for offloading, Linked Lists and metal local density code segments due to poor OpenMP scalability on the Xeon Phi can be challenging, for the rest of this section we will concentrate on the offload of two body forces.

One can offload data to the co-processor in two ways, using OpenMP 4.0 or Intel LEO (Language Extentions for Offload). As the code already uses OpenMP for parallelisation is a natural option to use OpenMP also for offloading. Unfortunately after implementing the offload code with OpenMP we discovered an issue with implementation of OpenMP tasks and offload. In order to avoid this we changed the offload implementation to Intel LEO. Both OpenMP and LEO implementations for the synchronous offloading achieved the same performance results.

One has to upload for each computation to the card: positions, the list of neighbours for each atom and all data related to the potential and download from the card to the host the contribution to forces and other statistical quantities.

In the Gramidicin case when running 1 MPI the data transfered to the Xeon Phi can be as big as 333.54MB and the data transfered from the Xeon Phi to the Host [3] is 11.08MB for every time step of the simulation, the

[3]Intel(R) Xeon(R) CPU E5-2660 v2, 2.20GHz, 64GiB.

time spent doing this is 0.24 seconds. By far the biggest chunk of data is associated with linked lists.

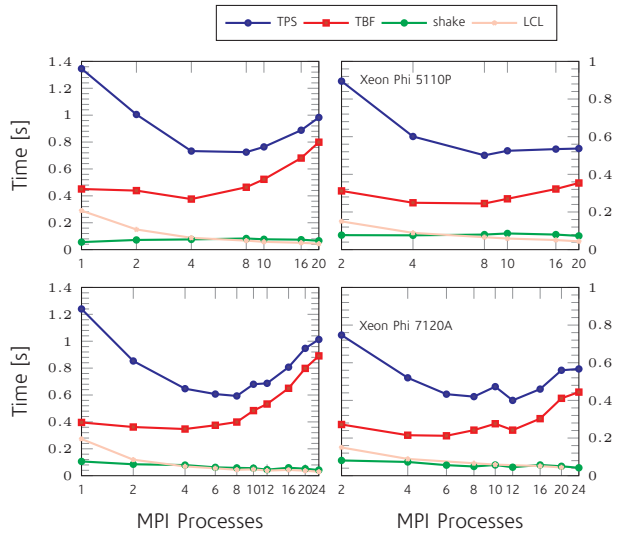

Figure 4. Best results for DL_POLY with synchronous offloading for two body forces in the case of Gramidicin, upper panels show data for the 5110P model card lower panels for the 7120A model card vs MPI process count. For each case both host and Xeon Phi are saturated with OpenMP threads.

We carried a systematic study of the offload performance when multiple MPI processes offload the code to one or two cards for two body forces the results are presented in fig. 4. Upper panels of fig. 4 present results for offloading to the 5110P model [4] of the Intel Xeon Phi and lower panels show the results for the model 7120A [5] of the Intel Xeon Phi. The results presented in fig. 4 were obtained by running DL_POLY with a varying number of MPI processes, when two cards are used for offloading first half of MPI processes offload to card number 0 and the second half to card number 1. For each case when the number of MPI processes was less than the number of cores of the Host where possible we saturated each MPI process with OpenMP threads. For each MPI process the card was equally partitioned and saturated with OpenMP threads[6]. The best results for each MPI process count is selected and presented.

When we offload to one card, left hand side panel, or two cards, right hand side panel, using more than 4 MPI processes offloading to the same card slows down two body forces. This is due to the fact that data transfers dominate over computations, hence the need of using a reduced number of MPI processes on Host and an increase of OpenMP threads on the Xeon Phi.

The best performace for time per step for the 5110P was 0.5 s and 0.42 s for the 7120A card(8 MPI processes)

[4] 8gb, stepping B1, 1052.630 ghz, 60 cores.

[5] 16gb, C0 stepping, 1.238 ghz, 61 cores, C0QS-7120 (pre-prod 7120A) ivb-ep 12c 2.5ghz, 128gb.

[6] for host by saturation we mean MPI processesxOpenMP Threads=20, for Xeon Phi the products is 240.

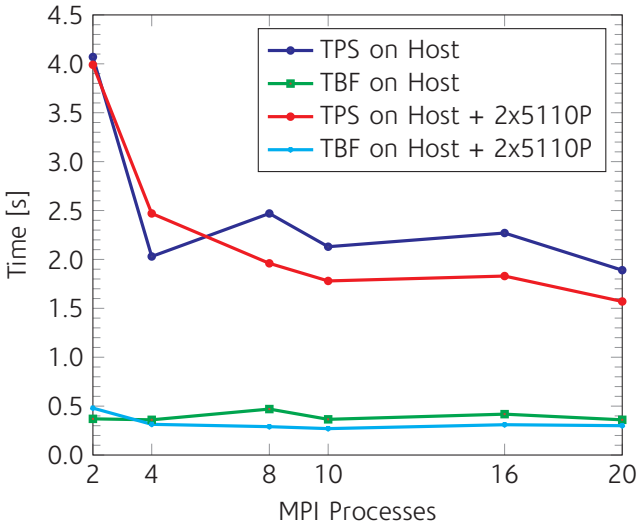

Figure 5. Two body forces and Time per step performance for asynchronous offload mode. When more than 1 MPI process offloaded to one card, we partition the card in equal shares between them.

which are slower than 0.3 s on the equivalent number of MPI processes on the Host. These results are encouraging as is possible to further improve them by using asynchronous computations and data transfers, as we used in a previous work[18].

Asynchronous transfer of both data and computation was implemented in code using Intel LEO. As strategy for async offlod we choose to split the forces that need to be evaluated between host and Xeon Phi. Results are presented in fig. 5. We find that when four MPI processes offload to a card, the speedup in two body forces evaluation is of 62% which translates in an overall speed up for time per step of 26%. Other good performance numbers are seen for five MPI processes offloading to a card, 35% for TBF and 19% for TPS, when 8 MPI processes offload to a card the speedup for TBF is 34% while for TPS is 24%. One may notice that the final speedup for TPS is not coming only from TBF speedup, but from code refactoring done in order to accommodate the asynchronous offload, mainly memory layout changes. One shall be aware of these unintended side effects, which proved to be lucky in our case.

C. MPI Native and Symmetric

Another way in which one can exploit the Intel Xeon Phi is MPI symmetric mode. Unfortunately the results we obtained by executing DL_POLY in symmetric mode were on the same level of performance with MPI native results. This was expected because of the nature of the calculation, for each time step a MPI collective synchronization is performed hence the slowest component dominates the timings. At the moment data distribution over MPI processes is homogeneous, an heterogeneous distribution should be implemented if one wants to exploit this execution model.

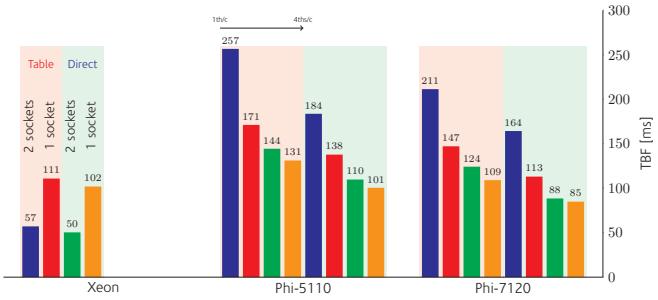

Figure 6. Two Body Forces best performance on Xeon (one and two sockets) and Xeon Phi (5110 and 7120), the lower the better. Red area indicates that a tabulated method was used for potential, while for green areas direct evaluation of the potential was used. For Xeon we used 10 MPI processes for one socket, 20 MPI processes for two sockets case. In the case of the Xeon Phi the best performance is used by using 30 MPI processes and for each of them two cores fully subscribed with theads. For Xeon Phi one can see how performance improves by using extra threads on each core.

Conclusions

We successfully ported DL_POLY to Xeon Phi and showed that without an intensive effort into optimisation it is difficult to fully take advantage of the Xeon Phi co-processor. Moderate success can be claimed for the OpenMP implementation were we showed that for intensive routines with the right type of parallelism in native mode we can be as fast as the host, *eg*. Shake. MPI symmetric seems to be a no go area as long as the homogenous decomposition of data is maintained. By far the most promising is the offload mode, where synchronous offloading showed encouraging results. Asynchronous offload concentrated on mixing asynchronous computation and data transfers and showed promising speed-up. However sources of vectorisation and efficient usage of many cores are still challenges that need to be solved in the same time to take advantage of the Xeon Phi potential.

The main result is improvements on both host and Xeon Phi for two body forces resulting from changing the way in which the two body forces are evaluated. The gains for the case of Gramidicin are of 11% on host and of 24% on Xeon Phi for two body forces, see Fig 6. Since two body forces represents only around one thirds of the total time for a time step, these improvements do not translate into big improvements overall.

Acknowledgments

Authors would like to thank to Intel for the finacial support via IPCC@Dublin and Marco Grossi form ICHEC for keeping the Xeon Phi cluster stable and alive.

References

[1] DL_POLY_4. -. *DL_POLY*, http://ccpforge.cse.rl.ac.uk/gf/project/dl-poly/, 1991-2016. 1

[2] Ilian T. Todorov, William Smith, Kostya Trachenko, and Martin T. Dove. DL_POLY_3: new dimensions in molecular dynamics simulations via massive parallelism. *J. Mater. Chem.*, 16:1911–1918, 2006. 1

[3] Message P Forum. MPI: A Message-Passing Interface Standard. *http://www.mpi-forum.org/docs/docs.html*, -, 1994. 1

[4] Irish Centre for High-End Computing. Fionn. *https://www.ichec.ie/infrastructure/*, -, 2013. 1

[5] Intel. Intel Xeon Processor E5-2660 v2. *http://goo.gl/QuzA4u*, -, 2013. 1

[6] Intel. Intel Xeon Phi Coprocessor 5110P. *http://goo.gl/BUQOyI*, -, 2013. 1

[7] H.J.C. Berendsen, D. van der Spoel, and R. van Drunen. GROMACS: A message-passing parallel molecular dynamics implementation. *Computer Physics Communications*, 91(1–3):43 – 56, 1995. 1

[8] Laxmikant Kale, Robert Skeel, Milind Bhandarkar, Robert Brunner, Attila Gursoy, Neal Krawetz, James Phillips, Aritomo Shinozaki, Krishnan Varadarajan, and Klaus Schulten. NAMD2: Greater Scalability for Parallel Molecular Dynamics. *Journal of Computational Physics*, 151(1):283 – 312, 1999. 1

[9] OpenMP Architecture Review Board. OpenMP application program interface version 4.0, 2013. 2

[10] J Ryckaert, G Ciccotti, and H Berendsen. Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes. *Journal of Computational Physics*, 23(3):327–341, 1977. 3

[11] W Smith and T Forester. Parallel macromolecular simulations and the replicated data strategy II. The RD-SHAKE algorithm. *Computer Physics Communications*, 79(1):63–77, 1994. 3

[12] Simone Meloni, Alessandro Federico, and Mario Rosati. Reduction on arrays: comparison of performances among different algorithms. *EWOMP'03*, 2003. 3

[13] Loup Verlet. Computer "experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Phys. Rev.*, 159:98–103, Jul 1967. 4

[14] M. R. S. Pinches, D. J. Tildesley, and W. Smith. Large scale molecular dynamics on parallel computers using the link-cell algorithm. *Molecular Simulation*, 6(1-3):51–87, 1991. 4

[15] W Smith and T Forester. Parallel macromolecular simulations and the replicated data strategy I. The computation of atomic forces. *Computer Physics Communications*, 79(1):52–62, 1994. 4

[16] R.W. Hockney and J.W. Eastwood. *Computer Simulation Using Particles*. Taylor & Francis, 2010. 4

[17] MW Finnis and JE Sinclair. A simple empirical N-body potential for transition metals. *Philosophical Magazine A*, 50(1):45–55, 1984. 4

[18] Alin Marin Elena and Ivan Rungger. Enabling Smeagol on Xeon Phi: Lessons Learned. *PRACE White Paper*, 2014. 5