

# Porting the DL\_POLY molecular dynamics package to GPGPUs

C. Kartsakalis and R. Nestor  
Irish Centre for High-End  
Computing,  
Grand Canal Quay,  
Dublin 2, Ireland

W. Smith and I.T. Todorov  
STFC Daresbury Laboratory,  
Warrington,  
Cheshire,  
UK WA4 4AD

## Introduction

DL\_POLY [1] is a well-known molecular dynamics simulation code developed primarily by the Science and Technology Facilities Council in Daresbury Laboratory, UK. The DL\_POLY application has already been parallelised using a domain decomposition approach, employing the Message Passing Interface (MPI) library for inter-process communication. The existing parallel implementation of DL\_POLY is highly efficient and scales well to many thousands of CPU cores.

Because of the need to perform increasingly realistic simulations, General-Purpose Graphics Processing Units (GPGPUs) are becoming increasingly popular in computational science. GPGPUs are massively parallel, with thousands of threads performing simultaneous computations. Therefore, they offer the potential to dramatically speed up the execution of numerically intensive applications. In this work, a port of DL\_POLY has been developed [2] for GPGPUs using the NVIDIA CUDA framework. This work was undertaken by the Irish Centre for High-End Computing (ICHEC) in collaboration with Daresbury Laboratory.

## Methodology

On assessing the original DL\_POLY code, the components listed in Table 1 were identified as the most computationally intensive parts of the application. These components were targeted for porting to the NVIDIA CUDA framework. These components were accelerated using a *single-client* approach, in which each host MPI process binds to a GPGPU device and offloads a proportion of its computations to the GPGPU.

In the majority of GPGPU-enabled clusters, the number of CPU cores is typically greater than the number of attached GPGPUs. However, to avoid GPGPU over-subscription, only one MPI process is run per attached GPGPU, thus resulting in idle CPU cores. To avoid this problem, most of the DL\_POLY functions ported to CUDA have also been parallelised using OpenMP. The computational effort is then dynamically distributed between the OpenMP threads and the GPGPU. The advantage of this approach is that it is not necessary to tune the code on a per-machine basis. The dynamic load-balancing is performed for each accelerated component on a per-iteration basis. With the exception of the Constraints Shake VV component, all of the accelerated components listed in Table 1 were also accelerated using OpenMP

Table 1: CUDA-accelerated DL\_POLY components

Component	Purpose
Two-body forces	Compute inter-atomic forces
Link-cell pairs	Construct atom neighbour lists
Ewald SPME forces	Compute Coulombic and force terms in periodic systems
Constraints Shake VV	Apply bond constraints between atoms

and used the dynamic load-balancing strategy described above to balance the load between the GPGPU and the OpenMP threads.

## Results and Discussion

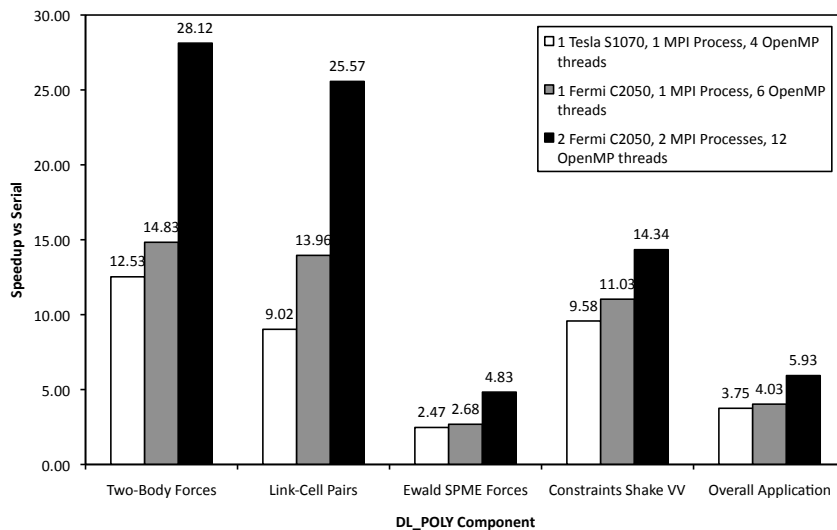
For a suite of test problems supplied with the DL\_POLY package it has been verified that the developed CUDA port reproduces the results obtained using the standard CPU version in double precision mode. The suite of test problems is also used to assess the performance of the developed CUDA port by measuring the speedup values for both each individual CUDA-accelerated component and the entire application.

Figure 1 shows the measured speedup values for two test problems. Speedup values are given for each accelerated component and the entire application for three different hardware and runtime configurations. The first configuration consists of one NVIDIA Tesla S1070 GPGPU attached to a quad-core Intel Xeon X5560 CPU with 4 OpenMP threads. The second configuration consists of one NVIDIA Fermi C2050 attached to a hex-core Intel Xeon X5650 CPU with 6 OpenMP threads. The third configuration is identical to the second configuration, except that two Fermi GPGPUs are bound to two MPI processes with a total of 12 OpenMP threads. All speedup figures are reported for the double precision version of the application.

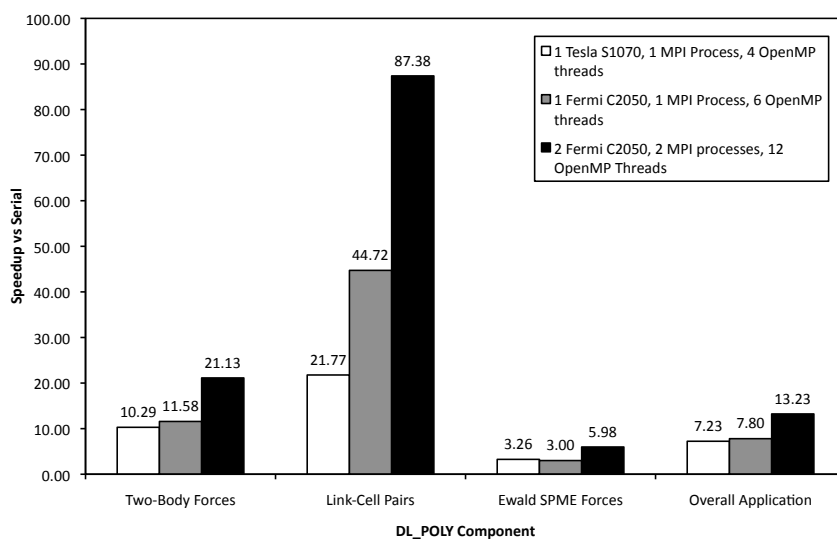
The maximum component speedup is observed on the NVIDIA Fermi architecture for test case 2, for which a speedup of about 45 is achieved for the Link-Cell Pairs component running on a single GPGPU with 6 OpenMP threads for the host computation. Furthermore, the developed port scales well to a twin GPGPU configuration, for which nearly linear scaling is achieved on a per-component basis for both test cases. These results demonstrate that excellent speedup values are achievable on a per-component basis, although the observed speedup is dependent on the physics of the problem.

## References

- [1] I.T. Todorov, W. Smith, K. Trachenko and M.T. Dove, "DL\_POLY\_3: new dimensions in molecular dynamics simulations via massive parallelism", *J. Mater. Chem.*, 16, pp. 1611-1618, 2006
- [2] C. Kartsaklis, I.T. Todorov, W. Smith, "DL\_POLY 3: Hybrid CUDA/OpenMP porting of the non-bonded force-field for two-body systems.", Symposium on Chemical Computations on GPGPUs, 240th ACS National Meeting and Exposition, Boston, 2010.



(a) DPMC in water, 413896 atoms (test case 4)



(b) Sodium chloride with Ewald sum, 216000 ions (test case 2)

Figure 1: Measured speedup versus one CPU core for several hardware and runtime configurations. All speedup values are for the double precision version of the application.