Computer Physics Communications ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

A domain specific language for performance portable molecular dynamics algorithms

William Robert Saunders [a], James Grant [b], Eike Hermann Müller [a,*]

[a] Department of Mathematical Sciences, University of Bath, Bath BA2 7AY, Bath, United Kingdom
[b] Department of Chemistry, University of Bath, Bath BA2 7AY, Bath, United Kingdom

ARTICLE INFO

Article history:
Received 12 April 2017
Received in revised form 10 November 2017
Accepted 12 November 2017
Available online xxxx

Keywords:
Molecular dynamics
Domain specific language
Performance portability
Parallel computing
GPU

ABSTRACT

Developers of Molecular Dynamics (MD) codes face significant challenges when adapting existing simulation packages to new hardware. In a continuously diversifying hardware landscape it becomes increasingly difficult for scientists to be experts both in their own domain (physics/chemistry/biology) and specialists in the low level parallelisation and optimisation of their codes. To address this challenge, we describe a "Separation of Concerns" approach for the development of parallel and optimised MD codes: the science specialist writes code at a high abstraction level in a domain specific language (DSL), which is then translated into efficient computer code by a scientific programmer. In a related context, an abstraction for the solution of partial differential equations with grid based methods has recently been implemented in the (Py)OP2 library. Inspired by this approach, we develop a Python code generation system for molecular dynamics simulations on different parallel architectures, including massively parallel distributed memory systems and GPUs. We demonstrate the efficiency of the auto-generated code by studying its performance and scalability on different hardware and compare it to other state-of-the-art simulation packages. With growing data volumes the extraction of physically meaningful information from the simulation becomes increasingly challenging and requires equally efficient implementations. A particular advantage of our approach is the easy expression of such analysis algorithms. We consider two popular methods for deducing the crystalline structure of a material from the local environment of each atom, show how they can be expressed in our abstraction and implement them in the code generation framework.

1. Introduction

Molecular Dynamics (MD) codes such as NAMD [1,2], LAMMPS [3], GROMACS [4,5] and DL-POLY [6,7] are important computational tools for understanding the fundamental properties of physical, chemical and biological systems. They can be used to verify phenomenological theories about atomistic interactions, understand complex biomolecules [8] and self assembly processes [9], replace costly laboratory experiments and allow access to areas of parameter space which are very difficult to reproduce experimentally. For example, simulations can be run at high pressures and temperatures found in stellar atmospheres [10], or for dangerous substances, such as radioactive materials (see e.g. [11]). Classical MD codes simulate a material by following the time evolution of a large number of particles which obey the laws of classical physics (in particular Newton's laws [12]) and interact via phenomenological potentials. To extract meaningful information, the state of the system (i.e. the distribution of particle positions and velocities) has to be analysed, for example by calculating pairwise distribution functions. Information on the crystalline structure of a material can be derived by inspecting the local environment of each particle [13–15].

In order to study systems at physically relevant length- and timescales and to produce statistically converged results, modern codes typically run in parallel on state-of-the art supercomputers [2]. With the recent rise of novel manycore chips, such as GPU and Xeon Phi processors, several popular MD simulation packages have been successfully adapted to those new architectures, see e.g. [16–21]. However, developers of MD codes face significant challenges: adapting and optimising existing codes requires not only a deep understanding of the physics and chemistry of the simulated system, but also detailed knowledge of the rapidly evolving hardware. To name just a few complications, GPUs have a complex memory hierarchy (host/device memory, shared memory and local registers) and any data access has to be coalesced to avoid unnecessary data movement. Write conflicts have to be avoided in threaded implementations on manycore chips and recent CPUs,

* Corresponding author.
E-mail addresses: w.r.saunders@bath.ac.uk (W.R. Saunders), r.j.grant@bath.ac.uk (J. Grant), e.mueller@bath.ac.uk (E.H. Müller).

https://doi.org/10.1016/j.cpc.2017.11.006
0010-4655/

such as the Intel Haswell and Broadwell chip, only run at peak performance if the code can be vectorised. Since in practice it is rare for a chemist/physicist to possess the skills for optimising code on this level, it can be very challenging to port MD software to a new architecture and maintain its performance in a rapidly evolving hardware landscape. To address this fundamental issue, we describe an approach based on the idea of a "Separation of Concerns" between the domain specialist and scientific programmer. By using a suitable abstraction, both the scientific capabilities and computational performance can be improved independently.

DSLs for grid-based PDE solvers.

Very similar issues have been faced by developers of grid-based solvers for partial differential equations (PDEs). The key observation there was that the fundamental and computationally most expensive operations can be expressed in terms of a suitable abstraction: the algorithms (e.g. explicit time stepping methods or iterative solvers for elliptic PDEs) can be formulated as the repeated iterations over a set of grid entities (cells, vertices, faces, edges), each of which can hold information, such as a local field value. This expression of the algorithm in a Domain Specific Language (DSL) simplifies the implementation significantly: once the domain-specialist has expressed the code in terms of those basic operations at the correct abstraction level and encapsulated any data in the corresponding fundamental data structures, a computational scientist can implement and optimise the code on a particular architecture.

By introducing the correct abstraction, only a small set of typical loops, which can be parametrised over the set of input and output data, has to be considered. This concept has been applied very successfully in the development of the performance-portable OP2 library [22,23], which allows the execution of finite element and finite volume codes on a range of architectures. As demonstrated in [22–25], the code achieves excellent performance on CPUs, GPUs and Xeon Phi processors. Similar techniques for structured grids have been used to develop the C++ based STELLA grid library for the COSMO numerical weather forecast model [26]. DSLs for highly efficient stencil computations on GPUs have also been described in [27,28].

Recently OP2 was re-implemented in Python as the PyOP2 [29] framework. In PyOP2 the science user specifies the computationally most expensive operations as a set of small kernels written in C. Using code generation techniques, those kernels are then compiled and executed on a particular architecture. By employing just-in-time compilation, the kernels are launched from a high-level Python code which implements the overall solver algorithm. The performance of the resulting code is on a par with that of monolithic Fortran- or C-implementations.

A new DSL for MD simulations.

In this paper we describe a similar DSL approach for molecular dynamics simulations. The fundamental operation we consider is a two-particle kernel: the user implements a short C-code which is executed for each combination of particle pairs in the simulation. This kernel can modify any properties stored on those particles. A classic example is the force calculation: for each pair of particles, the force (output) is calculated as a function of the two particle positions (input). This local operation can be expressed in a few lines of C-code. The code is then executed over all particle pairs, using the optimal algorithm for a particular hardware and the nature and size of the problem. For example, on a CPU architecture, cell-list or neighbour-list methods can be used, whereas on GPU a neighbour-matrix approach as in [30] might be more suitable. Those details of the kernel execution, however, are of no interest

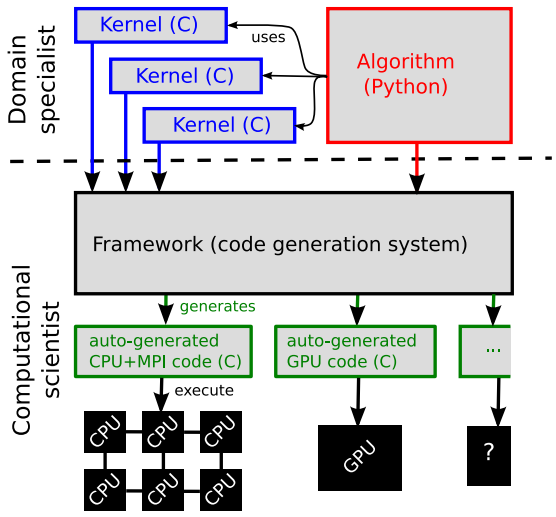

Fig. 1. Structure of the code generation framework. The "Separation of concerns" between the domain specialist user and computational scientist is indicated by the dashed horizontal line.

for the science developer who can focus on (i) the implementation of the local kernel and (ii) the overall algorithm which orchestrates the kernel calls in an outer timestepping loop.

To achieve this we developed a Python-based code generation system which creates and compiles fast, architecture dependent wrapper code to execute the C-kernel over all particle pairs. Our approach is shown schematically in Fig. 1. By using Python as a high-level language, looping algorithms such as the Velocity Verlet method [31] (see also e.g. [32,33]) for timestepping or advanced thermostats [34,35] can be implemented very easily, while still generating fast code for the computationally expensive particle loops.

In the following we describe a proof-of-concept implementation of the DSL and concentrate on short-range two-particle kernels, i.e. kernels which are only executed for particles which are separated by no more than a specified cutoff distance. We demonstrate that for a Lennard-Jones benchmark we achieve performance similar to state-of-the-art simulation tools such as DL-POLY and LAMMPS.

While many atomistic models require the calculation of long range forces and intra-molecular interactions, systems containing only short range interactions remain actively studied, particularly in problems in soft matter and nucleation see e.g. [36,37]. In a separate paper [38] we report on the implementation of a particle-Ewald method [39] for electrostatic forces in our framework. As discussed in Section 6, more advanced long range algorithms and further generalisations of the framework to support multiple species and bonded interactions for molecules will be implemented in the future.

We stress, however, that our approach is not limited to force calculations. To extract meaningful information from a simulation, the results have to be analysed. With growing problem sizes and data volumes, this step becomes computationally expensive and requires efficient and parallel implementations. Below we consider two methods for analysing local environments which can be used to classify the crystalline phase of a material: the bond order analysis in [13] and common neighbour analysis in [14] (see also [15] for an overview of other analysis methods). In the traditional approach, the user would run the simulation with an existing MD package and then write post-processing code to extract physically meaningful information from the output. However, in contrast to the MD code itself, parallelising this analysis code or

Please cite this article in press as: W.R. Saunders, et al., A domain specific language for performance portable molecular dynamics algorithms, Computer Physics Communications (2017), https://doi.org/10.1016/j.cpc.2017.11.006.