PotC: Many-body potential implementations à la carte

Poster Summary

Markus Höhnerbach
RWTH Aachen University
hoehnerbach@aices.rwth-aachen.de

Paolo Bientinesi*
RWTH Aachen University
pauldj@aices.rwth-aachen.de

ACM Reference Format:
Markus Höhnerbach and Paolo Bientinesi. 2018. PotC: Many-body potential implementations à la carte: Poster Summary. In *Proceedings of ACM Student Research Competition at SC17 (ACM SRC/SC18).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Molecular dynamics simulations are a valuable tool in computational chemistry and materials science. The atoms in these simulations move according to so-called potentials. Often, these potentials are pair-wise and describe distance-dependent interactions. However, some applications require many-body formulations, e.g. because of angle-dependent interactions.

Such potentials pose a number of challenges: Small neighborhoods complicate vectorization; redundant force expressions are tedious to manually derive; the implementations are large, runtime-critical, and can not be broken into simpler "kernels".

Effectively, only the most popular many-body potentials are optimized—with major effort.

To tackle these issues, we introduce a DSL for MD potentials and a corresponding compiler to derive high-performance implementations. Our goal is to centralize optimization knowledge for many-body potentials within the compiler, and unburden users from manual optimization and the derivation of forces.

2 MOLECULAR DYNAMICS

Molecular dynamics potentials specify how atoms attract and repulse each other and account for the majority of the simulation runtime. They are given as functions of all atom positions:

$$V(\mathbf{x}_1, \ldots, \mathbf{x}_N).$$

And forces can be derived using

$$\mathbf{F}_i = -\nabla_{\mathbf{x}_i} V.$$

Simplest and most popular potential (pairwise) depends on distance:

$$V = \sum_i \sum_{j, r_{ij} < r_c} f(r_{ij}).$$

Many-body potentials are required to model some materials, and depend on more than just distance: They can depend on the angles between three atoms, the "twist" (dihedral angle) between four atoms, some measure of bond-strength that depends on the local neighborhood, or some kind of per-atom embedding energy.

Fig. 2 lists some potentials included with LAMMPS together with their approximate complexity using LOCs as well as (if available) their vectorized (USER-INTEL package) counterparts, which reuse code from the regular implementation. For most listed potentials forces can not be computed from distances between atom pairs.

To illustrate, we fully define Tersoff potential (Fig. 3).

3 CHALLENGES

Writing good optimized many-body potential implementations poses a number of challenges:

- One has to pick a vectorization strategy that is suitable to the (potentially very short) neighbor list lengths.
- One has to reduce the amount of recalculation for the force computation.
- Especially when vectorizing it is imperative to explore the possibility of reduced accuracy because it allows twice as wide vectors.
- Vectorization of MD potentials has a number of idioms that need to be applied.

4 APPROACH

As indicated on the poster, the compiler performs four steps.

First, the input DSL is parsed, resulting in a "functional" representation of the potential energy (i.e. side effect free). An excerpt of the DSL for the Tersoff potential (Fig. 3) can be found on the poster.

Second, the force derivation step adds the statements for force accumulation, transferring to an imperative representation. It uses the adjoint-based approach well-known from the domain of Automatic Differentiation to do so. It essentially "interprets" the functional representation and emits the corresponding code.

Third, optimizations are performed on the imperative representation as multiple passes. Right now there are passes for inlining, loop and conditional fusion, dead code elimination, common subexpression elimination, arithmetic improvements (e.g. constant folding, sin/cos laws), identification of identical values, and loop invariant code motion. In the future, we plan to add improved common subexpression elimination, propagation of 0, some kind of intermediate value caching, batching and accuracy reduction. There exists rudimentary, buggy support for vectorization that will be improved further.

Last, the compiler generates source code that can be compiled into LAMMPS. LAMMPS is a popular, extensible, open-source, highly scalable molecular dynamics code developed by Sandia National Labs. It contains implementations for a number of many-body potentials, in various implementation variants (threaded, vectorized, GPU, …). In addition LAMMPS provides benchmarks for all its potentials, which can be used to evaluate our results.

* Advisor

ACM SRC/SC18, November 2018, Dallas, Texas USA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

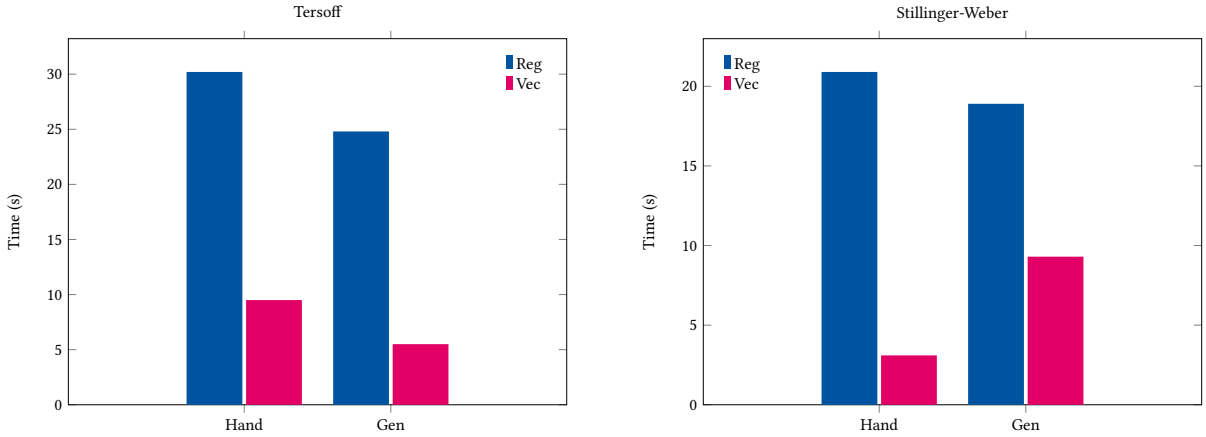

Figure 1: Performance results for two potentials. Handwritten implementation contained in LAMMPS vs automatically generated code, both either regular or vectorized.

Name	LOC-R	LOC-O	Structure
LJ	640	+480	$\sum_i \sum_j f(i,j)$
Stillinger-Weber	600	+1250	$\sum_i \sum_j \sum_k f(i,j,k)$
EAM	840	+820	$\sum_i f(i, \sum_j g(i,j))$
Tersoff	800	+1450	$\sum_i \sum_j f(i,j, \sum_k g(i,j,l)$
MEAM	890	✗	$\sum_i f(i, \sum_j g(i,j))$
ADP	940	✗	too complex to show
BOP	5950	✗	too complex to show
(AI)REBO	4240	+4550	too complex to show
COMB3	3560	✗	too complex to show
ReaxFF	10880	✗	too complex to show

LOC-R: Lines of code of the regular code.
LOC-O: Extra LOC in the optimized/vectorized code.

Figure 2: Selected Potentials from LAMMPS

$$V = \sum_i \sum_j f_C(r_{ij})[f_R(r_{ij}) + b_{ij} f_A(r_{ij})]$$

$$f_C(r) = \begin{cases} 0, & r < R - D \\ \frac{1}{2} - \frac{1}{2}\sin(\frac{\pi}{2}(r-R)/D), & R - D < r < R + D \\ 1 & r > R + D \end{cases}$$

$$f_R(r) = A\exp(-\lambda_1 r)$$

$$f_A(r) = -B\exp(-\lambda_2 r)$$

$$b_{ij} = (1 + \beta^n \zeta_{ij}^n)^{1/(2n)}$$

$$\zeta_{ij} = \sum_k f_C(r_{ik}) g(\theta_{jik}) \exp(\lambda_3^m (r_{ij} - r_{ik})^m)$$

$$g(\theta) = \gamma(1 + c^2/d^2 - c^2/(d^2 + (\cos(\theta) - \cos(\theta_0))^2))$$

Parameters: R, D, A, λ_1, B, λ_2, β, n, λ_3, m, γ, c, d, θ_0.

Figure 3: The Tersoff potential specified using mathematical notation.

5 RESULTS

We present preliminary results, since not all the planned optimizations are yet implemented (Fig. 1). We picked the potentials since they are popular, relatively simple and have both "regular" and "vectorized" implementations. The performance of the generated code matches the handwritten implementation quite well, beating it in all cases but the vectorized Stillinger-Weber case which aggressively caches intermediate values across loops, an optimization that we do not support yet.

6 FUTURE WORK

The tool is currently far from complete. In addition to implementing previously mentioned optimizations, we also want to add features that are not performance-related. In particular, we want support more potentials by adding tabulated functions, FFI and searches through neighbor lists. We are considering a LAMMPS/KOKKOS backend (GPU support) and a KIM API backend (better interoperability). In principle, we could also perform other symbolic manipulations, e.g. calculate parameter derivatives, or accurate Hessians.

7 CONCLUSION

PotC can be a valuable tool for molecular dynamics software authors and users. It allows them to quickly prototype novel potentials, test against a mechanical "ground truth", reach less popular potentials with optimizations, communicate potentials unambiguously, do symbolic manipulation of potentials (e.g. parameter derivatives or Hessians), and avoid writing redundant code and optimizing it by hand.