PotC: Many-body potential implementations à la carte

Markus Höhnerbachh; Advisor: Paolo Bientinesi

Molecular Dynamics (MD)

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:

The compiler is particularly intended for these potentials, since implementing them goes beyond taking the derivative of a scalar function.

Selected Potentials from LAMMPS

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.

Motivation

Many-body potentials are complex to implement, and require large effort to optimize. Less popular potentials will likely never be optimized manually.
And even if there is an implementation, it can be hard to assure that it is correct.
To tackle these issues, this work introduces a Domain Specific Language for molecular dynamics potentials and a corresponding compiler to derive high performance implementations.

Example: The Tersoff Potential

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

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

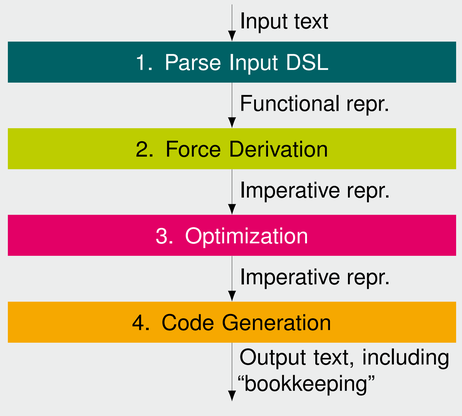
$$f_R(r) = A\exp(-\lambda_1 r)$$
$$f_A(r) = -B\exp(-\lambda_2 r)$$
$$b_{ij} = (1 + \beta^n \zeta_{ij}^n)^{1/(2n)}$$
$$\zeta_{ij} = \sum_k f_C(r_{ik}) g(\theta_{jik}) \exp(\lambda_3^m (r_{ij} - r_{ik})^m)$$
$$g(\theta) = \gamma(1 + c^2/d^2 - c^2/(d^2 + (\cos(\theta) - \cos(\theta_0))^2))$$

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

Challenges

- Picking a vectorization strategy suitable to the neighbor list lengths.
- Reducing the amount of recalculation for the force computation.
- Introducing reduced accuracy modes.
- Applying a number of vectorization idioms.
- Performing optimizations that normal compilers can not apply as they lack MD-specific knowledge.

Approach



DSL for Tersoff Potential

```
energy 1 / 2 * sum(i : all_atoms)
 sum(j : neighbors(i, R(i, j, j) + D(i, j, j)))
  V(i, j);
function V(i : atom; j : atom) =
 f_C(i, j, r(i, j)) *
 (f_R(i, j, r(i, j)) +
  b(i, j) * f_A(i, j, r(i, j)));
function f_C(i : atom_type; j : atom_type;
             k : atom_type; r : distance) =
 implicit(i : i; j : j; k : k)
  piecewise(
   r <= R - D : 1;
   R - D < r < R + D :
    1 / 2 - 1 / 2 * sin(pi / 2 * (r - R) / D);
   r >= R + D : 0);
# ...
function b(i : atom; j : atom) =
 (1 + beta(i, j) ^ n(i, j) *
  zeta(i, j) ^ n(i, j)) ^ (-1 / (2 * n(i, j)));
function zeta(i : atom; j : atom) =
 sum(k : neighbors(i, R(i, j, k) + D(i, j, k), j))
  implicit(i : i; j : j; k : k)
   f_C(r(i, k)) * g(theta(j, i, k)) *
   exp(lambda_3 ^ m * (r(i, j) - r(i, k)) ^ m);

parameter A(i : atom_type; j : atom_type) = file(1);
# ...
parameter gamma(i : atom_type; j : atom_type;
                k : atom_type) = file(5);
# Total: ~ 30 LOC, while LAMMPS impl > 500 LOC
```

Force Derivation

There is one variable (the potential energy) that we want to take many derivatives of.
An adjoint-based approach makes this calculation very straightforward, essentially "interpreting" the functional representation and emitting the corresponding code.

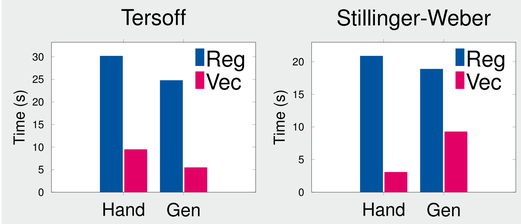
Optimization

- Inlining
- Loop/Conditional Fusion
- Dead Code Elimination
- Common Subexpression Elimination
- Arithmetic Improvements (e.g Constant Folding)
- Loop Invariant Code Motion
- Vectorization (partial)
- Check for 0
- Caching
- Batching
- Accuracy

Code Generation for LAMMPS

For now, 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.

Performance



KNL, double precision, single core.

Future Work

- Extend the range of supported potentials by adding the features they require.
- Target additional parts of LAMMPS, for example to enable parallelization via KOKKOS.
- Target the KIM API to make the generated output more generally interoperable.
- Perform other symbolic manipulations, e.g. calculate parameter derivatives, or accurate Hessians.

Conclusion

PotC can be a valuable tool for molecular dynamics software authors and users. It allows them to ...

- Quickly prototype novel potentials.
- Test other implementations against mechanically derived "ground truth".
- Generate mechanically optimized implementations for the "tail" of potentials that lacks the popularity to be ever optimized manually.
- Have an unambiguous and concise way to transfer and write about potentials.
- Avoid redundancies due to differentiation.
- Avoid optimizing code by hand.

Markus Höhnerbach
Telefon: +49 241 80 99 142
E-Mail: hoehnerbach@aices.rwth-aachen.de

High Performance and Automatic Computing
Schinkelstr. 2, 52062 Aachen, GERMANY
www.hpac.rwth-aachen.de
Telefon: +49 241 80 99 131, Fax: +49 241 80 628498

Aachen Institute for Advanced Study in Computational Engineering Science

High Performance and Automatic Computing

RWTH AACHEN UNIVERSITY