Optimization of the AIREBO Many-Body Potential on KNL

Poster Summary

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

Ahmed E. Ismail*
West Virginia University
ahmed.ismail@mail.wvu.edu

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

ACM Reference Format:
Markus Höhnerbach, Ahmed E. Ismail, and Paolo Bientinesi. 2017. Optimization of the AIREBO Many-Body Potential on KNL. In *Proceedings of ACM Student Research Competition at SC17, Denver, Colorado USA, November 2017 (ACM SRC/SC17)*, 2 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Molecular dynamics simulations provide valuable insights for computational chemists and material scientists. Empirical many-body potentials offer high-fidelity simulations that capture bonding and reaction behaviour accurately, providing a level of detail in between more classical molecular dynamics and quantum methods. The AIREBO potential is an example for a many-body potential, which models carbon and carbohydrates. We present a vectorized and optimized AIREBO implementation for the Intel Xeon and Xeon Phi (co)processors, and integrate it into the popular open-source molecular dynamics code LAMMPS.

This work continues previous efforts to implement an optimized Tersoff many-body potential into LAMMPS. Optimizing AIREBO is more challenging due to its complexity: With about 4000 lines, the original implementation is five times larger than Tersoff.

AIREBO consists of three parts: REBO, TORSION and LJ. While REBO and TORSION are short-ranged interactions between atoms and their closest three or four neighbors, the LJ interaction is longer-ranged (hundreds of interactions per atom). The many-body nature of the potential is owed to two mechanisms: The bondorder term b_{ij} and the connection term C_{ij}. The bondorder describes the bond strength between two given atoms and depends on all angles with their neighbors, torsions (twists) among these neighbors and the bond between the two atoms, as well as the coordination (number of neighbors) of all neighbors. The connection term searches for the "strongest" (in a certain sense) connecting path of up to length three between two atoms among their neighbors. The REBO term depends on the bondorder, the TORSION term on all the possible "twists", and the LJ term on the connecting term as well as (if switched on based on distance) the bondorder.

[*] Advisor
[†] Advisor

ACM SRC/SC17, November 2017, Denver, Colorado USA
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Consequently, there are two "neighborhoods" around each atom: A short one used for REBO, TORSION, b_{ij} and C_{ij}, and a longer one for the LJ interactions. While vectorization for the longer neighborhood is viable, it is not suitable for the short neighborhood which contains about four atoms—too few compared to vector lengths of e.g. eight (KNL double).

Profiling reveals that building the short and longer-ranged neighbor lists and calculating the TORSION and REBO term take about equally long while calculating the longer-ranged LJ interaction takes most of the time, due to more interaction "partners". The bondorder is a large part of REBO, but most often switched off in the LJ calculation.

2 OPTIMIZATION

To optimize most of these routines, we apply techniques tailored to the specific part of the code: First, we introduce an intermediate neighbor list to avoid rebuilding the short-ranged neighbor list from the longer-ranged list each timestep. The intermediate list includes atoms as far outside the cutoff as atoms can move in N timesteps, and as such only needs to be rebuilt every N steps. Consequently, we see an additional speedup of 3 over the vectorization speedup of 2.

Second, we integrate the short-ranged calculations, REBO and TORSION. By grouping interactions of multiple atoms and their respective neighborhoods together it becomes possible to fill up the vector registers. To improve memory access patterns, we segment by atom types and perform an AoS/SoA conversion on the neighbor lists.

For the longer-ranged LJ contribution, the neighbor list is long enough so grouping is not needed. Here we need to filter atoms that are too far away, have a connection term C_{ij} of zero or are calculated separately (when bondorder is required). To improve utilization the vectors are compressed after filtering.

Finally we optimize the C_{ij} calculation: Instead of one search per "i"/"j" pair to find a connection with "j" through all short-ranged neighbors of "i" and their short-ranged neighbors, we conduct one search per atom "i" that collects all those "j" for which the search would succeed into an appropriately-sized hash-map. The collision risk is low and, unlike search, hash-map lookups vectorize well.

3 RESULTS

The benchmarks were performed on a KNL-generation Xeon Phi, and a Broadwell-generation server. Due to platform-independent vector classes vectorization works on both systems (AVX2/AVX-512). We use the AIREBO benchmark provided with LAMMPS—other simulations performed similarly.

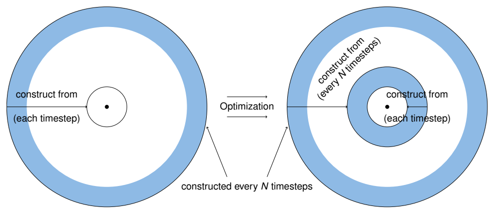

Figure 1: Introducing the intermediate neighbor list.

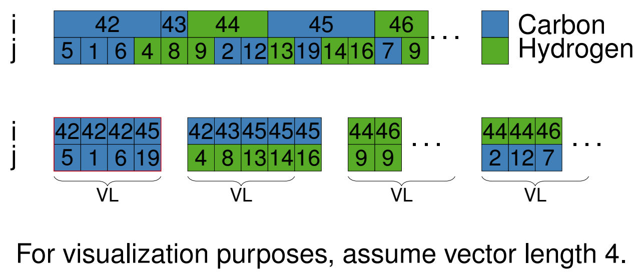

Figure 2: Grouping "i"/"j" pairs together for REBO and TORSION calculation.

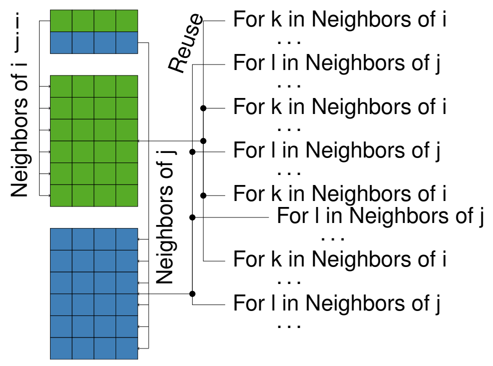

Figure 3: AoS/SoA transform and reuse for bondorder calculation.

Broadly speaking, the BDW system is twice as fast as the KNL system for the original code. On BDW, the optimization speedup is two, and four on KNL. This is consistent with the doubled vector length. Reduced precision leads to another performance gain. Our measurements indicate that the speedup is sustained when scaling on a node from a single rank to multiple ranks. Third-party benchmarks indicate speedups of 3.6 (Broadwell), 6.5 (Skylake SP) and 9.4 (Knights Landing) with reduced precision.

4 CONCLUSION

Allowing many-body potentials to profit from the recent architectural advances still poses a challenged due to deeply nested, short loops. Our work demonstrates that even this adverse case can profit from modern vector hardware. The optimized code will be integrated into the USER-INTEL package of LAMMPS and will soon be distributed with every download of the LAMMPS code.

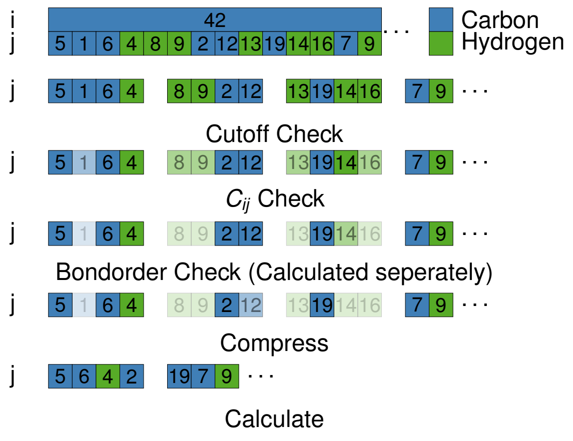

Figure 4: Masking and compression for LJ calculation.

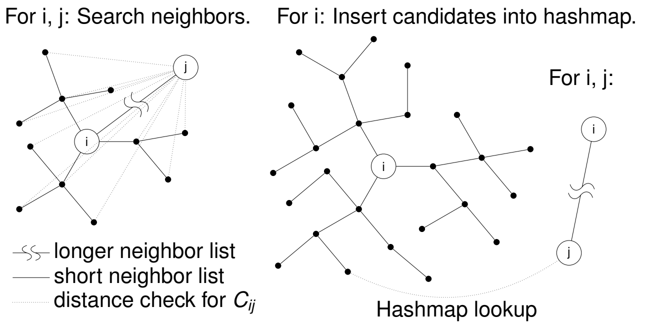
For i, j: Search neighbors.
For i: Insert candidates into hashmap.
For i, j:
longer neighbor list
short neighbor list
distance check for C_{ij}
Hashmap lookup

Figure 5: Replacing search per interaction by search through previously-constructed hashmap.

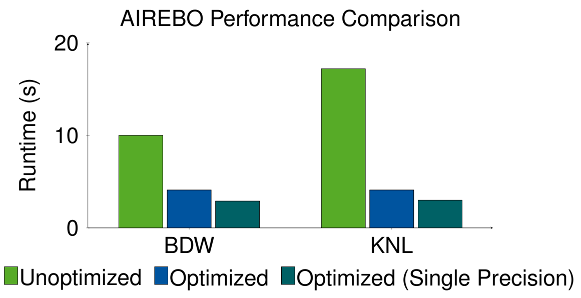

Figure 6: System Performance Comparison.

REFERENCES

[1] Plimpton, Thompson: "Computational aspects of many-body potentials," MRS Bulletin, 37(5), 513-521, (2012).
[2] Stuart, Tutein, Harrison: "A reactive potential for hydrocarbons with intermolecular interactions," J Chem Phys, 112, 6472-6486 (2000).
[3] Plimpton: "Fast Parallel Algorithms for Short-Range Molecular Dynamics," J Comp Phys, 117, 1-19 (1995).
[4] Sodani et al., "Knights Landing: Second-Generation Intel Xeon Phi Product," in IEEE Micro, vol. 36, no. 2, pp. 34-46, (2016).
[5] Höhnerbach, Ismail, Bientinesi: "The vectorization of the tersoff multi-body potential: an exercise in performance portability," SC '16 Proceedings, Article 7 (2016).
[6] M. Brown (Intel): http://lammps.sandia.gov/doc/accelerate_intel.html, Oct. 2017.