Reproduction of Tersoff Multi-Body Vectorization

Illinois Institute of Technology, Maine South High School, Adlai E Stevenson High School

Ryan Prendergast, Iva Veseli, Ryan Mitchell
Hasan Rizvi, Alex Ballmer, David Ghiurco

ABSTRACT

The Vectorization of the Tersoff Multi-Body Potential: an Exercise in Performance Portability (CITE) provides platform portable optimizations to the LAMMPS molecular dynamics algorithm for simulations too complex to optimize with the compiler. As part of the Student Cluster Competition at SC17, we are replicating the experiments presented in the paper, on an 8-node system using the Intel Skylake architecture. We hope to demonstrate that the performance benefits from the paper extend to our cluster's platform.

1. INTRODUCTION

LAMMPS, as a molecular dynamics simulator, recomputes Newton's laws of mechanics for each particle over the course of thousands to millions of timesteps. Each step involves nested *for* loop, iterating over every particle and calculating the force applied by all neighboring particles.

This two-loop structure, however, is only possible in the case of pair potentials, where the force between two particles depends exclusively on the properties of those two. Multi-body potentials such the Tersoff, however, depend on the position of neighboring particles around the two in question in addition to the particles themselves. The nested loop thus becomes much more computationally expensive, with a runtime complexity beyond $O(n^2)$.

M. Höhnerbach *et al* presented a series of optimizations to the LAMMPS algorithms for the Tersoff multi-body potential, with the intention of improving its simulation speed. These optimizations fall into three general categories.

The first change the authors made was to vectorize the code's loop structure. The second was scalar improvements: precalculating and storing derivatives rather than recomputing them for each particle. The third was changing LAMMPS to run in variable precision (either single, double, or mixed). rather than exclusively in double precision.

The paper demonstrates that the three optimizations together improve the simulation speed of LAMMPS across a variety of architectures and platforms. They provided runtime data from ARM processors, three generations of Intel CPUS, Nvidia GPUs, and Xeon Phi accelerators.

Plots from the paper also demonstrate the performance benefit of the optimizations on a variety of run-time parameters, including single-threaded runs, single-node runs, and scalability across multiple nodes.

In this paper, we are trying to replicate the benefits of the optimized code on a single thread, single node, and across nodes, using the standard in.tersoff_bench file as the in.porter *et al.* We also hope to replicate the accuracy These were figures 3, 4, 5, and 9 from the paper.

2. MACHINE DESCRIPTION

The cluster we ran on has 8 nodes. Intel is our vendor; each node has an Intel Xeon Platinum 8180 processor, 384 Gb/node of DDR4 2666 memory, and one NVIDIA V100 GPU. Each runs Fedora version 26, and we use Omni-Path as interconnect.

3. COMPILING AND RUNNING

To compile LAMMPS on our hardware, we used a modified version of the provided "intel_cpu_intelmpi" Makefile (note that we did not use the 2016Mar version of the code). We changed the Makefile to ensure it utilized the AVX instructions of our hardware.

We used Intel MPI, version 18, and the Intel C++ Compiler, version 18. Since we used the latest version of the Intel Compiler, which was incompatible with the March 2016 version of LAMMPS, we used the latest version from their repositories.

To run LAMMPS, simulations take an in.X file. These are formatted as a series of commands and values, specifying parameters such as the particles' initial positions, their type, and the type of potential to calculate (for the in.tersoff_bench and in.porter files, this was the Tersoff potential).

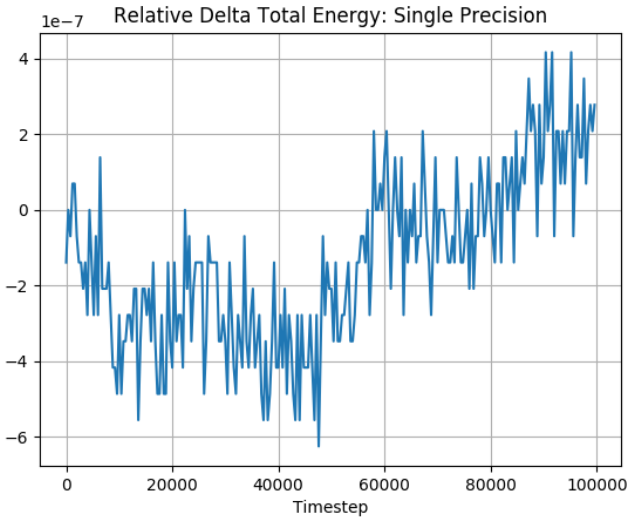
The run command itself takes this input file as a parameter: "lmp_binary -in in.X". It can take other parameters, including "mode [single,double,mixed]" or "balance [-1,0,1]" to run on a MIC accelerator. Our input command was "-in [in.tersoff_bench, in.porter] -pk intel 1 mode [single,double,mixed] balance 0 -sf intel".

4. ACCURACY STUDY

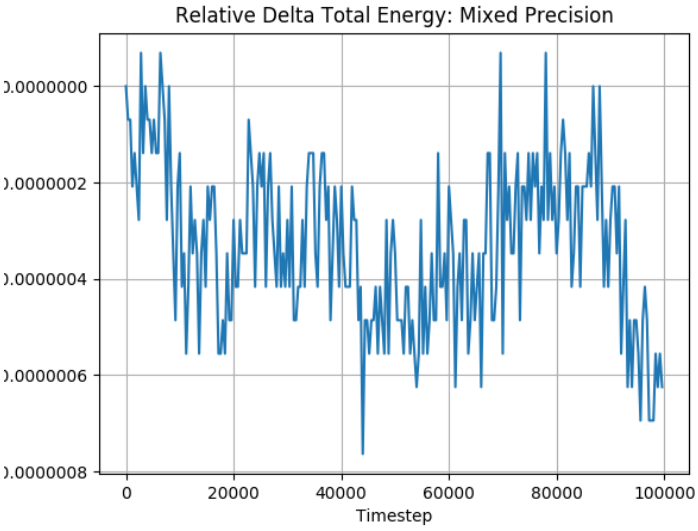
One optimization by Höhnerbach *et al* changed the precision of LAMMPS. The paper introduced three levels of precision: single, double (the default for LAMMPS), and "mixed" precision, with potential calculations done in single precision and accumulations in double precision.

Before they demonstrated the performance benefits of variable precision, they first showed that the relative accuracy loss from using a lower precision than double was not too significant (orders of magnitude bellow a percent). We sought to replicate this, both for the single and mixed precision version.

4.1 Single Precision



4.2 Mixed Precision



4.3 Analysis

We found that there was little significant loss in accuracy using the single or mixed precision versions of the code versus the double precision version.

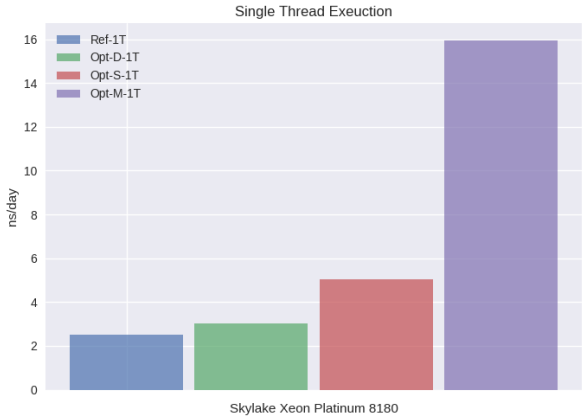
5. PERFORMANCE DIFFERENCE

Next, the authors of the paper compared the performance speed of the optimized version to the reference one. This is measured in terms of simulated time/run time (in nanoseconds per day). We sought to replicate their demonstrated the benefits of the improvements on a single thread, single node, and across nodes, with two separate input files.

\

5.1 Single Thread

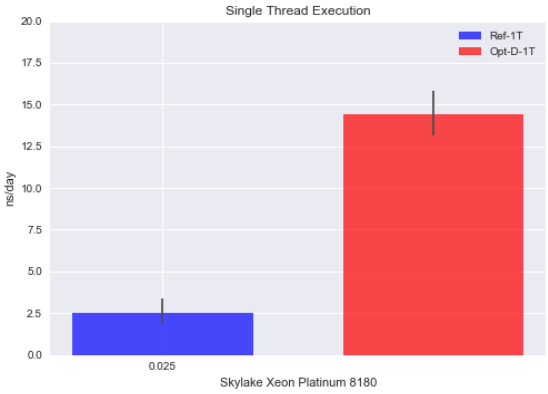
The single thread experiment was run using the in.tersoff_bench input file.



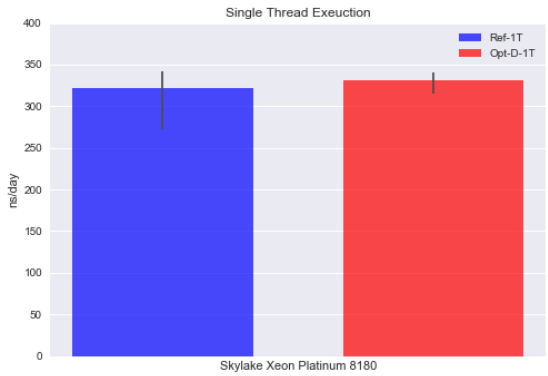
5.2 Single Node

We ran single node benchmarks using the the in.tersoff_bench input file and the in.porter file.

5.2.1 In.tersoff_bench



5.2.2 In.porter



5.3 Analysis

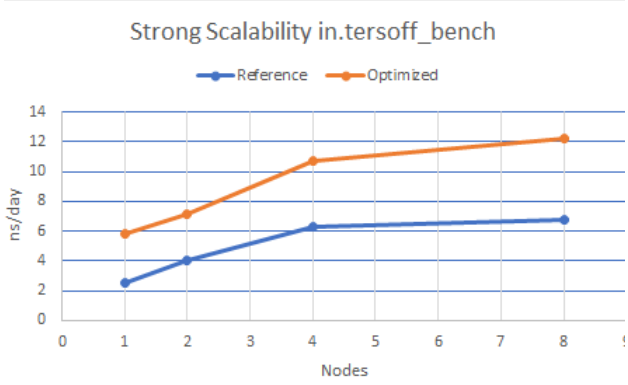
The optimizations seemed to provide a performance improvement for the in.tersoff_bench input file, but provided little to no difference to the in.porter file. The results from the in.tersoff_bench experiment correspond to the results presented in the paper, but the in.porter result's seem to contradict them.

We hypothesize that this has to do with in.porter's relative small runtime compared to in.tersoff_bench. At such a small runtime, computational improvements become less relatively consequental.

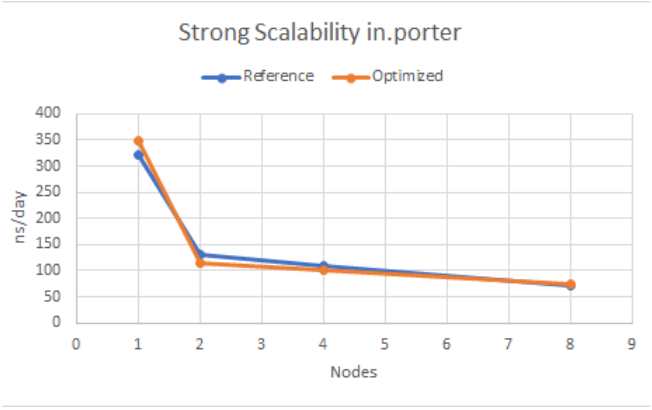
6. STRONG SCALABILITY

The last experiment we sought to replicate was a scalability experiment performed from a single to eight nodes. We performed this experiment with both the in.tersoff_bench and in.porter files.

6.1 In.tersoff_bench



6.2 In.porter



6.3 Analysis

We found a similar trend in the scalability study as with the single node run. We had similar results to the paper with the in.tersoff_bench experiment, but found the exact opposite to be the result with the in.porter file.

We hypothesize that this result stems from similar reasons as the results in the single node run. The in.porter takes significantly less time in computation than in.tersoff_bench and spends much more time in communication. This explains the decrease seen in simulated performance when increasing the number of nodes: with more nodes added, more time needed to be spent in MPI to communicate between them.

7. CONCLUSIONS

When LAMMPS uses an input file which makes it predominately compute bound, such as with the in.tersoff_bench input file (but not in.porter), the scalar, vector, and precision optimizations performed by Höhnerbach *et al* improve the simulated runtime by a significant margin. This was true on a single thread, a single node, and across nodes.

So, we were able to replicate the results of the paper for the in.tersoff_bench input file. We believe that this was possible because the improvements presented in the paper were platform independent; they changed the algorithm in its source code (which is itself able to run on any patform with a C++ compiler). Further, we ran the code on hardware similar (though not identical) to the Intel architectures which were tested in the paper itself.

We were not able to replicate the results of the paper with the in.porter file. However, we believe that if the runtime of the simulation were increased, then the plots for that would mirror that of the in.tersoff_bench file, as the overhead of inter-node communication with MPI would no longer overwhelm the computation.