Student Cluster Competition 2017, Team University of Utah: Reproducing Vectorization of the Tersoff Multi-Body Potential on the Intel Broadwell and Intel Skylake Platforms

Janaan Lake, Quixiang Chao, Hannah Eyre, Emerson Ford
Kevin Parker, Kincaid Savoie, Mary Hall, Hari Sundar

School of Computing, University of Utah, Salt Lake City, Utah, USA

Abstract

This paper describes the University of Utah SC17 Student Cluster Competition team's efforts to reproduce the results of an SC16 paper titled "The Vectorization of the Tersoff Multi-Body Potential: An Exercise in Performance Portability."

This application optimizes the Tersoff potential within the widely-used molecular dynamics code LAMMPS. Tersoff multi-body potentials compute the force between two particles as a function of their distance and the relative position of surrounding particles. The application described in the cited paper focuses on providing node level and SIMD parallelization strategies for the Tersoff multi-body potential that are portable to different CPU instruction sets and to GPUs. In this paper we collect measurements on our SC17 cluster, which consists of a Broadwell head node and Skylake compute nodes. Our testing demonstrates that we can produce similar speedups to the original publication on the AVX2 instruction set on both the Broadwell and Skylake platforms. Additionally, we examined the performance on the Skylake node using the AVX-512 instruction set.

Keywords: Reproducibility, LAMMPS, SCC17

1. Introduction

The ability to reproduce results is essential to advancing algorithms and optimizations suitable for high-performance computing. At the SC17 conference, a component of the Student Cluster Competition requires teams to reproduce the results from a prior conference by reading a paper and

running tests described therein on their cluster. The overall competition involves a team of six undergraduates designing a new cluster from state-of-the-art hardware and software, running a set of required applications on the hardware, and providing written and oral descriptions of the results. This paper presents the outcome of the reproducibility exercise from SC17 using results that were collected during the conference on the University of Utah Salt Flats cluster. This cluster consists of one Broadwell head node and four Skylake compute nodes, using a Mellanox EDR interconnect. The cluster also has eight Nvidia Tesla v100 GPUs.

The selected paper for the reproducibility exercise [1] describes an approach to extending the LAMMPS molecular dynamic simulator [2] with a newly optimized and portable implementation of the Tersoff multi-body potential. Tersoff multi-body potentials compute the force between two particles as a function of their distance and the relative position of surrounding particles. The focus of [1] is on providing node level, SIMD parallelization strategies and scalar optimizations for this computation that are portable to different CPU instruction sets and to GPUs. Performance portability is achieved by focusing on a core computation that consists of three nested loops. Depending on vector width, the computation is parallelized at different loop levels. MPI is used to provide parallelism within and across nodes; SIMD execution provides data parallelism.

In this paper we report the measurements recorded on our SC17 cluster. The competition required a comparison of results from running the code on either CPUs or GPUs. We chose to run the code on our CPUs rather than the GPUs because we wanted to compare the performance on the different CPU architectures within our cluster. We were particularly interested in comparing the performance on the AVX2 and AVX-512 instruction sets on the Skylake nodes. During the competition our tests were run using the AVX2 instruction set on both the Broadwell and Skylake nodes. After the competition, we ran the same tests using the AVX-512 instruction set on one of the Skylake nodes. We test the portability and speedups of these optimizations on our cluster using tests on (1) a single processor; (2) an entire node; and, (3) all four nodes of our cluster. Overall, we can reproduce most of the results from [1] except for the accuracy test.

2. Salt Flats Cluster and Software Details

Our team partnered with Dell, Nvidia and Intel to build the Salt Flats cluster. The resulting system has one Dell PowerEdge R730 head node and four Dell PowerEdge R740s compute nodes. The head node has two Intel

Xeon E5-2680 v4 CPUs clocked at 2.4 GHz and supports AVX2 instruction sets. Each compute node has two Intel Xeon Gold 6130 CPUs clocked at 2.1 GHz and supports AVX-512 instruction sets. The head node has 2 sockets with 14 cores and 128 GB memory for each socket. The compute nodes have 2 sockets with 16 cores, and each socket has 96 GB of memory. Each compute node has two Nvidia Tesla v100 GPUs. The interconnect is Mellanox EDR. The cluster runs CentOS 7 with Linux kernel version 3.10.0-693.2.2.el7.x86_64. To compile LAMMPS, we cloned the repository from GitHub at `https://github.com/HPAC/lammps-tersoff-vector`. We used the LAMMPS version 10Mar16 that was provided in this repository. We modified the build script for our architecture and used code from the USER-INTEL package, which collects optimizations for Intel hardware. The file `Makefile.intel_cpu`, found in the `/src/MAKE/OPTIONS/` directory, was used as the make file. We used Intel 18.0 and Intel MPI 5.1.3 to build the binaries.

3. Optimizations

Various optimizations have been made to the LAMMPS code to improve performance, and these are outlined in [1]. They include scalar optimizations, vectorization, and optimizations that aid vectorization. The descriptions of these optimizations have been taken from [1] and are reproduced below.

3.1. Scalar Optimizations

Scalar optimizations include restructuring the algorithm so that certain terms are computed only once. This requires more memory for additional storage. If the storage requirements exceed a certain limit, the original algorithm is used instead.

3.2. Vectorization

There are three vectorization strategies that map the three-dimensional iteration space to different levels of parallelism. The inner loop, or third dimension, is executed sequentially in each execution scheme. The first execution scheme maps the outer loop to parallel execution and the second loop to data-parallelism. It is used primarily for short vector lengths. The second execution scheme fuses the two outer loops, and both are mapped to parallelism and vectorization. This execution scheme is used for vector lengths of 8 to 16, which includes AVX2 and AVX-512, and is the scheme used during our tests. This mapping is utilized when the vector lengths

exceed the iteration count of the second loop. However, because the first iteration is not constant across all lanes it cannot attain the same efficiencies that the first scheme does. Special care has to be taken to avoid cache conflicts. The last execution scheme involves mapping the outer loop to parallel execution and data-parallelism, and the second loop is executed sequentially. This scheme is best suited for GPUs. An iteration of the first loop is assigned to each thread, and the thread sequentially works through the second loop nest. Our reproducibility attempt did not include any test runs on GPUs, so this execution scheme is less relevant to our results.

3.3. Optimizations to aid vectorization

Additional optimizations include avoiding masking for the last two execution schemes to increase the number of vector lanes actively computing the inner loop. This is done by manipulating the iteration index of the outer loop independently in the various lanes. Filtering the neighbor list is another technique used to reduce the amount of masking done to increase the time spent by the algorithm in the numerical calculation section.

4. Results

The architectures that are used in [1] include ARM, Westmere, Sandy Bridge, Ivy Bridge, Haswell, Broadwell, Nvidia's Kepler-generation Teslas, and two generations of Intel Xeon Phi (Knights Corner and Knights Landing). Our cluster is heterogeneous and includes a Broadwell head node and four Skylake compute nodes. Because of this configuration, we tested the optimizations on both types of nodes. The Broadwell platform is used in [1], which allows for an even comparison of the AVX2 instruction sets on our cluster. The Skylake architecture is not included in [1]. Our evaluation ranges from a single processor, a single node and to all four compute nodes of the cluster. We were provided four data sets to use for the test runs: `in.tersoff`, `in.tersoff_bench`, `in.tersoff-acc` and `in.porter`.

Our intent was to run the tests using the AVX2 instruction set architecture on the Broadwell node and the AVX-512 instruction set on the Skylake nodes. Because the Skylake platform supports the wider vector length and more parallelism, we expected the speedups to be greater on the Skylake node. However, our tests during the competition exhibited lower speedups on the Skylake node than on the Broadwell node. After the competition we investigated this discrepancy. We discovered that due to a missing compilation flag our code was compiled to run on the AVX2 instruction set on the Skylake node rather than the AVX-512 instruction set. Therefore,

our results from the competition compare the differences of running the optimized LAMMPS code using AVX2 on both the Broadwell and Skylake nodes. Later, we recompiled the code for AVX-512 and ran the same tests on one of the Skylake nodes. We compared these results to the ones we produced during the competition, and these are discussed in Section 4.5.

4.1. Accuracy Test

The default LAMMPS code uses double precision floating point operations. The optimized code provides implementations to compute the Tersoff potential using single and mixed precision. To validate the reduced precision implementations, the difference in energy is measured during long-running simulations. We examined the results of two long-running simulations tests, comparing the accuracy of the double precision with the mixed precision and the accuracy of the double precision with the single precision. We used the `in.tersoff-acc` data set. These tests were run on one of our compute nodes, and the results can be seen in Figures 1 and 2. In these figures the x axis represents the number of timesteps up to 10^6, and the y axis is the relative difference in the total energy used by each precision model.

Our results diverge from those in [1] as the relative difference increases over the number of timesteps. The author of [1] responded to the results, saying the code used by the teams for the competition contained a bug that could cause the relative difference to increase over many timesteps. However, this bug has been fixed and included in the current public repository. Unfortunately, we were not able to conduct this accuracy test on the updated code for the competition.

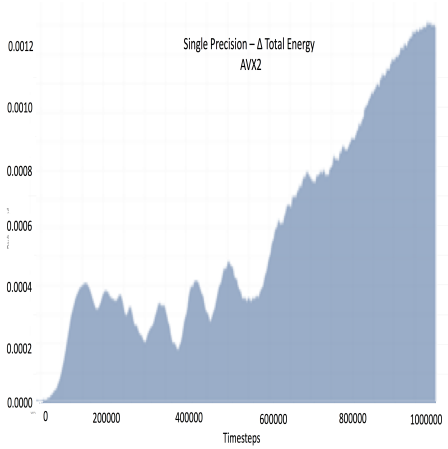

Figure 1: Validation of the single precision solver: relative difference between the single and double precision solvers for a system of 32,000 atoms for 10^6 timesteps.

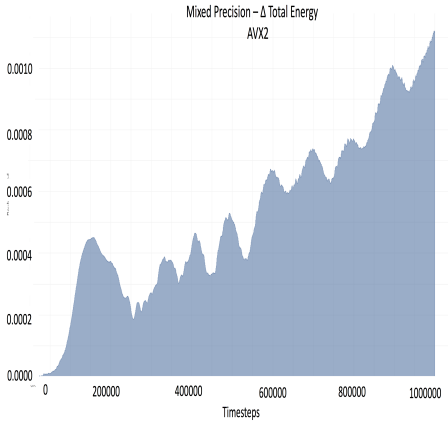

Figure 2: Validation of the mixed precision solver: relative difference between the mixed and double precision solvers for a system of 32,000 atoms for 10^6 timesteps.

4.2. Single-Threaded Execution

Using the `in.tersoff` data set, we ran single-threaded execution tests using four different modes: Ref, Opt-D, Opt-S, and Opt-M.

- *Ref:* This mode is the original LAMMPS distribution, which is performed in double precision. It is used as the reference for the tests in [1].
- *Opt-D:* This is the most accurate version of the code, performing calculations in double precision. It includes optimizations made to the LAMMPS distribution, including scalar improvements and vectorization.
- *Opt-S:* This is the least accurate version of the code, performing calculations in single precision and including the scalar optimizations and vectorization. The vector length is typically twice that of the Opt-D mode.
- *Opt-M:* This is a mixed-precision version of the code with the purpose of compromising between speed and accuracy. It uses single-precision for all calculations except the accumulation operation, which is done in double precision.

Figure 3 illustrates single-threaded performance for all the execution modes on both the Broadwell and Skylake platforms, using the `in.tersoff` dataset. The error bars represent the range between the minimum and maximum results from a repeated series of similar tests. We observed speedups of 3.7 between Ref and Opt-D and 4.8 between Ref and Opt-S on the Broadwell node. On the Skylake node, we saw speedups of 3.4 between Ref and Opt-D and 4.3 between Ref and Opt-S. In [1] the results recorded for the AVX2 instruction set using the Haswell architecture are 3.0 between the Ref and Opt-D tests and 4.8 between Opt-S and Ref. Our results, therefore, are similar for AVX2. A potential explanation for the smaller speedups shown by the Skylake platform compared to the Broadwell may be some overhead incurred when mapping the AVX2 instruction set to the wider vector length of the AVX-512 instruction set.

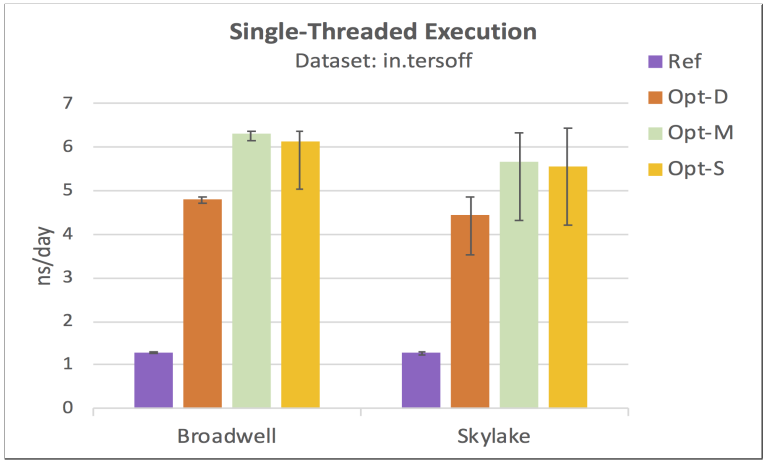

Figure 3: Evaluation of performance portability across CPUs; single-threaded execution for a system of 32,000 atoms.

4.3. Single Node Execution

Using a single node, we ran tests implementing MPI and OpenMP and including the scalar optimizations and vectorization of the original LAMMPS code. These speedups are a bit less dramatic as the cost of parallelism reduces some of the efficiencies of the optimizations. Only two modes were used for these tests in [1], Ref and Opt-M. Therefore, we ran our tests in these two modes. We used two datasets for these experiments: `in.tersoff_bench` and `in.porter`. We ran these tests on a single Broadwell node with 14 MPI

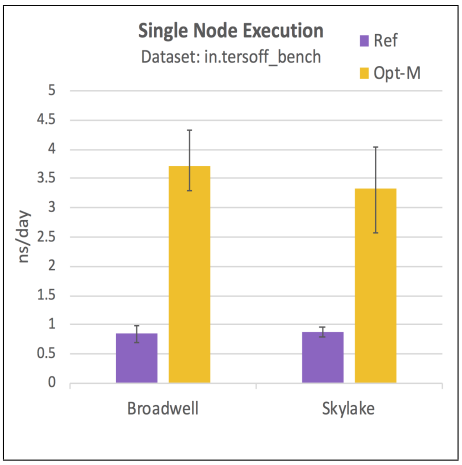

Figure 4: Evaluation of performance portability across CPUs; single node execution for a system of 32,000 atoms.

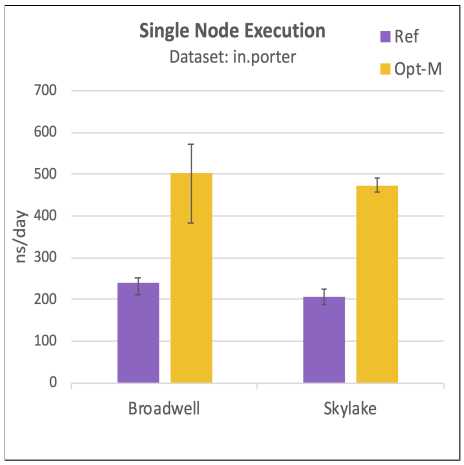

Figure 5: Evaluation of performance portability across CPUs; single node execution for a system of 216 atoms.

ranks per node. For the single Skylake node we used 16 MPI ranks per node. Using the `in.tersoff_bench` data set we observed speedups of 4.3 on the Broadwell platform and of 3.76 on the Skylake platform. These results are shown in Figure 4. Figure 5 shows observed speedups of 2.01 and 2.28 on the Broadwell and Skylake nodes respectively using the `in.porter` data set. The speedups for the `in.tersoff_bench` are above the range of speedups for the same tests in [1]. However, the range of speedups for the `in.porter` data set are slightly below the range displayed in [1] for similar vector units. This dataset had neighbor lists nearly double in size to the `in.tersoff_bench` dataset, which affected its performance. The average of the results for these two datasets is within the range of between 2.69 and 3.15 recorded in [1] for architectures using the AVX2 instruction set.

4.4. Scalability

The last tests of our reproducibility exercise involve scaling the optimizations on the Salt Flats cluster. We used the `in.tersoff_bench` and `in.porter` datasets, running the same tests on one node, two nodes, and four nodes. For the test running on one node we used 16 MPI ranks. For tests running on two nodes we used 32 MPI ranks with 16 ranks per node, and for tests with four nodes we used 64 MPI ranks with 16 ranks per node. All of these tests were run on our Skylake compute nodes, using the Ref and Opt-M modes. As shown in Figure 5, the tests on dataset

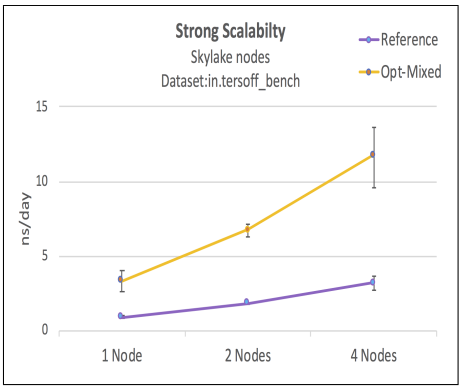

Figure 6: Optimization results on the SaltFlats cluster using the `in.tersoff_bench` dataset.

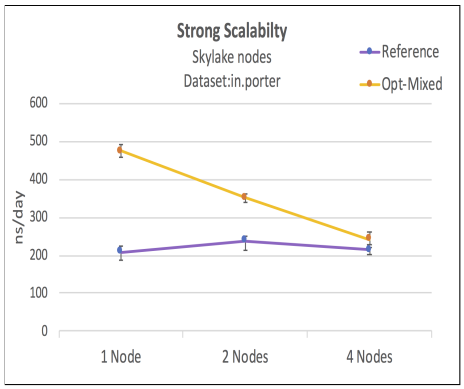

Figure 7: Optimization results on the SaltFlats cluster using the `in.porter` dataset.

`in.tersoff_bench` show strong scaling across all four nodes. However, the dataset `in.porter` does not scale well across the nodes. The time spent in communications for this dataset appears to outweigh the performance gains from the optimizations.

4.5. AVX2 vs. AVX-512

As discussed earlier, our goal during the competition was to compare the results of the AVX2 and AVX-512 instruction sets. After the competition ended we discovered that a flag was missing during the compilation, causing the code to run on AVX2 for all of our tests. By adding the flag `-xCore-AVX512`, the tests could be run on the AVX-512 instruction set. We recompiled the code and reran the tests on one of our Skylake nodes. The results of the single-node tests using the two modes, Ref and Opt-M, on the `in.tersoff_bench` data set are shown in Figure 8. This graph compares the results of these tests on the Skylake node using the AVX2 and the AVX-512 instruction set. The AVX-512 instruction set reported speedups of 4.3, which is somewhat lower than the 5.0 speedup demonstrated in [1] for the Xeon Phi systems but higher than the 3.7 speedup using the AVX2 instruction set on the same Skylake node. The Knight's Landing architecture has a throughput-oriented core so its performance will vary from the general-purpose platform of the Skylake architecture. Therefore, even though the AVX-512 instruction set is tested in [1], the comparison of our results to [1] is not necessarily a fair comparison. However, our results do demonstrate some speedups of the optimized LAMMPS code in [1] for the AVX-512 instruction set.

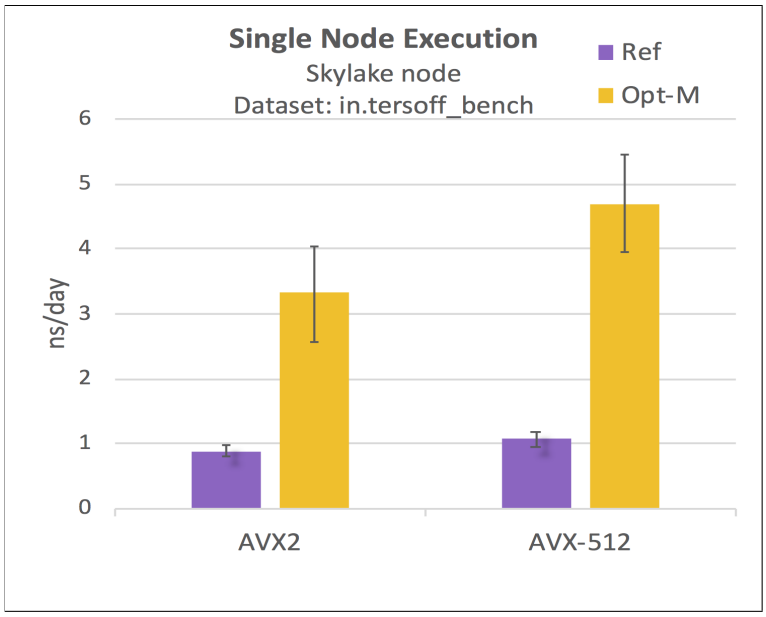

Figure 8: Evaluation of performance portability across CPUs; single node execution for a system of 32,000 atoms.

5. Conclusion

We ran numerous tests on four data sets to reproduce the results presented in [1]. The accuracy test results are not similar to the accuracy test presented in [1] due to a bug in the code. The relative difference in our tests is larger by a factor of 50 and increases over the number of timesteps. The datasets `in.tersoff` and `in.tersoff_bench` display speedups similar to [1] for the single-threaded execution and single node execution test on the AVX2 instruction set. The `in.tersoff_bench` data set displayed strong scaling on our compute nodes. These results demonstrate the portability of the optimized LAMMPS code for the Tersoff potential on our cluster. We hypothesize that the slightly smaller speedups on the Skylake node versus the Broadwell node illustrate some overhead in mapping the data to a smaller vector width on the AVX-512 architecture. The results from the `in.porter` dataset are not as strong in displaying the portability of the optimized code. We observed speedups just below the range reported in [1]. This dataset also did not scale well across our cluster. These results suggest that different datasets will display varying levels of speedups and scalability. Possible reasons include the size of neighbor lists, how well the data set maps to the parallelization scheme, and the time spent on computation

versus communication.

The observed speedups from our later tests using the AVX-512 instruction set are below the speedups reported in [1] for the Xeon Phi platforms. However, the comparison of our results to the AVX-512 measurements in [1] is not necessarily an accurate comparison since the Knights Landing and Skylake architectures have different performance goals and therefore varying performance results.

References

[1] M. Höhnerbach, A. E. Ismail, P. Bientinesi, The vectorization of the tersoff multi-body potential: An exercise in performance portability, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '16, IEEE Press, Piscataway, NJ, USA, 2016, pp. 7:1–7:13.
URL `http://dl.acm.org/citation.cfm?id=3014904.3014914`

[2] http://lammps.sandia.gov.