# Reproducing the Vectorization of the Tersoff Multi-Body Potential: An Exercise in Performance Portability on NVIDIA Tesla V100 GPUs

Zachary Marcus
Northeastern University of Electrical
and Computer Engineering
Boston, MA 02119
marcus.z@husky.neu.edu

## ABSTRACT
Molecular dynamics simulations have become an indispensable research tool in computational chemistry and materials science. As such, they consume a significant portion of compute resources around the world. Multi-body potentials provide increased accuracy but are complex and present portable, performant programming challenges. A paper was published at the Supercomputing 16 Conference last year presenting the results of developing an efficient and portable Tersoff potential within the molecular dynamics code LAMMPS. The paper provided a detailed artifact description, from which this paper presents the validation of the results on a recent NVIDIA GPU, the Volta V100.

## CCS Concepts
• **Computing methodologies~Molecular simulation** • **Applied computing~Chemistry** • *Software and its engineering~Software performance*

## Keywords
Reproducibility; LAMMPS; Molecular Dynamics; Portability

## 1. INTRODUCTION
Molecular dynamics simulations trace particles and their trajectories over time for use in materials science research. The computation for this is targeted at highly parallel architectures. Pairwise interactions make up much of these interactions, but for certain use cases the computations must be adjusted given the interactions between other nearby particles. This adjustment makes the computation of such multi-body potentials much more expensive. The Tersoff multi-body potential, one such computation found within LAMMPS, was explored and optimized in the SC16 paper "The Vectorization of the Tersoff Multi-Body Potential: An Exercise in Performance Portability".

The paper explored the improved performance effects of using a modified programming model to allow for increased performance through scalar and vector optimizations in a way that is portable across architectures. The presented architectures included ARM architectures, Westmere, Sandy Bridge, Ivy Bridge, Haswell, Broadwell, Kepler, Knight's Corner and Knight's Landing. The authors built upon existing LAMMPS packages – USER-INTEL, USER-OMP, KOKKOS, and GPU – for optimization.

The paper provided a compliant artifact description in the Appendix which described how the results were achieved, how they could be reduced, and where others might find the source code itself to build and reproduce their results. It is the intent of this work to replicate a subset of those results on an x86 platform using NVIDIA Tesla V100 accelerators.

## 2. MACHINE DESCRIPTION
The system used for reproducing the work was a four-node Supermicro system connected via Mellanox Infiniband EDR, with the following hardware per node:

**Table 1 Node Configuration**

| Hardware | Description |
|---|---|
| Physical Cores | 2x 32 core EPYC 7551 @2GHz |
| SMT | Enabled |
| Memory | 256GB @2666MHz |
| Operating System | Ubuntu 16.04.3 LTS |
| Kernel | 4.11.0-kfd-compute-rocm-rel-1.6-180 |
| Accelerator | 2x Nvidia V100 GPU for 2 nodes |
| Accelerator | 2x Nvidia P100 GPU for 2 nodes |

## 3. COMPILATION / RUN DESCRIPTION
### 3.1 Compilation
The compilation for this system is based off the Tersoff repository published in the paper. The rwth-sb_tesla machine description was used as a base but modified for the system: no binaries dependent on the USER-INTEL package were used, as they were not found to run on the AMD CPUs. The NVIDIA CUDA toolkit version 9.0 was installed on the system along with the Mellanox bundled version of OpenMPI, 3.0.0rc6, and gcc 5.4.0. The LAMMPS version is the one bundled into the Tersoff repository from March 10th, 2016, with relevant bug fix. The Tersoff version is commit hash 092e33b70c53b71310bc93c51e89d1f48cc1b870. There are also necessary modifications to the source code that had to be made for the test system. For a detailed version of these modifications and the build instructions, see Appendix A.

### 3.2 Run
To run the Tersoff potential code, ensure that mpirun is in the PATH and libmpi.so is in the LD_LIBRARY_PATH. To run from a directory with the proper input files, the command was issued with relative path to the binary along with appropriate flags. Such a run for the KOKKOS package would resemble "mpirun -np 1 ../../machines/reproduce/lammps-10Mar16/src/lmp_kokkos_cuda_vect -in in.tersoff_bench -k on t 0 g 1 -sf kk -log output.log". These flags are the same ones provided in the paper to run with the package; to run for the GPU package, a similar command would be issued but specifying a different

package, and using a separate input file that did not include the KOKKOS tweaks present in the KOKKOS input file. For a more detailed discussion of what was run, see Appendix A.

## 4. ACCURACY STUDY

The authors of the original paper created versions that compute the Tersoff potential in single and mixed precision in addition to the pre-existing double precision calculations. The results of running the provided *in.tersoff-acc* with the GPU package with varied precision showed more deviation than the paper presented by a factor of 3.5.

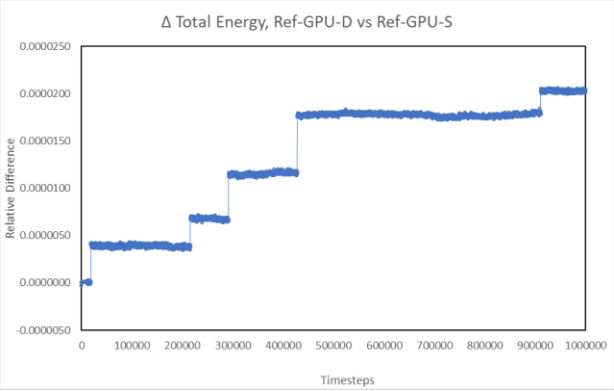


**Figure 1. Validation of the single precision solver: relative difference between the single and double precision solvers for a system of 32000 atoms for 10^6 timesteps using the GPU package.**

In the presented comparison between single and double precision for the LAMMPS GPU package, the error grows increasingly larger than the error reported in the original paper.

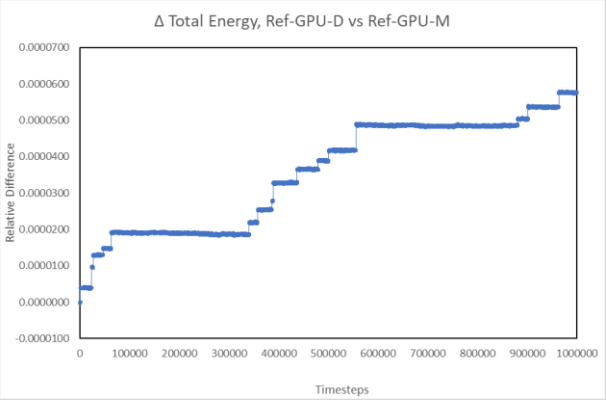


**Figure 2. Validation of the mixed precision solver: relative difference between the mixed and double precision solvers for a system of 32000 atoms for 10^6 timesteps using the GPU package.**

In this second comparison between mixed and double precision, the error still grows increasingly larger. The error presented here is even larger than that of the single precision run. This result is not intuitive, but clearly accumulates error the longer it gets run on the GPU. As a comparison, the KOKKOS package with its double precision computation is presented here as well.

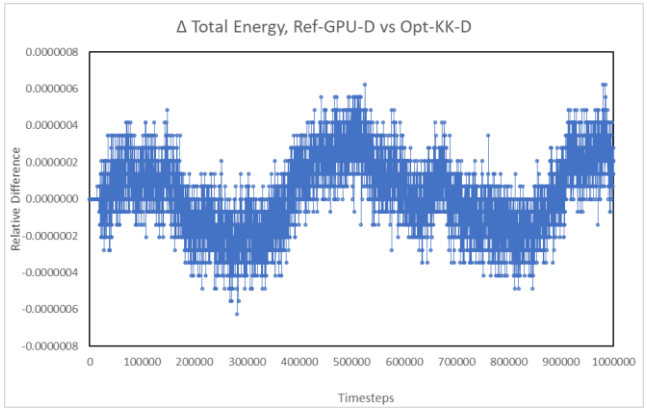


**Figure 3. Validation of the mixed precision solver: relative difference between the vectorized KOKKOS double and GPU double precision solvers for a system of 32000 atoms for 10^6.**

While the calculation is very similar, the difference between these two runs demonstrates the precision capable on the GPU. It is believed that the kernel calculation run on the GPU are formulated in a way that causes a loss in precision that only ever causes it to round up. As seen in these figures, the deviation is within 0.007% of the reference. The round-off error is shown to accumulate in contrast with the presented findings of the paper but still remains within relatively small bounds for a simulation of this size.

## 5. PERFORMANCE DIFFERENCES

The GPU benchmarks were run on a single Volta generation GPU as listed in Table 2.

**Table 2. GPU Information**

| Hardware | Description |
|---|---|
| Name | Tesla V100-PCIE |
| Memory | 16GB @877MHz |
| Clock Speed | 1380Mhz |
| Compute Units | 80 |
| CUDA Runtime | 9.0 |

The performance of the NVIDIA Tesla V100 GPU was mostly in line with what was found on the Kepler card performance. As in the original paper, the LAMMPS GPU package was measured in double (Ref-GPU-D), single (Ref-GPU-S), and mixed (Ref-GPU-M) precision; the KOKKOS package (Ref-KK-D) and the optimized implementation building off of KOKKOS (Opt-KK-D) run in double precision.

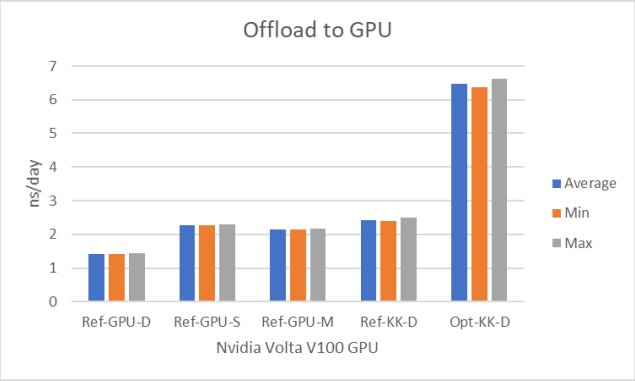


**Figure 4. Evaluation of performance portability for NVIDIA V100 GPUs. 256000 atoms, using**

Figure 4 presents performance measurements on one GPU, just as in the main report, with as small host involvement as possible. The measured performance is very stable, with small deviations of less than 0.5%. The GPU here is the only hardware doing and processing of the data which the single core offloads the work to it. The speedup achieved on the V100 GPU, around three times the performance, is in line with the reported speed-up from the paper. Running on a newer architecture has resulted in the speed of simulation in nanoseconds per day to increase, but the proportional increase in simulation speed is the same.

For the second input file to be studied, in.porter, functionality not present in either the GPU or the KOKKOS packages was required. See Appendix B.

## 6. STRONG SCALING STUDY

The KOKKOS package and GPU package both seem to encounter issues with this system configuration to scale beyond a singular node. See Appendix B for more information on this observed behavior.

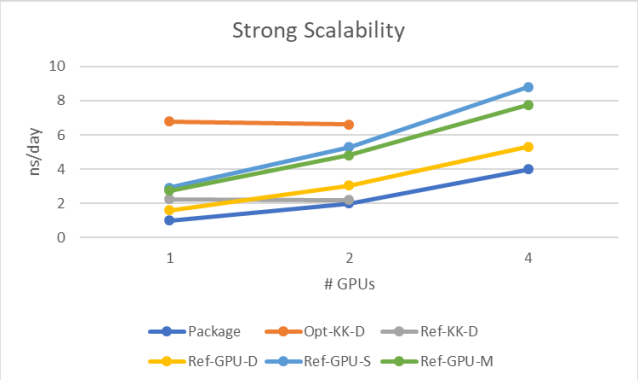


**Figure 5. Evaluation of strong scaling for NVIDIA V100 GPUs. 256000 atoms, using in.tersoff_bench**

Because of the errors with running across nodes with KOKKKOS, the scaling study here must be limited to two GPUs on a single node. The observed behavior is a loss in performance when scaling to even one more GPU with the KOKKOS package. The GPU package exhibits much stronger weak scaling, with a near doubling of performance when adding a second GPU (84% improvement) and another major improvement when moving to 4 GPUs. This trend represents the vast efforts that went into making the LAMMPS GPU package useful at even the largest of scales.

## 7. CONCLUSIONS

The reported results from the paper were mostly reproduceable. The performance improvements can largely be seen in the optimized version of KOKKOS as shown in Figure 4. The error found and shown in Figures 2 and 3 is larger than is reported in the paper, which does not have any direct bearing on the work done in the paper, but does imply that optimizing the single precision version of the GPU code might result in poor accuracy that gets progressively worse with more timesteps. The performance is supposed to be portable and easy to move to another architecture; that performance improvement is shown to be there for a single Nvidia GPU using the KOKKOS-based optimization. It was not found to run across nodes, either by design flaw or test system configuration. Beyond this limitation, the optimizations introduced by the paper are both scalable and produce no change in relative accuracy of the Tersoff potential. The libraries built around GPU-based computation function reasonably well for this model, but do seem to accumulate error and do not yet support all of the functionality needed to perform the computation on the GPU.

## 8. ACKNOWLEDGMENTS

My thanks to David Kaeli from the Northeastern University Computer Architecture Research lab for his support in getting our team to the student cluster competition this year and all of the mentoring he has provided to us over the years; the Student Cluster Competition Committee for supporting this competition and providing us with the opportunity to work on applications like this one at SC17; and the rest of the Northeastern team for assistance in debugging system and software issues.

## 9. REFERENCES

[1] Hohnerbach, M., Ismail, A. E., Bientinesi, P. 2016. The Vectorization of the Tersoff Multi-Body Potential: An Exercise in Performance Portability. *Proceedings of the International Conference for High Performance Computing, Networking, and Storage, in: SC '16*. SLC '16. ACM, New York, NY, 526-531. DOI= http://dl.acm.org/citation.cfm?id=3014914.

[2] Brown, W. M., Wang P., Plimpton S. J., Tharrington A.N. 2011. Implementing Molecular Dynamics on Hybrid High Performance Computers – Short Range Forces. *Comp.~Phys.~Comm.*

[3] Brown, W. M., Wang P., Plimpton S. J., Tharrington A.N. 2012. Implementing Molecular Dynamics on Hybrid High Performance Computers – Particle-Particle Particle-Mesh. *Comp.~Phys.~Comm.*

[4] Brown, W. M., Wang P., Plimpton S. J., Tharrington A.N. 2013. Implementing Molecular Dynamics on Hybrid High Performance Computers – Three Body Potentials. *Comp.~Phys.~Comm.*

# 10. APPENDICES

## 10.1 Appendix A: Reproducing these results

The reproducibility test for the Tersoff Multi-Body Potential was done on a system using the CUDA Dev-Kit and GCC. The entire repository with modifications, run scripts and all output data can be found in the tersoff directory within the Github repository at https://github.com/ZachMarcus/SC17. See the build.sh file in https://github.com/ZachMarcus/SC17/tersoff/lammps-tersoff-vector/machines/reproduce/. The datasets tested were as provided by LAMMPS as well as the unsuccessful in.porter file provided as part of this competition and available at SC17/tersoff/INPUT. The run-time environment used was OpenMPI 3.0.0 with the specified CUDA 9.0 runtime and gcc 5.4.0. The experiments were all run on a system with AMD EPYC 7551 CPUs and NVIDIA Tesla V100 GPUs. Selecting older CUDA platforms is possible by modifying the compute capability in the build script. The directory SC17/tersoff/lammps-tersoff-vector/benchmarks/lammps contains the benchmark's scripts, modified for this particular system; the experiments are also all executed from that directory. The logs for the majority of the runs presented in this paper are also present in that directory structure.

## 10.2 Appendix B: Errors in Reproducing Results

There were several issues in reproducing the results during this competition using the KOKKOS and GPU packages.

The provided in.porter file uses functionality not available in those packages and efforts to have it complete those simulation steps using other packages were unsuccessful. An example output from such an attempt can be found at https://github.com/ZachMarcus/SC17/blob/master/tersoff/lammps-tersoff-vector/benchmarks/lammps/kokkos-porter.log.

Additionally, there was an error when attempting to scale to more than one MPI process, either on one node or across nodes, over the Infiniband or the Ethernet interconnect, when using the KOKKOS package.