Studying Performance Portability of LAMMPS across Diverse GPU-based Platforms

Nick Hagerty
Oak Ridge National Laboratory
Oak Ridge, TN, USA
hagertynl@ornl.gov

Verónica G. Melesse Vergara
Oak Ridge National Laboratory
Oak Ridge, TN, USA
vergaravg@ornl.gov

Arnold Tharrington
Oak Ridge National Laboratory
Oak Ridge, TN, USA
arnoldt@ornl.gov

***Abstract*—The molecular dynamics simulation software, LAMMPS, utilizes the Kokkos acceleration library to port computation to a diverse set of architectures including those based on GPU accelerators. In addition to Kokkos, LAMMPS contains a vast code base that leverages the CUDA application programming interface using library functions such as `cuFFT`, CUDA's fast-fourier transform (FFT) library, and, more recently, also support for AMD's Heterogeneous Interface for Portability (HIP) that is rapidly growing. While preparing LAMMPS tests for the AMD GPU-based test system precursors to Frontier, we investigated several strategies for accelerating LAMMPS on AMD GPUs, using the AMD Instinct MI100 and MI250X. In this work, we integrated the HIP FFT library, `hipFFT`, into the particle-particle particle-mesh (PPPM) long-range solver, which allowed the porting of PPPM calculations to the GPUs. Kokkos behavior on the MI100 and MI250X was also investigated through the `package kokkos` command of LAMMPS, targeting communication, memory usage, and particle grid decomposition. The Tersoff, Reax, Lennard-Jones (LJ), EAM, Granular, and PPPM potentials were investigated in this effort, and results from these experiments are provided. The selected potentials were run on Spock (AMD Instinct MI100), Crusher (AMD Instinct MI250X), AFW HPC11 (NVIDIA A100) and Summit (NVIDIA V100), for comparison. Operational roofline models were constructed and analyzed for the Tersoff, Reax, and Lennard-Jones potentials on Crusher and Summit.**

***Index Terms*—molecular dynamics, performance portability, GPU accelerated computing, roofline model**

I. Introduction

In recent years, the diversity of GPU-accelerated architectures has broaden considerably with vendors such as Advanced Micro Devices (AMD) and Intel joining NVIDIA to compete in this space and now providing devices designed with high performance computing (HPC) and artificial intelligence (AI) workloads in mind.

In 2019, the United States Department of Energy (DOE) announced Oak Ridge National Laboratory's Frontier supercomputer, expected to be the first Exascale system in the nation [1]. The Frontier system is an AMD GPU-based HPE/Cray EX supercomputer and, as a result, several new technologies have been developed to support scientific workloads on AMD GPU architectures. The ecosystem supporting AMD GPUs has opened new alternatives for application developers interested in running their application across platforms. AMD's Radeon Open Compute platform (ROCm) provides application programming interfaces including the Heterogenous Interface for Portability (HIP), libraries, and debugger and profiling tools. This ecosystem allows application developers to execute and optimize workloads on AMD GPUs.

In this work, we utilize LAMMPS, a popular open source molecular dynamics application, as a case study to understand its performance on different GPU-based platforms.

LAMMPS utilizes the Kokkos library to port calculations to a diverse set of architectures including those based on GPU accelerators. In addition to Kokkos, LAMMPS contains a vast code base that leverages the CUDA application programming interface using library functions such as cuFFT, and, more recently, also support for AMD's Heterogeneous Interface for Portability (HIP) that is rapidly growing.

While preparing LAMMPS tests for Spock and Crusher, the AMD GPU-based test systems precursors to Frontier, we investigated several strategies for accelerating LAMMPS on their two distinct AMD GPU architectures. Spock is a 36-node HPE Cray EX supercomputer powered by the AMD EPYC 7702 64-core processor with 4 AMD Instinct MI100 GPUs per node [2]. Crusher is a 192-node Test and Development System (TDS) powered by the AMD EPYC 7A53 64-core processor with 4 AMD Instinct MI250X GPUs per node [5]. The same tests were also executed on production Oak Ridge National Laboratory supercomputers based on NVIDIA GPUs including Summit, the United State's fastest supercomputer as of the November 2021 TOP500 list [3] comprised of 4,680 nodes with two IBM POWER9 22-core processors and six NVIDIA V100 GPUs, and the Air Force Weather HPC11 system [4] which is comprised of two clusters each with 800 nodes powered by the AMD EPYC 7742 64-core processor and a GPU-partition with 20 nodes that, in addition, include 4 NVIDIA A100 GPUs per node.

The tests executed include liquid, metal, granular, biological, and polymer systems up to 55 million atoms in size, evaluating the Reax, Lennard-Jones, EAM, Granular, Tersoff, and particle-particle particle-mesh (PPPM) potentials. The

selected LAMMPS potentials were run on Spock, Crusher, AFW HPC11, and Summit, in order to collect sufficient data to better understand the performance of the application in a diverse set of architectures.

As part of this work, we implemented hipFFT, which is AMD's HIP fast fourier transform (FFT) library into the PPPM long-range solver, which allowed us to port the PPPM calculations to AMD GPU architectures. Prior to hipFFT integration, the PPPM calculations were constrained to running on the host using the LAMMPS default (KISS FFT) or FFTW, and the simulation was forced to send data between the CPU and GPU at every timestep.

Kokkos behaviors in each potential were then evaluated using combinations of values of the LAMMPS `package kokkos` flags `neigh`, `neigh/thread`, `pair/only`, `comm`, `newton`, and `pair/qeq` (Reax only). Targeted Kokkos behaviors include memory and thread utilization, communication procedures, and neighbor list construction. LAMMPS provides several options for the user to take advantage of memory characteristics such as cache levels, memory clock frequency, memory bandwidth, and threading behavior through the Kokkos library. These options enable the user to customize simulation behavior for GPU architectures.

The AMD profiler, `rocprof`, and NVIDIA NSight Compute profiler, `ncu`, were used to conduct performance analysis on Crusher and Summit. Performance metrics targeting memory accesses and floating-point operations were gathered, and the results were used to evaluate experiments and inform further areas of optimization. These results are aggregated into operational roofline models [11], which present achieved floating-point performance.

This work presents an overview of performance portability of the LAMMPS molecular dynamics code across NVIDIA and AMD GPU-based architectures. The findings observed for the different potentials, as well as the methodology and optimizations explored for each architecture will be useful for LAMMPS users as well as the HPE/Cray EX supercomputer community interested in porting applications across NVIDIA and AMD GPU-accelerated platforms.

II. Related Work

Roofline models are an increasingly popular method of visualizing performance capability on a multicore CPU or on a GPU. As such, there are many variations of rooflines. For this work, we selected the operational roofline and focus on the double-precision floating point operations (FLOPS) performed. The operational roofline is first described in 2009 by Williams et al. [11]. The peak roofline is constructed by Equation 1, where `MemBW` is the bandwidth of the off-chip memory (DRAM or HBM) and `OpIntens`, operational intensity, is the ratio of double-precision floating point operations to bytes moved.

$$FLOPS/s_{attainable} = MIN \begin{cases} FLOPS/s_{peak} \\ MemBW \times OperIntens \end{cases} \quad (1)$$

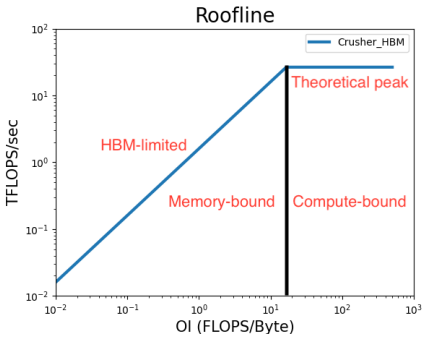

Fig. 1. Theoretical operational roofline

Then, for each kernel of interest, we calculate the achieved FLOPS/s and operational intensity, and compare it to the theoretical operational roofline. Figure 1 shows an annotated theoretical operational roofline. To the left of the vertical black line, the bandwidth of the GPU high-bandwidth memory (HBM) is the limiting factor on performance, and the kernel is termed "memory-bound". To the right, a kernel is labeled "compute-bound".

Hierarchical roofline models include the theoretical rooflines for various levels of memory in the memory-bound region, including caches [12]. We do not use hierarchical roofline models in this study, as profiling cache performance is an evolving capability on AMD GPU architectures.

To extract performance counters from AMD GPU architectures, we follow a process similar to Leinhauser et al. [10] which uses the `SQ_INSTS_VALU` and `SQ_INSTS_SALU` instruction counters, and `FETCH_SIZE` and `WRITE_SIZE` for memory analysis. Since that publication, `rocprof` now provides a more detailed instruction breakdown through `SQ_INSTS_VALU_{ADD,MUL,FMA}_FP{16,32,64}`, which we use in this work, in place of `SQ_INSTS_VALU` and `SQ_INSTS_SALU`. The metrics required to calculate `FETCH_SIZE` and `WRITE_SIZE` are directly available in `rocprof`, so we query those, and calculate the fetch and write sizes while post-processing. Our roofline model is different from that of [10], as they employ an instruction roofline model, first developed by Ding et al. [9], which evaluates performance by the instructions issued, not by the operations performed. Evaluating by instructions issued targets identifying fetch-decode-issue bottlenecks, and seeks to evaluate performance when there are a large number of integer instructions [9].

III. Experimental Setup

This study utilized four Oak Ridge National Laboratory (ORNL) supercomputers; Summit, AFW HPC11, Spock, and Crusher. Table I summarizes the GPU architectures for each of these machines.

A. Summit

Summit is an IBM POWER9 supercomputer with more than 4,600 compute nodes, currently ranked the number two fastest

TABLE I
SUMMARY OF GPU ARCHITECTURES USED

Machine	GPU Model	Memory Size
Summit	NVIDIA V100	16 GB
AFW HPC11	NVIDIA A100	40 GB
Spock	AMD Instinct MI100	32 GB
Crusher	AMD Instinct MI250X	64 GB (1-GCD)

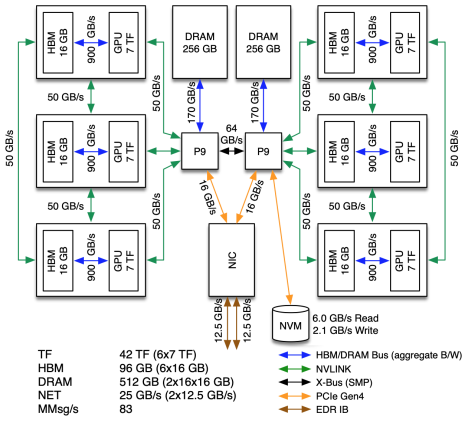

Fig. 2. Summit node architecture

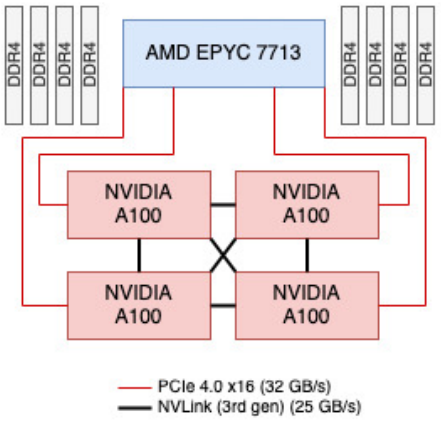

Fig. 3. AFW HPC11 GPU node architecture

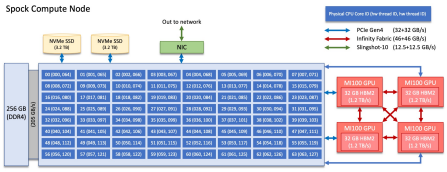

Fig. 4. Spock node architecture

supercomputer in the world on the most recent TOP500 list (Nov 2021) [3]. Each compute node has two 22-core POWER9 CPUs and six NVIDIA Tesla V100 GPUs. The GPUs on each compute node are interconnected via NVLink 2.0, with 50 GB/s peak bandwidth between devices. Each POWER9 CPU is connected via NVLink 2.0 to three of the six GPUs. Summit's node architecture is shown in Figure 2 [8].

The NVHPC v21.11 toolkit was used to compile and profile LAMMPS on Summit for this study.

B. HPC11

The Air Force Weather (AFW) HPC11 system consists of two independent clusters, Fawbush and Miller, each with 800 nodes comprised of a single AMD EPYC 7713 64-core processor. In addition, there is a GPU-based partition which consists of 20 nodes, powered by one AMD EPYC 7713 64-core processor and four 40 GB NVIDIA A100 GPUs. The GPUs on each compute node are interconnected in all-to-all arrangement via NVLink2.0, providing device-to-device bi-directional bandwidths of 25 GB/s. The AMD EPYC processor is connected to GPUs via PCIe v4, with a bi-directional bandwidth of 32 GB/s. HPC11's GPU-partition compute node architecture is shown in Figure 3.

The NVHPC v21.3 toolkit was used to compile LAMMPS on HPC11.

C. Spock

Spock is a 36-node HPE/Cray EX early access precursor to Frontier. Spock is powered by one AMD EPYC 7702 64-core processor with four 32GB AMD Instinct MI100 GPUs per node [2]. Each MI100 consists of 120 compute units (analogous to an NVIDIA streaming multiprocessor) and 7680 total cores. The GPUs on each compute node are interconnected in all-to-all arrangement via Infinity Fabric, providing device-to-device bi-directional bandwidths of 92 GB/s. The AMD EPYC processor is connected via PCIe Gen4 to all GPUs, providing bi-directional bandwidths of 64 GB/s. Spock's node architecture is shown in Figure 4.

The HIP C++ compiler wrapper, `hipcc`, provided by ROCm v4.5.0, was used to compile LAMMPS on Spock.

D. Crusher

Crusher is a 192-node Test and Development System (TDS) precursor to Frontier, powered by one AMD EPYC 7A53 64-core processor and four AMD Instinct MI250X GPUs per node. Each MI250X GPU contains two graphics compute dies (GCDs), and is seen as two distinct GPUs by applications and schedulers. Each GCD contains 64 GB high-bandwidth memory (HBM), accessed at 1.6 TB/s. GCDs within an MI250X GPUs are connected via Infinity Fabric, with a peak bandwidth of 200 GB/s. GCDs on different MI250X GPUs are connected via Infinity Fabric GPU-GPU, as shown in Figure 5, with peak bandwidth of up to 100 GB/s, based on the number of Infinity Fabric connections between GCDs [5].

The HIP C++ compiler wrapper, `hipcc`, provided by ROCm v4.5.0, was used to compile LAMMPS on Crusher.

IV. EXPERIMENTAL RESULTS

Following changes to port the LAMMPS PPPM solver to AMD GPUs, we tested and analyzed several common

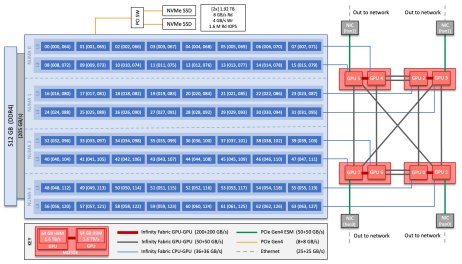

Fig. 5. Crusher node architecture

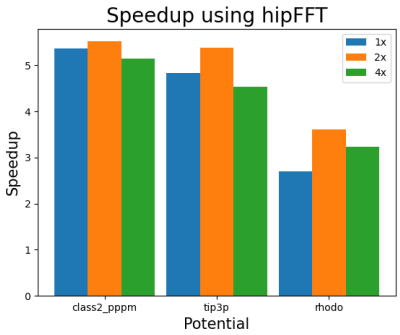

Fig. 6. Performance improvement of hipFFT-accelerated PPPM

TABLE II
SUMMARY OF POTENTIALS EVALUATED

Potential	# atoms (base)	Avg # neighs/atom
TIP3P water	45K	1405
CLASS2+PPPM polymer	42K	564
ReaxFF	30K	667
Tersoff*	32K	16.6
Rhodo*	32K	447
LJ liquid*	32K	76.9
EAM metal*	32K	75.5
Chute granular flow*	32K	376

TABLE III
SUMMARY OF KOKKOS FLAGS EXAMINED

Flag name	Values	Description
neigh	half,full	Full or half neighbor list
neigh/qeq	half,full	Size of neighbor list for fix/qeq
neigh/thread	on,off	Thread over atoms and neighbors
newton	on,off	Set Newton pairwise, bonded flags
comm	no,device,host	Location of packing/unpacking of data
pair/only	on,off	Use device only for pair styles

LAMMPS potentials using one CPU core and one GPU (one GCD for AMD Instinct MI250X).

We constructed a diverse set of models, comprised of several LAMMPS-provided benchmarks and some common models. The systems are summarized in Table II. An asterisk indicates LAMMPS-provided benchmarks.

These potentials assess a wide range of the average neighbors per atom. Our tests replicated each system in 3 dimensions (2 dimensions for Chute) by 1 (base), 2, 4, 6, 8, and 12, or until each system exceeded device memory. All tests were averaged over 5 runs. Results are presented in units of `million-atom * steps / second`.

A. Porting PPPM Solver to AMD GPU architectures

The particle-particle particle-mesh (PPPM) solver maps atom charges to a 3D mesh, then uses 3D fast fourier transforms (FFTs) to solve Poisson's equations on the 3D mesh [6]. LAMMPS uses the built-in KISS FFT by default, but vendor-provided FFT libraries like cuFFT and FFTW can replace KISS FFT. LAMMPS stable releases do not support hipFFT in PPPM as of April 2022, so PPPM is constrained to running on the CPU on AMD GPU architectures. We implemented hipFFT into the KOKKOS package PPPM source. The performance improvement from hipFFT was evaluated using a TIP3P water model, the LAMMPS Rhodo benchmark, and a Class2 Lennard-Jones polymer model. The speedup for these potentials as a function of model size is shown in Figure 6. Each color represents how many times the system was replicated in the x, y, and z directions. Base (1x) system sizes are between 32K and 45K atoms.

The increase in speedup from 1x to 2x system sizes is discussed in depth in subsequent sections, and is attributed to under-utilization of the GPU threads, and the increased significance of overhead incurred from communication, proportional to elapsed time.

Prior to the addition of hipFFT, FFT computations had to be performed on the CPU, which required communication from the GPU to CPU at every timestep.

B. Kokkos Package Parameter Study

LAMMPS provides several options to customize the workload sent through Kokkos through the `package` command [7]. Table III lists the Kokkos flags examined in this study.

There are several constraints on combinations of flags. The `neigh/qeq` flag is only used for Reax tests. The `neigh/thread on` flag can only be used with `neigh full`. The Chute, Rhodo, Reax, and Tersoff benchmarks all require `neigh half`, so `neigh/thread on` is not possible for these benchmarks. Reax and Tersoff also require `newton on`. Each valid combination of flags was tested on a simulation of at least 100 steps, and the `timesteps/s` were gathered. To compare performance for a specific parameter combination, we computed `million-atom * steps / s` for each model size. This value is a good indication of the simulation-based throughput attainable by the GPU. For each potential, we selected one system size to discuss results for, summarized in Table IV. We analyzed the performance of each Kokkos parameter, and summarize our findings below.

1) Kokkos parameter: neigh and neigh/thread: The `neigh` keyword specifies how neighbor lists are built, and the `neigh/thread` keyword determines if the Kokkos package threads over only atoms, or over atoms and their neighbors. `neigh half` utilizes a thread-safe implementation of half-neighbor lists, while `full` utilizes full neigbor lists. To use `neigh/thread on`, we must use full neighbor lists. The

TABLE IV
SUMMARY OF SYSTEM SIZES PRESENTED

Potential	System Size Reported
LJ	6.91 mil atoms
Tersoff	2.05 mil atoms
Chute	4.6 mil atoms
Rhodo	2.05 mil atoms
EAM	2.05 mil atoms
Class2+PPPM	340K atoms
TIP3P	360K atoms
ReaxFF	238K atoms

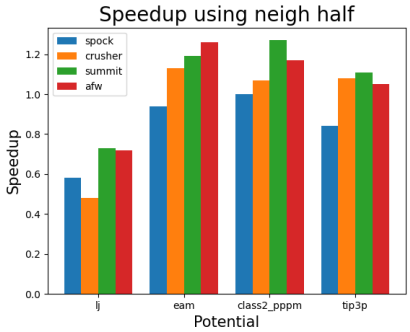

Fig. 7. Performance improvement of using neigh half

Chute, Rhodo, Reax, and Tersoff potentials all require `neigh half`, so those potentials are excluded from these results. `neigh full` is the default when running on the GPU, and `neigh/thread on` is default only when less than 16K atoms per MPI rank are used, which is a condition not met by any system in this study. The average neighbors per atom for each potential can be found in Table II.

Figure 7 shows the performance improvement from using `neigh half` relative to `neigh full` with threading off for each potential, with the additional Kokkos flags `pair/only off neigh/thread off newton off comm device`.

Figure 8 shows the performance improvement for `neigh full neigh/thread on`, relative to threading off, with the additional Kokkos flags `pair/only off newton off comm device`.

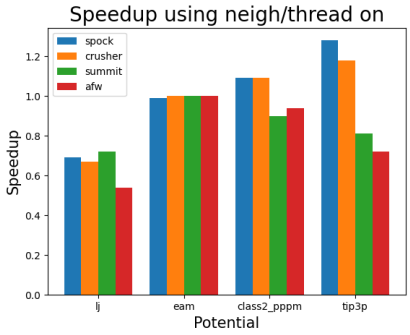

Fig. 8. Performance improvement of using neigh/thread on

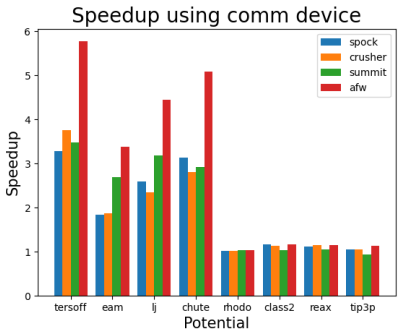

Fig. 9. Performance improvement of using comm device

The potentials in Figure 7 and Figure 8 are ordered from left to right by increasing average neighbors per atom. For Spock and Crusher, we can see that the speedup steadily increases as we move from left to right (increasing average neighbors per atom), while Summit and HPC11 do not exhibit this same behavior. This is especially visible in Figure 8, where Spock and Crusher achieve a 1.2x speedup when using `neigh/thread on`, while Summit and AFW observe a 20% slowdown. Summit and AFW did not benefit from `neigh/thread on` for any of the sampled potentials.

2) Kokkos parameter: comm: The `comm` keyword controls whether the host or device performs unpacking and packing when communicating per-atom data between processors. The `comm` keyword controls the value of all `comm` sub-keywords, such as `comm/forward`, which are detailed in the LAMMPS `package` command manual page [7]. We did not investigate every `comm` sub-keyword, but rather set all `comm` sub-keywords to the value of `comm`, which is the default behavior. The `comm` keyword can hold values of `no` (use standard single-thread non-Kokkos methods), `device` (use the device), or `host` (use the host, multi-threaded). On GPUs, the default is `comm device`. Figure 9 shows the performance improvement of `comm device`, compared to `comm no`, using the additional Kokkos flags listed in Table V. `comm device` was shown to be optimal in nearly all experiments, except for the smallest system size of Reax, where `comm no` achieves a 4% speedup over `comm device` on Summit.

3) Kokkos parameter: neigh/qeq: The `neigh/qeq` keyword determines how neighbor lists are built for `fix qeq/reaxff/kk` fixes, which are used in the Reax potential in this study. Reax requires `neigh half`, but `neigh/qeq` can be full or half. For all machines and system sizes, `neigh/qeq full` was found to be the optimal setting. Figure 10 shows the speedup from using `neigh/qeq full`, relative to `neigh/qeq half`, with the additional Kokkos flags `neigh half pair/only off neigh/thread off newton on comm device`. The 2-million atom system exceeds device memory on Spock, Summit, and AFW.

TABLE V
KOKKOS FLAGS USED TO EVALUATE `COMM DEVICE`

Potential	Kokkos flags
LJ	neigh full neigh/thread off newton off pair/only off
Tersoff	neigh half neigh/thread off newton on pair/only off
Chute	neigh half neigh/thread off newton off pair/only off
Rhodo	neigh half neigh/thread off newton off pair/only off
EAM	neigh half neigh/thread off newton off pair/only off
Class2+PPPM (Crusher, Spock)	neigh full neigh/thread on newton off pair/only off
Class2+PPPM (Summit, AFW)	neigh half neigh/thread off newton off pair/only off
TIP3P (Crusher, Spock)	neigh full neigh/thread on newton off pair/only off
TIP3P (Summit, AFW)	neigh half neigh/thread off newton off pair/only off
ReaxFF	neigh half neigh/qeq full newton on pair/only off

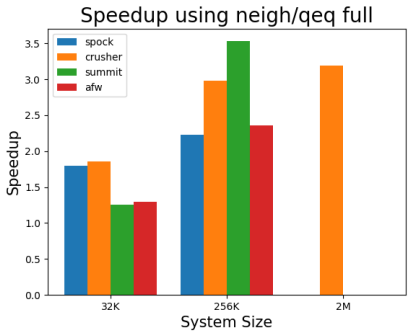

Fig. 10. Performance improvement of using neigh/qeq full

TABLE VI
KOKKOS FLAGS FOR ROOFLINE ANALYSIS

Potential	Kokkos flags
LJ	neigh full neigh/thread off newton off pair/only off
Tersoff	neigh half neigh/thread off newton on pair/only off
ReaxFF	neigh half neigh/qeq full newton on pair/only off

C. Roofline Performance Analysis

After analyzing LAMMPS `package kokkos` parameters, we performed roofline analysis on 3 potentials - Lennard-Jones, Tersoff, and Reax [9], [11]. We limited our analysis to double-precision floating-point operations and high-bandwidth memory (HBM) accesses.

Table VI provides the `package kokkos` flags used in the LAMMPS launch command for roofline experiments of Crusher and Summit.

1) Computing kernel performance on AMD GPUs: We used ROCm v4.5.0 on Crusher to compile and profile LAMMPS. AMD's profiler, `rocprof`, provides instruction counters for double-precision floating point instructions in a `per-simd` context. To translate these instruction counters to floating point operations and normalize to a global context on an AMD Instinct MI250X, we use Equation 2. For roofline models that use issued instructions as a measure of performance instead of performed operations, the `per-simd` instructions are only multiplied by the number of SIMD units, 4 on the MI250X.

$$FLOPS_{global} = Instr_{per-simd} \times 4\ SIMD \times 16 \frac{cores}{SIMD} \quad (2)$$

To profile double-precision floating point operations, we query the `SQ_INSTS_VALU_{ADD,MUL,FMA}_F64` metrics. We sum these metrics, weighting the `FMA` instruction by 2, to find the total double-precision floating point instructions issued. Then we use Equation 2 to calculate the number of operations performed.

To gather HBM usage, we use the `TCC_EA_{RDREQ,WREQ}` metrics, which count the number of memory transactions performed between the L2 cache and HBM. Memory transactions are either 32 bytes or 64 bytes. The `RDREQ` offers two suffixes, `_32B` and `_sum`, and `WREQ` offers `_64B` and `_sum`. The metrics with the suffix of `_32B` are 32-byte transactions and the suffix of `_64B` are 64-byte transactions. The number of 32-byte write requests can be calculated by subtracting the number of 64-byte write requests from the sum of all write requests. The number of 64-byte read requests can be calculated in a similar manner. Equation 3 calculates the number of bytes read from HBM, and Equation 4 calculates the number of bytes written to HBM. The sum of these two equations yields the total bytes moved.

$$\begin{aligned} BytesRead = {} & 32 \times TCC\_EA\_RDREQ\_32B \\ & + 64 \times (TCC\_EA\_RDREQ\_sum \\ & - TCC\_EA\_RDREQ\_32B) \end{aligned} \quad (3)$$

$$\begin{aligned} BytesWrite = {} & 64 \times TCC\_EA\_WREQ\_64B \\ & + 32 \times (TCC\_EA\_WREQ\_sum \\ & - TCC\_EA\_WREQ\_64B) \end{aligned} \quad (4)$$

Using these metrics, we can locate a kernel's achieved performance relative to the theoretical roofline. The x-axis of the roofline model is operational intensity, which we calculate for each kernel by Equation 5. The y-axis of the roofline is performance, which we calculate for each kernel by Equation 6.

$$OI(FLOPS/Byte) = \frac{FLOPS}{BytesWrite + BytesRead} \quad (5)$$

$$Performance(FLOPS/s) = \frac{FLOPS}{elapsed\_seconds} \quad (6)$$

2) Computing kernel performance on NVIDIA GPUs: We used NSight Compute v2021.3.0.0 on Summit to gather metrics for roofline analysis. NSight Compute provides the `ncu` profiler, and by using the flags `--set full -k kernel_name`, NSight Compute gathers the metrics required for roofline profiling for the kernels with the specified name. We loaded the generated `ncu-rep` file into NSight Compute GUI, which constructed the floating point operational roofline model for each kernel. NSight Compute also provides a `-k kernel_name` flag, which allows the user to specify the name of the kernel to profile. Unfortunately, NSight Compute sees all Kokkos-launched kernels as *cuda_parallel_launch_{local,constant}_memory*, and thus cannot be filtered by a string such as `PairTersoff` to only profile Tersoff pair style kernels. The "mangled" kernel name, which is provided in NSight Compute GUI, is more than 100 characters long and cannot be filtered on using the `-k` option. Due to this limitation, all Kokkos kernels were profiled, which greatly increased the walltime of the profiling experiments. The desired kernels were then hand-selected from NSight Compute GUI. The overhead for the Reax potential increased beyond the walltime limits available on the Summit batch queue, so Reax is excluded from roofline profiling on Summit.

We calculated the operational intensity and performance using the metrics provided to confirm that the roofline provided by NSight Compute GUI is computed the same way as we computed for AMD. NSight Compute GUI provides instruction counters for `DADD`, `DMUL`, and `DFMA`. We sum these metrics together, weighting `DFMA` by 2, as we did for Crusher, and multiply by the warp size, 32, to calculate the number of double-precision floating point operations performed, and calculate achieved performance using Equation 6.

NSight Compute GUI also builds memory tables, which show how many bytes of data are moved between caches and HBM. NSight Compute directly provides the information needed for our roofline through `dram__bytes_read.sum` and `dram__bytes_write.sum`. As with AMD, we use Equation 5 to calculate the operational intensity.

3) Roofline results: We profiled several sizes of the Lennard-Jones, Tersoff, and Reax benchmarks on Crusher and Summit for roofline analysis. Figure 11 shows the operational roofline for these potentials on Crusher.

Reax has the highest operational intensity of the three sampled potentials, followed by Tersoff, then Lennard-Jones. As system sizes increase, the operational intensity decreases, but the achieved performance of the kernel may increase before eventually decreasing. This moves the data point closer to the memory-bound region of the theoretical roofline. Tersoff and Lennard-Jones distinctly display this behavior, most visibly from the smallest system size to the second-smallest. Supplementing this observation, we also noticed an increase

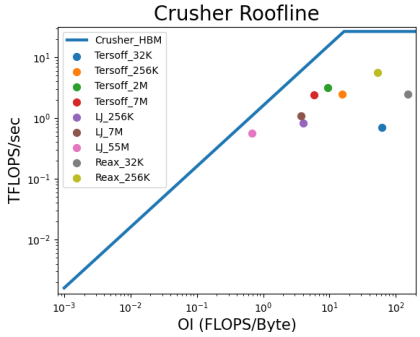

Fig. 11. Roofline plot of Crusher

in the `million-atom steps / s` when systems were initially enlarged from their base size, followed by a decrease, as systems became too large to efficiently balance on a single GPU.

It is worth noting that the Tersoff benchmark requires the use of half neighbor lists, which utilize atomics to maintain thread-safe behavior. The Reax benchmark uses 2 neighbor lists, one is a half neighbor list for the `reax/c/kk` pair style, and the other is a full neighbor list for `fix qeq/reax/kk`.

The Reax potential resides in the compute-bound region of the roofline, and the Tersoff potential starts in the compute-bound region at the smallest system size, before transitioning to the memory-bound region for larger systems. The Lennard-Jones potential is also found in the memory-bound region, and has the lowest operational intensity of the three potentials. The Reax 256K-atom system achieves about 20% of peak floating point performance on a single GCD on Crusher with 5.6 TFLOPS/s.

It is worth noting that these are preliminary results on the Crusher test and development system for Reax, Tersoff, and Lennard-Jones potentials. It is evident in the roofline model that further work is needed to continue improving the performance of these particular kernels and fully optimize them for the system. In addition, it is important to keep in mind that Crusher's software stack is rapidly evolving and as it matures, new features and improvements are continuously added. Similarly, LAMMPS is actively working on optimizations for AMD GPU architectures and although the modifications thus far already show improvement, we expect further improvements as the system and the software stack matures. Figure 12 shows the performance improvement on small, medium, and large systems using the Lennard-Jones, Tersoff, and Reax potentials, comparing the September 2021 release of LAMMPS to the development branch, February 2022.

Figure 13 shows the operation roofline for the selected potentials on Summit (**Note:** Reax is not present in this roofline). The overhead from profiling all kernels greatly increased the runtime of Reax on Summit, beyond what is reasonable for a batch queue, even for a 10-step simulation on a 32K atom system.

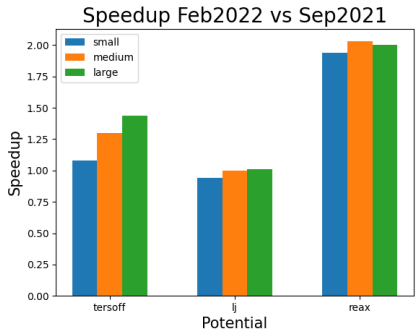

Fig. 12. Speedup of LAMMPS potentials, Sep 2021 to Feb 2022

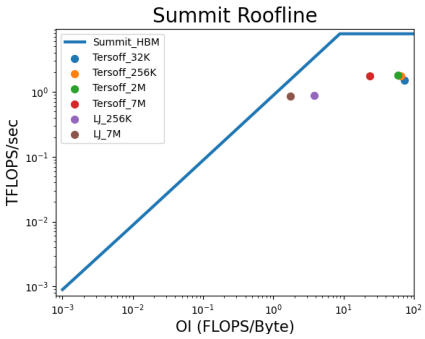

Fig. 13. Roofline plot of Summit

The Tersoff potential reaches the compute-bound region of the roofline for all system sizes on Summit and achieves about 20% of the peak double-precision floating point performance. The Tersoff potential, as mentioned above, requires half neighbor lists when run with Kokkos, which maintain thread-safe behavior through atomics.

Summit displays the same pattern as Crusher, where increasing the size of the system results in a decrease of the operation intensity, accompanied by a slight temporary increase in the achieved performance.

V. Lessons Learned

As expected when studying a scientific application on a new architecture, we encountered several difficulties that resulted in lessons learned that we considered would be valuable for users attempting similar efforts. This section highlights some of these challenges and presents our recommendations.

- Due to the fact that AMD ROCm is a relatively new software platform, the libraries and tools are in active development. As a result, gathering performance metrics from AMD GPU hardware is not well-established in literature yet, and with all the variations of roofline models that exist (instructional, operational, hierarchical), the process required us to customize some of the metrics to suit the available information. After selecting instruction counters and memory metrics from `rocprof --list-basic`, we profiled simple kernels on Crusher to confirm that the metrics we gathered are calculated as expected. We then cross-referenced this with profiling output from NSight Compute on Summit to ensure correct and identical roofline computation between AMD and NVIDIA architectures. Understanding some of the memory metrics and their meaning, like `TCC_EA_WREQ_sum`, required additional searching and experimentation.
- The ROCm compiler requires several optimizations to be turned on manually, such as `-munsafe-fp-atomics`, which applies to the MI250X GPU architecture. This flag is safe under many common atomic conditions, like those found in most LAMMPS kernels, but is off by default. Adding this compilation flag to the LAMMPS build resulted in a 30% improvement in performance of the Tersoff potential, and this flag is used in this work. For a double-precision "unsafe" atomic, it is guaranteed safe if it is 8-byte aligned, and if it resides in a coarse-grained allocation.
- Since ROCm is in active development, there are frequent releases, which sometimes move ahead of the HPE/Cray Programming Environment. We used the latest HPE/Cray-supported ROCm in this study, ROCm v4.5.0.
- NSight Compute was unable to separate out kernels launched by Kokkos in LAMMPS, the kernel name visible to `ncu` was the same for all Kokkos-launched kernels. The result was profiling all Kokkos kernels, then selecting our desired kernels in NSight Compute GUI while post-processing the data. But, for a computationally dense potential like Reax, the walltime increased beyond batch queue limits.
- The sub-keywords of `package kokkos comm` are numerous, but may hold a significant benefit for performance tuning of Kokkos.

VI. Conclusion

While preparing LAMMPS tests for Spock and Crusher, the AMD GPU-based test systems precursors to Frontier, we investigated performance behaviors of LAMMPS using Kokkos on their two distinct AMD GPU architectures. We examined the performance of LAMMPS on NVIDIA V100 and A100 architectures, and AMD Instinct MI100 and MI250X architectures.

Experiments included liquid, metal, granular, biological, and polymer systems up to 55 million atoms in size, evaluating the Reax, Lennard-Jones, EAM, Granular, Tersoff, and particle-particle particle-mesh (PPPM) potentials.

We first implemented hipFFT into the PPPM long-range solver, which allowed us to port the PPPM calculations to AMD GPU architectures. The speeds of a TIP3P water model and Lennard-Jones Class2 polymer model utilizing PPPM improved by up to 6x with the implementation of hipFFT.

Kokkos behaviors in each potential were then evaluated using combinations of values of the LAMMPS `package kokkos` flags `neigh`, `neigh/thread`, `pair/only`, `comm`, `newton`, and `pair/qeq` (Reax only).

By customizing Kokkos behavior to take advantage of these strengths, simulation speed was improved by more than 20%

on the MI100 and 25% on the MI250X, compared to the speed using default settings. For example, a 360,000-atom TIP3P water model improved by 18% on the MI100 when using the keyword `neigh/thread on`.

The AMD profiler, `rocprof`, and NVIDIA NSight Compute profiler, `ncu`, were used to conduct performance analysis on Crusher and Summit. Performance metrics targeting memory accesses and double-precision floating-point operations were gathered, and the results were used to evaluate experiments and inform further areas of optimization. These results are aggregated into operational roofline models [11], which present achieved floating-point performance. Operational rooflines were constructed for the Tersoff, Reax, and Lennard-Jones potentials.

This work presents an overview of performance portability of the LAMMPS molecular dynamics code across NVIDIA and AMD GPU-based architectures. The findings observed for the different potentials, the methodology utilized in developing operational roofline models on NVIDIA and AMD GPUs, as well as optimizations explored for each architecture will be useful for LAMMPS users as well as the HPE/Cray EX supercomputer community interested in porting applications across NVIDIA and AMD GPU-accelerated platforms.

ACKNOWLEDGMENT

The authors would like to thank Sunita Chandrasekaran and Nick Curtis for their valuable insights in the development of operational roofline models, and Stan Moore for the feedback provided while developing this suite of LAMMPS tests.

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

REFERENCES

[1] . https://www.hpcwire.com/2019/05/07/cray-amd-exascale-frontier-at-oak-ridge/, 2019.

[2] Spock quickstart guide. https://docs.olcf.ornl.gov/systems/spock_quick_start_guide.html, 2021.

[3] TOP500 Supercomputer Sites. https://www.top500.org, 2021.

[4] AFW HPC-11 User Documentation. https://docs.afw.ornl.gov, 2022.

[5] Crusher quickstart guide. https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html, 2022.

[6] LAMMPS user manual, kspace_style command. https://docs.lammps.org/kspace_style.html, 2022.

[7] LAMMPS user manual, package command. https://docs.lammps.org/package.html, 2022.

[8] Summit quickstart guide. https://docs.olcf.ornl.gov/systems/summit_user_guide.html, 2022.

[9] Ding, Nan and Williams, Samuel. An Instruction Roofline Model for GPUs. In *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, 2019.

[10] Matthew Leinhauser, René Widera, Sergei Bastrakov, Alexander Debus, Michael Bussmann, and Sunita Chandrasekaran. Metrics and design of an instruction roofline model for amd gpus, 2021.

[11] Williams, Samuel, Waterman, Andrew, and Patterson, David. An Insightful Visual Performance Model for Multicore Architectures. In *Communications of the ACM*, volume 52, pages 65–76, April 2009.

[12] Charlene Yang, Kurth, Thorsten, and Williams, Samuel. Hierarchical Roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system. *Concurrency Computational Pract and Exper*, 2020.