ARL-TR-8018 ● MAY 2017

US Army Research Laboratory

Accelerating Calculations of Reaction Dissipative Particle Dynamics in LAMMPS

by Christopher P Stone, Timothy I Mattox, James P Larentzos, and John K Brennan

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-8018 ● MAY 2017

US Army Research Laboratory

Accelerating Calculations of Reaction Dissipative Particle Dynamics in LAMMPS

by Christopher P Stone
Computational Science and Engineering, LLC, DOD HPCMP PETTT, APG, MD

Timothy I Mattox
Engility Corporation, DOD HPCMP PETTT, APG, MD

James P Larentzos and John K Brennan
Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
May 2017	Technical Report	September 2015–December 2016

4. TITLE AND SUBTITLE
Accelerating Calculations of Reaction Dissipative Particle Dynamics in LAMMPS

5a. CONTRACT NUMBER
PP-CCM-KY07-005

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Christopher P Stone, Timothy I Mattox, James P Larentzos, and John K Brennan

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
Weapons and Materials Research Directorate
Energetic Materials Science Branch (ATTN: RDRL-WML-B)
Aberdeen Proving Ground, MD 21005

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-8018

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
High Performance Computing Modernization Program (HPCMP)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Reaction Dissipative Particle Dynamics (DPD-RX) is a promising coarse-graining (CG) method for modeling energetic materials at the mesoscale. The LAMMPS DPD-RX multiscale-modeling software combines stochastic particle dynamics with intra-particle chemical kinetics. The chemical kinetics model requires the solution of a system of ordinary differential equations (ODEs) within each CG particle at each time step. The ODE solutions are computationally intensive and exceed 99% of the run time for some cases. Several acceleration methods were tested for the chemical kinetics DPD-RX component including different ODE solver methods (implicit vs. explicit), parallel programming paradigms (MPI vs. OpenMP vs. GPU), and matrix storage representations (dense vs. sparse). For a small, reduced-order reaction mechanism, the best acceleration was 6.1 times. For a larger, more chemically detailed mechanism, the best acceleration exceeded 60 times the baseline performance. This level of acceleration enables the use of higher fidelity reaction mechanisms, which have a broader modeling applicability.

15. SUBJECT TERMS
DPD, LAMMPS, hybrid/heterogeneous/accelerated algorithms, numerical methods, linear and nonlinear systems, computational materials science and engineering, multiscale modeling, GPU, ODE solvers, dissipative particle dynamics, coarse-graining

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			James P Larentzos
Unclassified	Unclassified	Unclassified	UU	34	19b. TELEPHONE NUMBER (Include area code) 410-306-0809

Contents

List of Figures

Acknowledgments

The authors acknowledge the contribution of Dr Martin Lísal (Hála Laboratory of Thermodynamics, Institute of Chemical Process Fundamentals of the Advanced Scientific Computing Research and Department of Physics, JE Purkinje University, Ústí nad Labem, Czech Republic) for his contribution to the Reaction Dissipative Particle Dynamics software.

This study was supported by the US Department of Defense High Performance Computing Modernization Program User Productivity Enhancement, Technology Transfer, and Training (PETTT) activity (General Services Administration Contract No. GS04T09DBC0017 through Engility Corporation).

1. Introduction

Dissipative Particle Dynamics (DPD) is a well-established coarse-graining (CG) technique for simulating materials at the micro- and mesoscale. A novel extension of this simulation capability is a unique mesoscale description of chemical reactivity that occurs within the local volume of the CG particles.[1] In particular, the extension of Reaction DPD (DPD-RX) into LAMMPS[2] is a critical advance in capability for modeling the multiscale nature of the energy release and propagation mechanisms in advanced energetic materials (EM).[3] Decomposition and energy release of EMs occurs at the molecular scale, yet the explosive response manifests at the macroscale. The microstructural heterogeneities that dictate the response of composite EMs to various stimuli require multiscale modeling at length and time scales that are far beyond those amenable to both quantum mechanical and atomistic simulation approaches.

Despite the tremendous gain in modeling capability enabled by our prior work,[3] the time-to-solution of DPD-RX simulations must be further reduced to capture the complete chemically reacting response of EMs. DPD simulations of EMs are computationally taxing and routinely require tracking $O(10^{7\text{-}8})$ CG particles to examine microstructural heterogeneities. Thus, the primary goal of this study is to investigate methods to optimize and accelerate the calculations of the DPD-RX software to better utilize high performance computing (HPC) resources and exploit emerging, heterogeneous architectures (e.g., co-processors and graphics processing units [GPUs]), while enabling EM simulations at previously inaccessible scales.

A principle feature of DPD-RX is its ability to model chemical reactions within each CG particle. The change in composition due to chemical reactions is described by a system of ordinary differential equations (ODEs) that are evaluated at each DPD time step. These ODE systems are local to each particle and can be solved using a variety of ODE solver methods. The ODE system for each particle is written as

$$\frac{dy_k}{dt} = V_{CG} \sum_i^{NR} \nu_{ki} \left[r^f - r^b\right]_i , \tag{1}$$

where y_k is the number of moles of the k^{th} species within the *CG* particle, ν_{ki} is the net stoichiometric coefficient, and $[r^f - r^b]_i$ is the rate-of-progress of the i^{th} reaction of *NR* total reactions, and V_{CG} is the volume of the *CG* particle. The current implementation uses only irreversible reactions and the forward rate-of-progress for reaction *j* is written as

$$r_j^f = k_j^f \prod_i^{NS} \left[\frac{y_i}{V_{CG}}\right]^{\upsilon' ij} \tag{2}$$

$$k_j^f = A_j \exp\left(-\frac{E_j}{k_B\theta}\right), \quad (3)$$

where υ'_{ij} is the reactant stoichiometric coefficient; k_j^f is the forward Arrhenius rate with activation energy E_j and pre-exponential A_j; k_B is the Boltzmann constant; and θ is the internal temperature of the particle.

The right-hand-side (RHS) of Eq. 1 is a nonlinear function of the chemical composition (with *NS* species) and internal temperature. Nonlinear ODEs are often categorized as "stiff" and are usually solved using implicit or semi-implicit methods. Implicit methods are generally more efficient (e.g., they require fewer integrator steps) than explicit solvers for stiff systems since the explicit step size (h) is limited by stability, not by the desired accuracy.[4] However, implicit solvers are more expensive per step, so moderately stiff ODE systems may be solved faster with explicit schemes under certain conditions.

The ODE systems of each of the CG particles are independent and can be solved concurrently using various data and thread parallel processing techniques. Recent studies[5–11] related to atmospheric chemistry and combustion modeling have investigated solving concurrent ODE systems on GPUs or similar many-core devices. GPUs are throughput-oriented devices designed to operate on tens of thousands of threads concurrently. In the majority of the cited studies, a single GPU thread is used to solve each individual ODE system. That is, thousands of independent ODE systems are solved concurrently.

In this study, we employ 2 ODE solvers—CVODE* and RKF45—which we previously developed for NVIDIA Compute Unified Device Architecture (CUDA) GPUs.[9] The CPU versions of both algorithms implemented by us are also used for comparison. CVODE is a widely used multistep, variable-order (up to fifth-order), implicit ODE solver based on a backwards differentiation formula (BDF).[12] RKF45 is an explicit, single-step, fourth-order adaptive Runge-Kutta-Fehlberg (RKF) method with a fifth-order error estimator.[13]

CUDA GPUs rely upon the single-instruction, multiple-thread (SIMT) parallel processing paradigm. In SIMT, optimal efficiency is achieved when each thread within the same "warp" (i.e., a team of 32 threads) executes the same instruction. In the context of parallel ODE solvers, this is achieved if each thread follows the same control flow through the solver logic. When ODE systems require different numbers of integrator steps or nonlinear iterations (e.g., different initial conditions), the warp's sibling threads will diverge (i.e., follow different code paths) and the throughput of the GPU will degrade. In previous work,[9] we demonstrated that single-step, explicit ODE solvers such as RKF45 have lower divergence on mildly

*CVODE stands for C Variable Order Differential Equation.

stiff ODEs and can be faster than implicit methods such as CVODE when executed on a GPU. Niemeyer and Sung[11] showed similar behavior with a Runge-Kutta-Chebyshev solver for moderately stiff combustion kinetics.

In the next section we present details of the baseline LAMMPS DPD-RX software, the benchmark design, and the methods for measuring the computational performance. This is followed by an analysis of the baseline performance and identification of performance-limiting aspects of the application. In the results section, we present several techniques aimed at improving the performance of the DPD-RX software. These include using different ODE solver algorithms, offloading the reaction kinetics to GPUs and investigating different matrix representations for the stoichiometric matrices in Eqs. 1 and 2. Finally, we summarize the optimization methods and analyze how these performance improvements will impact the scientific productivity of the LAMMPS DPD-RX software.

2. Methodology

2.1 System Hardware used for Benchmarks

All of the benchmarks presented here were run on the Thunder system located at the US Air Force Research Laboratory's Department of Defense Supercomputing Resource Center. Thunder is an SGI ICE X* system with 356 accelerator nodes and 3,216 CPU-only nodes. Half of the accelerator nodes (178) have 2 NVIDIA Kepler K40m GPUs and the remaining 178 accelerator nodes have 2 Intel Xeon Phi 7120P co-processors. All accelerator nodes have 2 Intel E5-2697v3 (Haswell) CPUs with 14 cores per CPU. All compute nodes use a dedicated (NUMAlink v5) communications network for message passing interface (MPI) messages. For uniformity, all of the benchmarks reported here were run on accelerator nodes even when not offloading the computations to the GPUs. Also, for all MPI-only benchmarks, except serial cases, a multiple of 14 MPI processes were used so that all 14 cores within each CPU were fully utilized.

2.2 DPD-RX Benchmark Scenario

A prototypical DPD-RX benchmark scenario was created that can be scaled from a single-core simulation to a very-large simulation requiring thousands of cores and hundreds of GPU accelerators. An ideal DPD fluid is modeled, where the conservative forces between particles are neglected and the dynamics are governed

*SGI is a corporate name standing for Silicon Graphics Inc. ICE is a product line; X is a series.

purely by the dissipative and random forces. The simulation sizes studied are 16,000; 128,000; 1,024,000; and 8,192,000 particles, respectively (i.e., increasing by a factor of 8). All benchmarks initialize half of the particles with high temperature (3,500 K) and the other half as low temperature (1,500 K). All particles are initially pure reactant (RDX, detailed as follows). The simulation is run for 100 DPD time steps at 1.0 fs per time step. Complete simulations typically require a few nanoseconds to a microsecond of simulated time, thus requiring several orders of magnitude more time steps; however, the salient performance features of the DPD-RX algorithm can be assessed with this shorter benchmark.

2.3 Two Reaction Mechanisms

To make practical use of DPD-RX simulations, a proper balance of overall run time versus fidelity of the reaction kinetics must be found. We use 2 reaction mechanisms to study this tradeoff: a small, reduced-order reaction mechanism with 9 chemical species and 4 reactions (9S4R) and a larger mechanism with 45 species and 201 reactions (45S201R).

The 9S4R mechanism models the thermal decomposition of crystalline cyclotrimethylenetrinitramine, a model explosive commonly referred to as RDX, with 4 global, irreversible reactions.[3] The 45S201R mechanism is based on a detailed RDX decomposition reaction mechanism[14] containing 228 reactions with 45 intermediate and product species. A simplified version of this mechanism was created that excludes third-body fall-off reactions, which are not currently supported within the DPD-RX software. The focus of this study is to gauge the performance impact of large and complex mechanisms of comparable size to the complete RDX[14] mechanism on the DPD-RX software. Therefore, the 45S201R mechanism, even incomplete, is representative of higher fidelity reaction mechanisms and is suitable for gauging the performance levels that would be needed for the practical use of such detailed chemical kinetics in DPD-RX simulations.

2.4 Baseline LAMMPS DPD-RX Software

The baseline DPD-RX method was implemented within the LAMMPS software package (the “15 May 2015” version), which uses a distributed memory parallel model with MPI. Particles are partitioned using a spatial partitioning scheme to balance the number of particles per MPI process. The CVODE implicit ODE solver[12] was used to integrate Eq. 1. The stiff BDF method was used with equal (scalar) absolute and relative tolerances (10^{-8}). The Jacobian of Eq. 1 was derived and used by CVODE to solve the nonlinear BDF equations with Newton iterations.

Finally, the initial step size (h_o) was automatically selected using the default algorithm within CVODE. A dense matrix representation was used for the stoichiometric matrices in Eqs. 1 and 2.

3. Analysis

3.1 Performance and Strong Scaling of Baseline DPD-RX

The performance on Thunder of the baseline DPD-RX software is shown in Fig. 1. It shows the total wall-clock time for executing 100-DPD time steps with the 9S4R and 45S201R reaction mechanisms in systems with 16,000–8,192,000 particles on 1–3,584 CPU cores. Dotted lines denote runs with the 9S4R mechanism; solid lines are for the 45S201R mechanism. As can be seen in Fig. 1, the execution time for the 45S201R mechanism is approximately 100 times longer than for the 9S4R mechanism regardless of the number of particles in the system. As the benchmarks are run on larger numbers of MPI processes for a given number of particles, the simpler 9S4R mechanism does not strong-scale as well as the 45S201R mechanism. This effect of Amdahl's law can be seen in Fig. 1 as the dotted lines diverge upward from the lower-right-end of each of the solid lines.

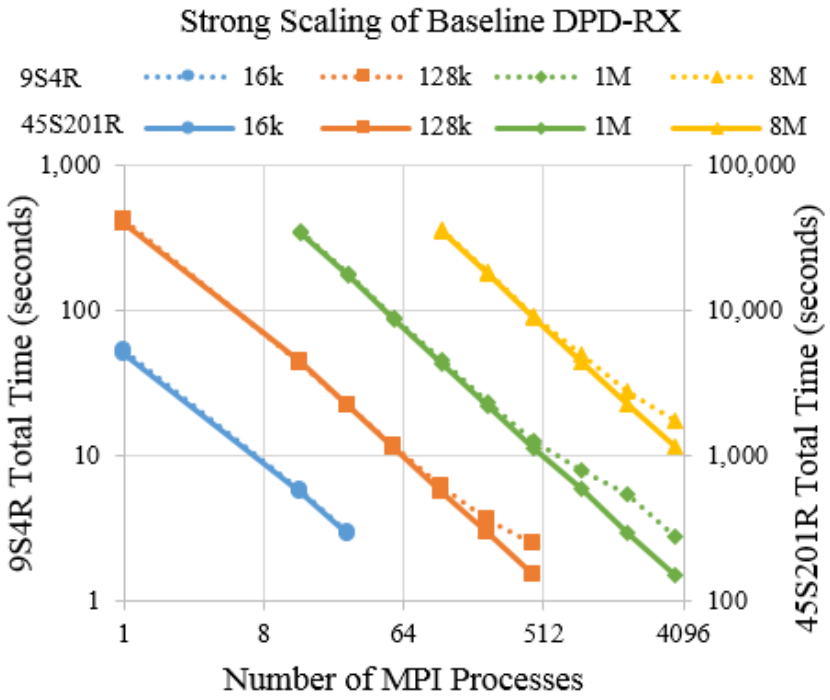

Fig. 1 Baseline DPD-RX total simulation time on Thunder for 100-DPD time steps using MPI and the CVODE implicit solver with an analytical Jacobian with a dense stoichiometric matrix for the 9S4R and 45S201R reaction mechanisms for systems of 16-k, 128-k, 1-M, and 8-M particles

3.2 Cost and Weak Scaling of Baseline DPD-RX

To efficiently use compute resources for typical production runs of the baseline DPD-RX software with the 9S4R mechanism, the workload intensity (WI) is generally kept between 2,000 and 10,000 particles per CPU core. As shown by the rising dotted lines going toward the left in Fig. 2, the computational cost-per-particle is significantly higher when there are too few particles per core for the 9S4R mechanism. Above 10,000 particles per CPU core, the efficiency of the baseline DPD-RX with the 9S4R mechanism levels out. Because of the higher absolute cost of the 45S201R mechanism, WIs under 300 particles-per-CPU-core are still efficient. The 4 serial runs, circled in Fig. 2, are outliers in efficiency simply because they forgo any MPI communication costs, as well as any parallel load imbalance costs.

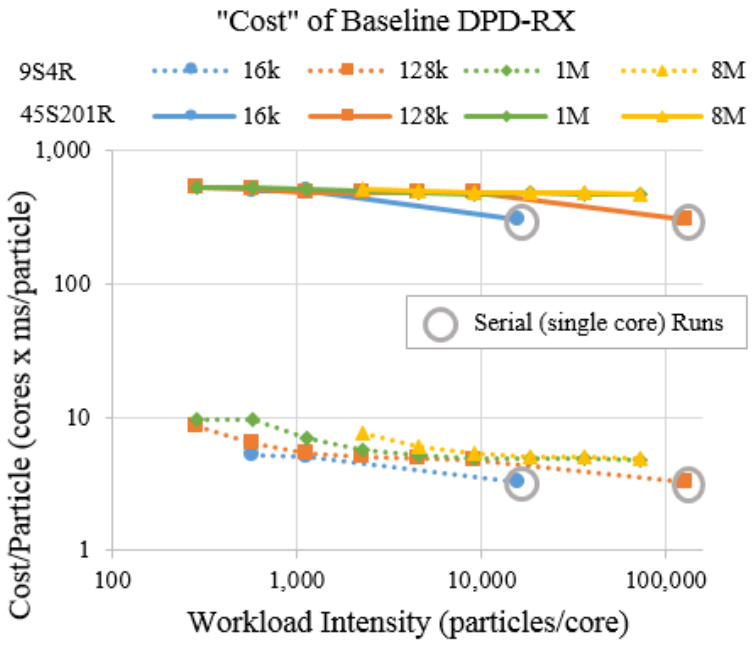

Fig. 2 "Cost" of baseline DPD-RX on Thunder where cost is computed as CPU cores multiplied by total time in milliseconds for 100-DPD time steps. Runs used MPI and the CVODE implicit solver with an analytical Jacobian with a dense stoichiometric matrix for the 9S4R and 45S201R reaction mechanisms for systems of 16-k, 128-k, 1-M, and 8-M particles.

3.3 Percentages and Load Imbalance of Baseline DPD-RX

Solving the reaction kinetics mechanism for ideal DPD fluids was measured as the costliest individual component of the DPD-RX software. For the 9S4R reaction mechanism, the reaction component is approximately 70% of the run time as shown in Fig. 3, while it is more than 99% with the 45S201R mechanism as shown in Fig. 4.

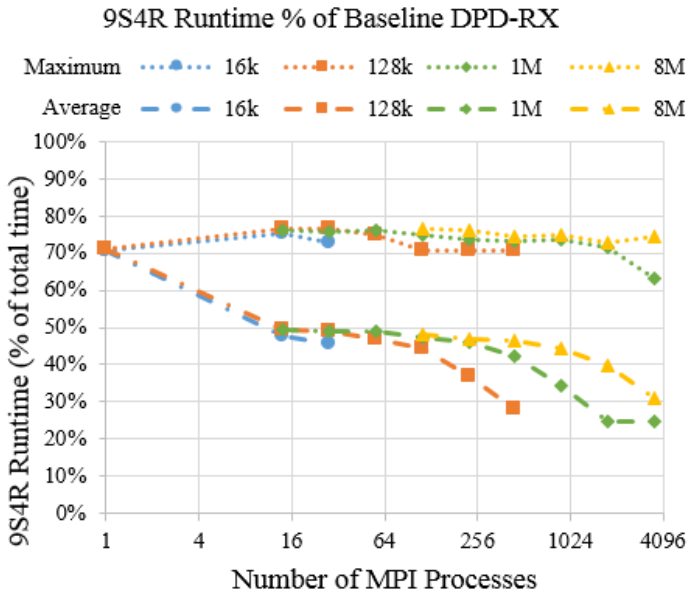

Fig. 3 Maximum and average run-time percentage of 9S4R mechanism across all MPI processes in the baseline DPD-RX relative to the total run time

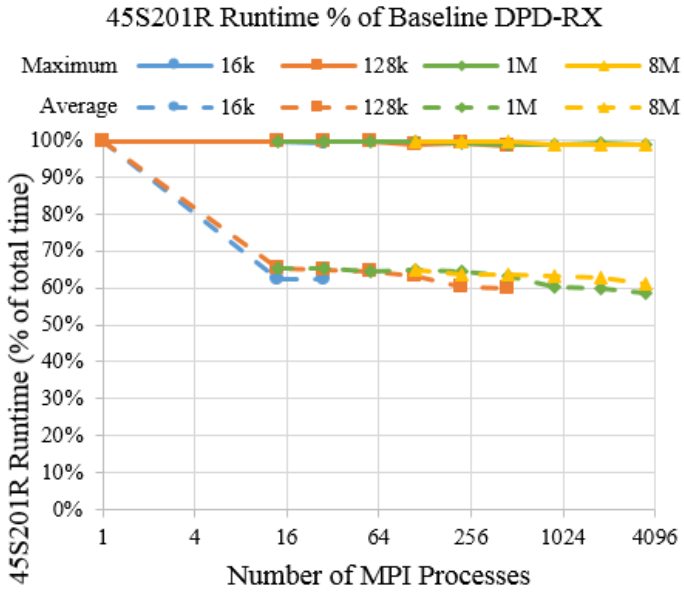

Fig. 4 Maximum and average run-time percentage of 45S201R mechanism across all MPI processes in the baseline DPD-RX relative to the total run time

Both the average and maximum reaction run times over all of the MPI processes are shown in Figs. 3 and 4 to highlight the load imbalance from the ODE solvers. Each ODE system is unique and independent of all other particles. Certain particles will require more ODE integrator steps, which will generally take longer to solve. This can lead to a nonuniform workload since the LAMMPS partitioning algorithm assumes a uniform cost-per-particle.

In the current benchmark simulation, the hot half of the domain will react more quickly and generally requires more time to solve per particle. This means that roughly half of the MPI processes will have higher reaction run times, while the remaining processes must wait to exchange neighboring particle information. The load imbalances do not appear to grow significantly since the parallel efficiencies are relatively flat for practical workloads if we ignore the unique serial cases.

Because of the extremely high cost of solving the reaction ODE systems and since the reaction component does not involve communication, the remainder of this discussion focuses upon single-device optimization (i.e., a single multicore CPU or many-core GPU accelerator). Any single-device performance improvement will have a direct impact on large-scale parallel simulations with many compute nodes. The impact of nonuniform workloads will only be examined within the context of a single compute device. Distributed, multidevice load-balancing will be addressed in future studies.

4. Results

In this section, we examine several strategies for improving the performance of the reaction kinetics component of the DPD-RX software within a single compute device. The run times using these optimization strategies are compared to the baseline DPD-RX run times presented in the previous section.

The 128-k particle case running on a single multicore CPU or a single GPU was selected as the primary benchmark for this optimization study. Thus, for both the MPI-only and the OpenMP implementations, the 14 cores of a single CPU are used for computing the reaction kinetics. Similarly, for the CUDA implementation, a single-Kepler GPU is used for computing the reaction kinetics. Running the 128-k particle case on only a single compute device permits the evaluation of various optimization methods for the kinetics integration as well as to assess the impact of load-balancing methods within a single device using a multithreaded approach. This study can then be used as a guide for future dynamic load-balancing methods across distributed MPI processes. The workload intensity of the 128-k particle case is 9.1-k particles per CPU core, which is also representative of standard practice for DPD-RX simulations.

Although LAMMPS itself has support for OpenMP and CUDA parallelism for particle–particle interactions, for this study those non-RX computations were limited to MPI-only parallelism to simplify the experiment. Thus, unless stated otherwise, the timing measurements presented in this section include only the

reaction kinetics wall-clock time, and exclude the rest of the LAMMPS execution time.

As noted, the baseline DPD-RX algorithm used the implicit CVODE stiff ODE solver with an analytical Jacobian matrix function of Eq. 1. The CVODE has the capability of estimating the system Jacobian using finite-difference quotients. This strategy requires *NS* RHS function evaluations (i.e., the number of chemical constituents in the current context). In general, an analytical Jacobian is more efficient since the *NS* RHS evaluations can be expensive and repetitive. However, the finite-difference Jacobian is considered since the RHS function is less complex and may offer more optimization opportunities (e.g., concurrently evaluating the *NS* RHS functions) than the analytical Jacobian. In the rest of this report, the CVODE with analytical Jacobian (CVAN) and CVODE with difference quotient (CVDQ) notations indicate a CVODE solver with an analytical and finite-difference Jacobian, respectively.

We have also used the RKF45 ODE solver introduced previously. The local error tolerances and h_o estimation algorithm are the same as were used for CVODE previously.[9] This ensures that the integrated solutions produced by the different ODE solver methods are equivalent (to within the specified tolerance) and that their run times can be consistently and directly compared.

Three parallel processing methods were implemented for these 3 ODE solver strategies: the baseline MPI-only approach, a multithreaded approach using OpenMP with dynamic scheduling, and an accelerator approach whereby the ODE solvers are offloaded to a CUDA GPU. The OpenMP method was designed to handle the nonuniform workload by using dynamic scheduling. This approach is suitable for nonuniform workloads but is only applicable to shared-memory platforms (e.g., a multicore CPU). The GPU offloading approach uses CUDA versions of the CVODE and RKF45 solvers we previously developed and validated.[9] All GPU benchmarks reported in the following use 128 threads per CUDA thread-block and the run times include all data transfer costs.

4.1 Impact of Solvers and Parallelism

The run time of the reaction component of the DPD-RX algorithm using the 3 ODE solvers and the 3 parallel implementations are shown in Fig. 5 for the 9S4R and 45S201R reaction mechanisms.

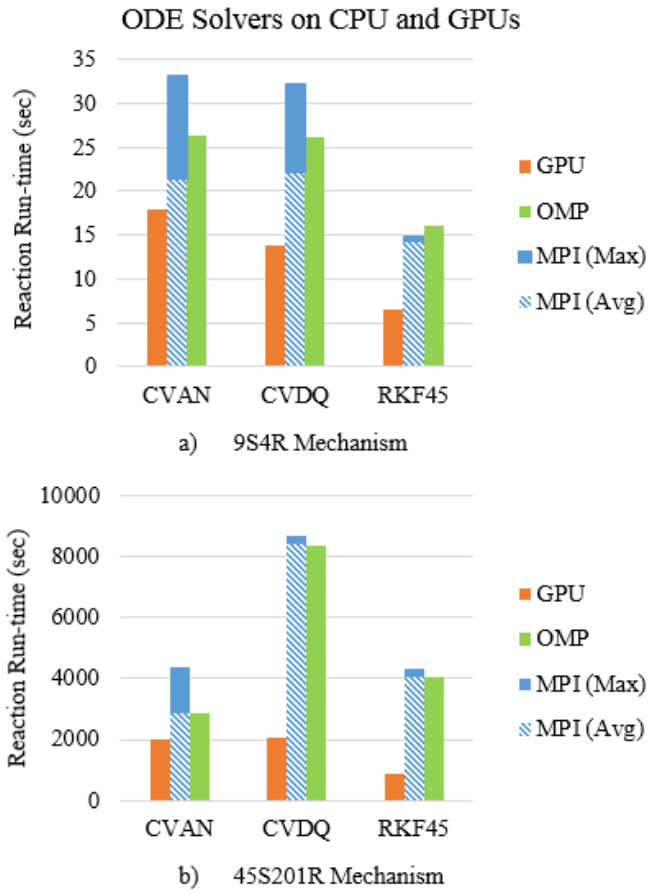

Fig. 5 Comparison of the average reaction run times using various ODE solver methods and compute devices for the 128-k particle case with the a) 9S4R and b) 45S201R reaction mechanisms. Three ODE solver algorithms are compared: CVODE with an analytical Jacobian (CVAN), CVODE with a finite-difference Jacobian (CVDQ), and the explicit RKF45 solver. Three implementations of each of those algorithms are compared: the original MPI-only implementation (MPI) running on 14 cores (i.e., one CPU), an OpenMP implementation (OMP) using dynamic load-balancing on 14 cores, and a CUDA implementation (GPU) offloaded onto a Kepler K40m GPU. For MPI, the hashed bars show the average MPI process run times and the solid bars show the maximum MPI process run time over all 14 MPI processes.

Both the average and maximum MPI process run times across all 14 processes are show in Fig. 5. This demonstrates the wide variation possible due to the nonuniform workload using the MPI-only paradigm. The maximum run time is what all MPI processes ultimately experience due to the bulk-synchronous nature of LAMMPS. The average run time is representative of the performance that could be achieved using a load-balancing technique that spans MPI processes.

The OpenMP case uses dynamic loop scheduling to balance the nonuniform workload across the OpenMP threads (and CPU cores). A “chunk-size” of 5

particles per thread was found to be the most efficient, on average, for OpenMP dynamic scheduling of the DPD-RX component. For the 9S4R reaction mechanism, the OpenMP run time is higher than the average MPI run time for the CVAN and CVDQ solvers but is 21% less than the maximum MPI run time. This demonstrates the benefit of adaptive load-balancing for the nonuniform workload.

For the larger 45S201R reaction mechanism, the average MPI and OpenMP reaction times are statistically equal but the OpenMP is 35% faster than the slowest MPI processes. Since the reaction component dominates the total cost for the 45S201R reaction mechanism (>99%), OpenMP improves the total run time by 32% (not shown). This again demonstrates the benefits of adaptive load-balancing for the nonuniform workload as well as the dominant cost of the DPD-RX component for larger, detailed mechanisms.

Compared to CVAN with MPI, the average CVDQ method with MPI is 4% slower with the 9S4R reaction mechanism but is 193% slower with the 45S201R mechanism. This is due to the higher cost of the finite-difference Jacobian when scaled to larger mechanisms. There are several operations within the RHS function that are unnecessarily repeated with the *NS* RHS function evaluations in the CVDQ method. The analytical Jacobian is more expensive than a single-RHS function evaluation; however, the absolute cost is significantly less than the 45 RHS functions needed with the 45S201R mechanism using the finite-different approximation in CVDQ.

The CVDQ method, while slower, does exhibit less variability than the CVAN method. The difference in variability (i.e., the % difference between the maximum and the mean) of the CVAN and CVDQ methods is small for the 9S4R mechanism, 54% and 45%, respectively. The variability for CVAN is similarly high (52%) for the 45S201R mechanism, yet is only 3% for CVDQ. The absolute variability is also significantly less with the CVDQ method on the larger mechanism. We observed that the number of Jacobian evaluations required by the CVDQ solver was constant (1 Jacobian per integrated CG particle) for all time steps and for both 9S4R and 45S201R mechanisms. However, the number of Jacobians used by CVAN varied and required over 2 Jacobians per particle at times. This variation could be caused by numerical differences (e.g., condition number) in the 2 Jacobian matrix representations, which could cause subtle differences in the ODE integration trajectories.

The RKF45 method with MPI also exhibits much lower variability across the MPI processes and is 33% faster than the baseline CVAN implicit solver for the 9S4R mechanism. For the 45S201R mechanism, however, the average RKF45 run time is 41% slower than the average CVAN run time but the maximum run times are

statistically equal. The ODE systems in the DPD-RX simulations are generally considered stiff; however, the nonstiff solver appears quite capable of handling the specified tolerances efficiently. The small DPD time step may suppress the stiffness by restricting longer time-scale differences, a primary cause of ODE stiffness. That is, time scales are limited to 1 fs or less.

The different ODE solvers all require approximately the same number of integrator steps. The RKF45 solver always requires 6 RHS function evaluations per integration step while the CVODE solver may need only one per integration step. On average, the RKF45 solver required 3 times as many total RHS function evaluations than the CVAN solver. The higher cost of the RHS function with the 45S201R mechanism leads to higher overall cost for the RKF45 solver compared to the CVAN solver. This higher cost is most clearly seen in Fig. 5b when looking at the dynamically load-balanced OMP (green) case.

The run times using the GPU version of the CVODE and RKF45 solvers are also shown in Fig. 5. The CVAN solver is accelerated by 32% over the OpenMP CVAN run time with the 9S4R mechanism. The OpenMP run time is used as a reference here since it is dynamically load-balanced and is more likely a truer estimate of the total single-device run time. The CVDQ is accelerated by 47% while RKF45 is accelerated by 60% over the OpenMP run time. The acceleration with the 45S201R mechanism is more substantial with the CVDQ and RKF45 solvers, achieving 75% and 78% improvement, respectively. The CVAN solver, however, is only improved by 30%. The CPU-GPU data transfer costs for the 9S4R mechanism account for 13% of the run time with the RKF45 solver. The transfer costs become negligible with the large 45S201R mechanism due to the higher integration cost.

The lower acceleration with the CVAN solver on the GPU can be attributed to the higher level of variability observed. As noted, divergence within a GPU warp decreases the efficiency. The higher level of variability observed with the CVAN solver indicates a higher level of divergence when compared to the other 2 solver methods. This explains the high-GPU acceleration for the RKF45 solver with both the 9S4R and 45S201R mechanisms and for the CVDQ solver with the 45S201R mechanism. For the RKF45 solver in particular, there is little opportunity for divergence owing to the simplicity of the RKF45 scheme. The only source of divergence is in the total number of ODE integration steps-per-particle and there was little observed variation in this metric.

4.2 Impact of Sparse-Matrix Formulation

The baseline implementation of the stoichiometric matrix (ν) in Eqs. 1 and 2 used a dense matrix format with *NS* rows and *NR* columns. The net species production

rate in Eq. 1 and the reaction rates in Eq. 2 were evaluated assuming a dense stoichiometric matrix. The net species reaction rates in Eq. 1 can be written as a matrix-vector product of stoichiometric matrix and the rate-of-progress vector. This is suitable for the 9S4R reaction mechanism since there are few zero elements in ν. There, the relative number of zero elements in the matrix (i.e., the sparsity) is 44%. The cost of including the zero elements in the matrix-vector products was assumed negligible due to the small matrix size. This design, however, is not suitable for reaction mechanisms with larger numbers of chemical species and reactions such as in the 45S201R mechanism. The sparsity for the 45S201R mechanism is 91% and zero elements will constitute the vast majority of the matrix-vector products in Eqs. 1 and 2. A sparse matrix formulation was implemented following the *ELLpack* format,[15] which has been shown to be efficient on throughput-oriented devices such as GPUs. This format is also suitable since the number of nonzero elements per row (i.e., the number of species per reaction) is small and does not vary significantly.

An additional optimization was investigated regarding the stoichiometric matrix. The majority of stoichiometric terms in the 45S201R reaction mechanisms are small, integral values (i.e., positive integers less than 3). The standard C/C++ exponentiation function (i.e., *pow(x,y)* evaluates x^y) uses floating-point values for both the base and exponent. This generality adds significant cost for small, integer exponents. We have added a test to determine if all of the stoichiometric terms in a given reaction are integral. If so, we use a specialized function (*powi*) designed for small, positive integer exponents. Otherwise, the standard *pow* function is used.

Sparse versions of the RHS and analytical Jacobian functions were implemented for both the host CPU and GPU. The run times using the original dense baseline matrix format (*Dense*), sparse format (*Sparse*), and sparse format plus the integral exponentiation function (*Sp-PowI*) with the CVAN solver on the 128-k particle case with the 9S4R and 45S201R reaction mechanisms are shown in Fig. 6. In Fig. 6, the average MPI process run times are shown by the hashed bar and the maximum MPI process run times by the solid bar. GPU is the run time (including data transfer) using the Kepler K40m GPU; and OMP is the OpenMP thread parallel model with dynamic load-balancing using all 14 cores on 1 Thunder CPU. Specific data is shown in the figure for small values. With the 9S4R mechanism, there is a small but noticeable (22%) improvement in the average MPI process run time for the reaction component using the sparse format. The variation between the MPI processes is proportionally reduced. The OpenMP implementation is similarly improved by 25%. The MPI run times are unaffected by the *powi* function and the OpenMP throughput is worsened. This result is expected since none of the 9S4R reactions qualify for the *powi* function; therefore, this method only adds conditional testing overhead.

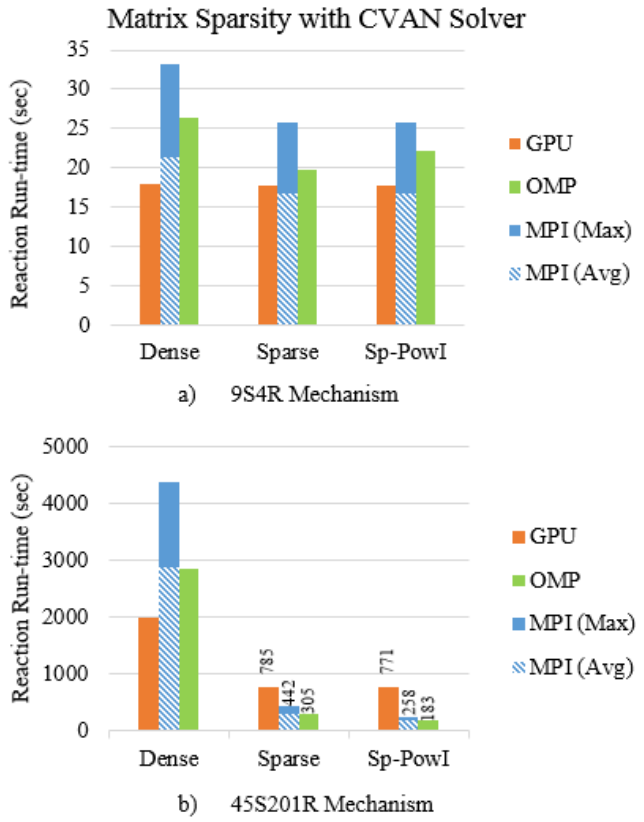

Fig. 6 Comparison of 128-k particle reaction solution times of the CVODE solver with an analytical Jacobian (CVAN) using different matrix storage formats while implementing the a) 9S4R and b) 45S201R reaction mechanisms.

Unlike the 9S4R reaction mechanism, the 45S201R reaction mechanism with the sparse version of the CVAN solver is significantly improved (Fig. 6b). The average MPI run time is reduced by 90% using the sparse format. (Again, the variations are proportionally similar). The *powi* function improves the average MPI process run time by an additional 39%. The OpenMP reaction run time is statistically the same as the average MPI process run times and the net performance improvement is identical.

The GPU run times are also improved but to a lesser extent than the CPU run times. The sparse format improves the GPU throughput by 61% but the *powi* function only adds an additional 2% to the performance. The lower improvement for the GPU run times is typical of bandwidth-limited conditions. That is, the *powi* function reduces the floating-point intensity but the limiting factor is the memory throughput.

The results for the same sparsity study using the explicit RKF45 solver are shown in Fig. 7 using the same notations as in Fig. 6. For the 9S4R mechanism, the sparse format improves the average MPI process and OpenMP run times by 43% and 39%,

respectively, and the GPU run time by 15%. These improvements are greater than observed with the CVAN solver. The sparse implementation of the analytical Jacobian function requires a matrix–matrix product, which is less efficient in terms of data parallelism (i.e., vectorization) than the RHS function implementation. Furthermore, the Jacobian matrix itself is dense and must be factorized as part of the nonlinear Newton iteration within CVODE. These dense matrix operations are unaffected by the sparse-stoichiometric-matrix optimizations. The RKF45 solver, which uses only sparse-matrix-vector products in the RHS function, benefits proportionally more since the dominant cost of the explicit method is the RHS function itself.

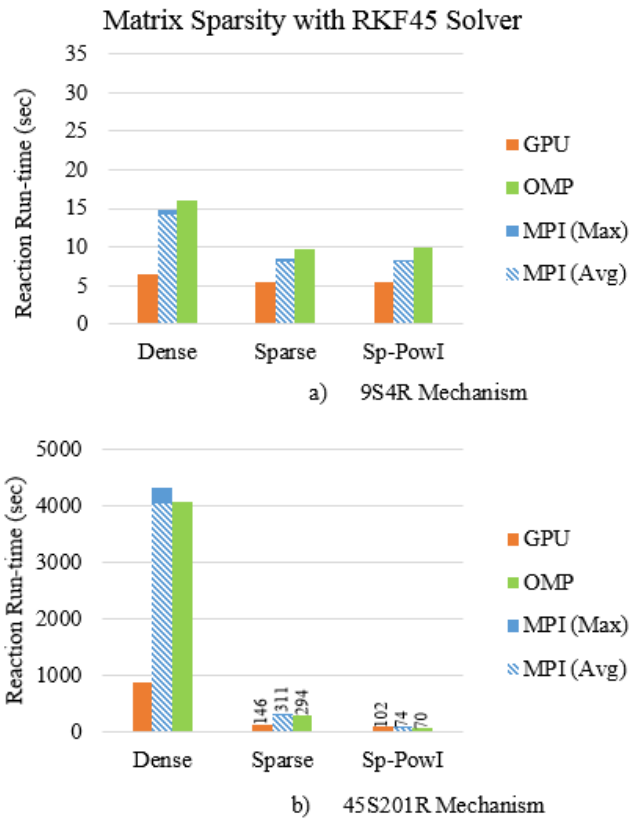

Fig. 7 Comparison of 128-k particle reaction solution times of the RKF45 ODE solver using different matrix storage formats while implementing the a) 9S4R and b) 45S201R reaction mechanisms

The sparse RKF45 solver has significantly higher performance than the dense method on the 45S201R reaction mechanism. The average MPI process and OpenMP run times are both reduced by 93% and the GPU reaction run time is reduced by 83.5%. The performance improvements match close to the sparsity pattern on the latency-oriented platform (i.e., the CPU). On these devices,

performance is improved most effectively by reducing the number of operations (e.g., floating-point multiplication). This is especially relevant here since the individual ODE systems are small enough to fit within the L1 data cache on the CPU cores (i.e., memory bandwidth should not be a limiting factor). This effect is seen again with regards to the *powi* optimization: the integer exponent function improves the CPU run times by an additional 76%. With the *powi* optimization the GPU run time is also improved (by 30%) but this is less significant than on the CPU cores. Again, this indicates that the GPU is limited by bandwidth, not floating-point throughput.

4.3 Impact across All Optimizations

The impact of the various optimizations detailed in the previous sections can vary for different simulation parameters. To fairly compare them, we will use a normalized measure of performance of the reaction kinetics measured via a parameter sweep consisting of over 1,000 unique benchmark runs. The normalized performance measure is the throughput of particles integrated per device millisecond, where each device is either a single CPU or GPU.

Figure 8 shows the throughput of the 9S4R mechanism on 18 different combinations of solvers (CVAN, CVDQ, and RKF45), stoichiometric matrix (dense and sparse), and parallelism paradigms (MPI, OpenMP, and GPU) run with 23 different combinations of particle and device counts. The horizontal axis is organized from left-to-right with generally increasing WI, starting at 4,000 particles-per-device, and ending at 1,024,00 particles-per-device. For the CPU devices, that range corresponds to 285–73,142 particles-per-core; thus, we are exploring both below and above the typical WI of 2,000–10,000 particles-per-core as previously discussed.

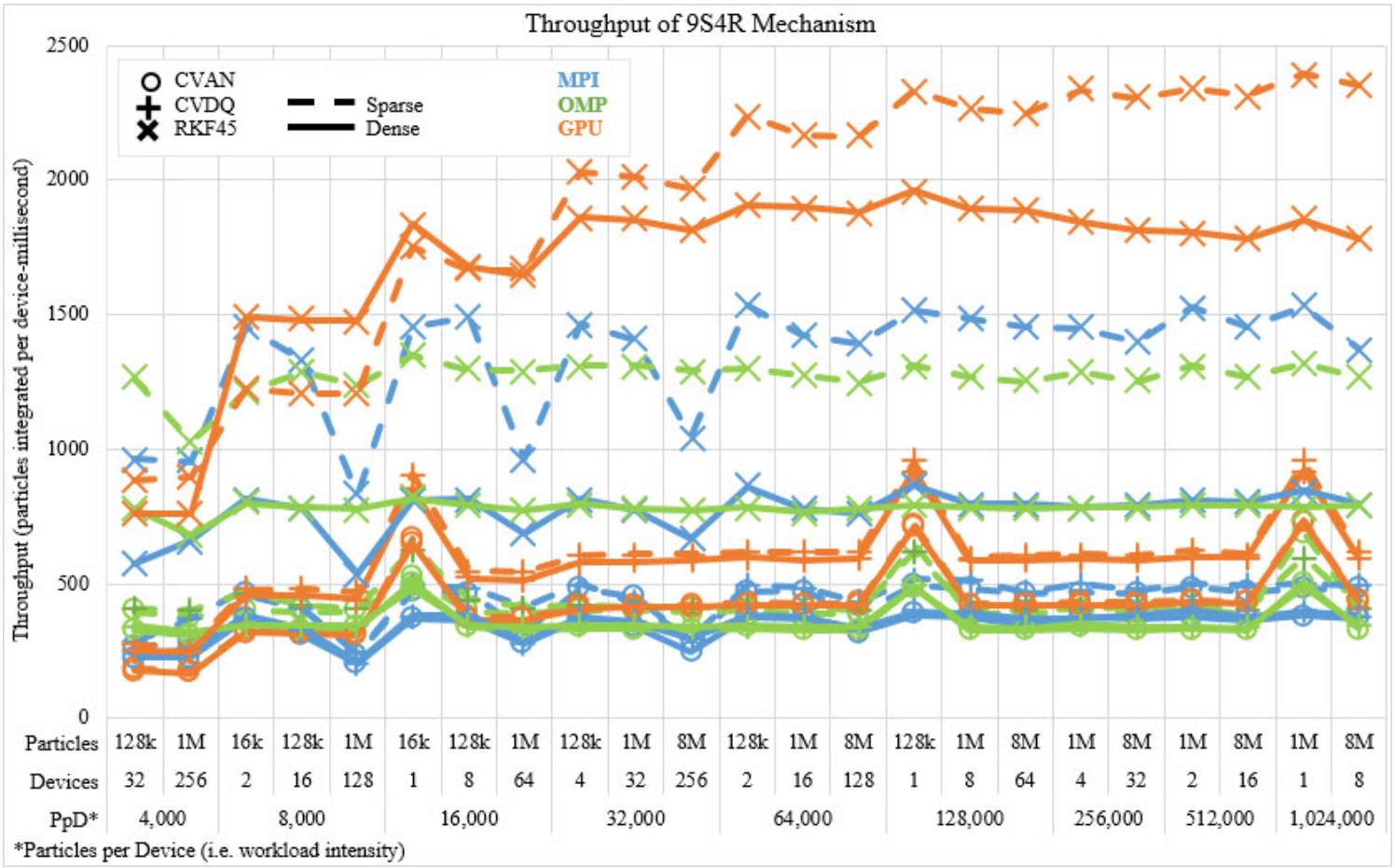

Fig. 8 Throughput (particles integrated per device millisecond) of the reaction kinetics integration run time as a function of workload intensity with the 9S4R mechanism with the 16-k, 128-k, 1-M, and 8-M particle simulations using MPI, OMP, or GPU; CVAN, CVDQ, or RKF45; and dense or sparse stoichiometric matrix. Parallelism paradigms are delineated by colors, solvers by symbols, and matrix format by line structure.

As can be seen by the dashed lines in Fig. 8, the sparse matrix representation gives a significant performance boost across all of the 9S4R scenarios. Similarly, the GPU runs (orange) show improvement for all 9S4R scenarios with at least 8,000 particles-per-device, though the GPU performance does not saturate until 32,000–64,000 particles-per-device for the 9S4R mechanism. The increased load imbalance observed with the CVAN and CVDQ solvers can be clearly seen by the dramatic performance spikes for single-device runs for both the GPU (orange) and OpenMP (green) paradigms. For the OpenMP runs, this spike can be attributed to the dynamic load-balancing that is easy to realize within a single OpenMP context. For the single GPU runs, this performance spike can also be attributed to a dynamic load-balancing effect for the 9S4R mechanism.

Figure 9 shows the throughput of the 45S201R mechanism on a similar combination of solvers, matrix formats, and parallelism paradigms run with the same 23 combinations of particle and device counts. The dynamically load-balanced performance spikes for single device runs can also be seen in Fig. 9 for the 45S201R mechanism, but they are not as dramatic an improvement for the GPU as they are for the OpenMP runs. This can be explained by considering the granularity of the scheduling schemes on the GPU and OpenMP. On the GPU, thread blocks are dynamically scheduled to available streaming multiprocessors. We are using 128 threads per-thread-block for all GPU benchmarks, which equates to a scheduling granularity of 128 ODE systems. The OpenMP dynamic scheduler uses only 5 ODE systems-per-thread. The finer granularity appears better suited to handle the variation between the ODE system-integration costs in the 45S201R mechanism.

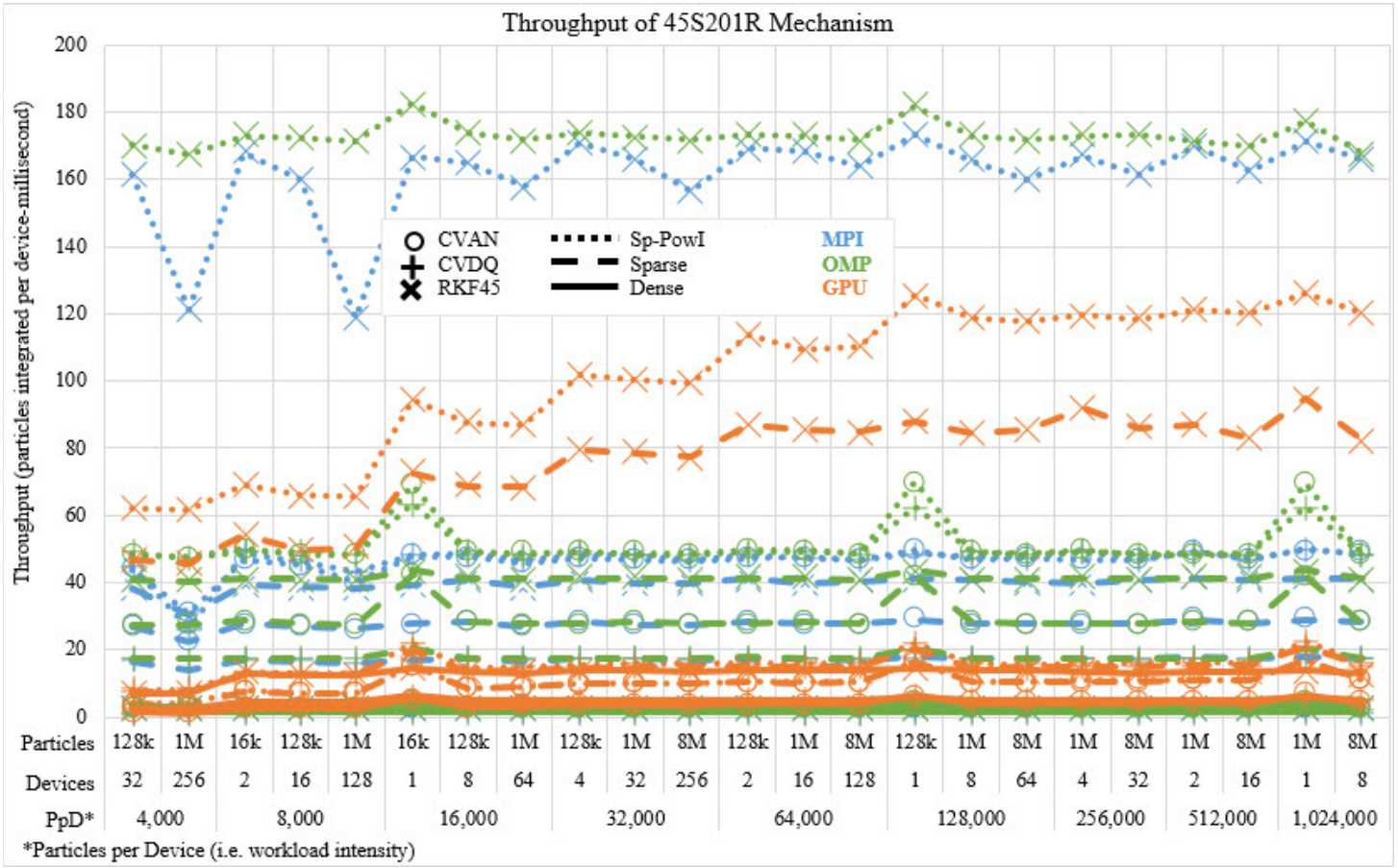

Fig. 9 Throughput (particles integrated per device millisecond) of the reaction kinetics integration run time as a function of workload intensity with the 45S201R mechanism with the 16-k, 128-k, 1-M, and 8-M particle simulations using MPI, OMP, or GPU; CVAN, CVDQ, or RKF45; and dense, sparse, or Sp-PowI stoichiometric matrix. Parallelism paradigms are delineated by colors, solvers by symbols, and matrix format by line structure.

As can be seen by the dashed lines in Fig. 9, the sparse stoichiometric matrix also gives a significant performance boost across all of the 45S201R scenarios. However, the Sp-PowI (dotted lines) and RKF45 (X symbols) yield the best performance regardless of the parallelism paradigm for the 45S201R mechanism. The most striking observation from Fig. 9 is that the RKF45 solver with Sp-PowI and OpenMP is the fastest approach in all of the 45S201R scenarios we tested.

5. Conclusions

We have presented results from an optimization effort to improve the throughput of LAMMPS DPD-RX simulations of energetic materials. The reaction kinetics component was shown to be the limiting factor in a scaling study of the DPD-RX software. For the 9S4R reaction mechanism, the reaction component accounted for 70% or more for typical DPD-RX simulation WIs (i.e., 2,000–10,000 particles-per-CPU-core). For the 45S201R reaction mechanism, the reactions accounted for more than 99% of the run time in our baseline benchmark. Because of this high cost, we investigated several strategies to improve the performance of this specific component alone. The performance improvement strategies focused upon single-device optimizations as the costly reaction components do not involve communication and any single-device optimizations are applicable across any number of devices.

Load imbalances were identified as a performance limitation in MPI-only parallel environments. The load imbalances were caused by the variable workloads of the reaction systems. The benchmarked version of LAMMPS distributes particles evenly across the available MPI processes assuming uniform cost-per-particle. We tested OpenMP as a way to dynamically load-balance the ODE integrations within a single multicore CPU. The average run time of the reaction component with only MPI parallelism was faster than or equal to the OpenMP run time; however, the OpenMP was faster than the slowest MPI process, which is the limiting factor in the bulk synchronous LAMMPS framework. This dynamic load-balancing technique was limited to a single CPU device but demonstrates the improved parallel efficiency that could be attained with an effective distributed memory load-balancing strategy. Strategies for improving distributed load-balance shall be investigated in future studies. Methods such as those demonstrated by Nakashima et al.[16] should be applicable for the particle-based LAMMPS framework. Furthermore, weighted dynamic load-balancing capabilities[17] were recently added to LAMMPS and will be investigated.

We investigated different methods for integrating the system of stiff ordinary differential equations that arise from the reaction mechanism. The baseline

integration method used the implicit CVODE BDF solver with an analytical Jacobian matrix function (CVAN). We also evaluated the CVODE solver using a Jacobian matrix automatically computed using finite-differences (CVDQ). The CVDQ method was found to be equally performant for the 9S4R reaction mechanism but was significantly slower for the 45S201R mechanism due to the high cost of estimating the Jacobian matrix. We also investigated the explicit RKF45 ODE solver.

The performance of the different ODE solvers was dependent upon the reaction mechanism size and the computational device. In addition to the OpenMP method with dynamic scheduling, GPU (CUDA) implementations of the CVODE and RKF45 solvers were investigated.

We found that for the 9S4R reaction mechanism, the RKF45 solver was more efficient than the CVAN and CVDQ solvers on both CPU and GPU platforms. The acceleration on the GPU was significantly higher with the RKF45 solver due to lower control flow divergence compared to the complex BDF schemes. This result was consistent with prior studies by Stone and Davis and Niemeyer and Sung.[9,11] For the 45S201R mechanism, the baseline (i.e., dense stoichiometric matrix) CVAN solver using OpenMP was faster than the RKF45 solver using OpenMP. The 45S201R mechanism has many more reactions and intermediate species. Under the benchmark scenario, this may lead to a higher level of stiffness. The GPU RKF45 solver with the 45S201R reaction mechanism was 4.6 times faster than the dynamically load-balanced OpenMP version and was shown to be the fastest combination of solver and device for this mechanism and matrix structure.

The stoichiometric matrix storage structure was then investigated. A sparse storage format was implemented and exponentiation with integer stoichiometric coefficients was optimized. The latter was only applicable for the 45S201R reaction mechanism since the 9S4R mechanism has only nonintegral stoichiometric coefficients, a common difference between detailed and reduced-order mechanisms.

We observed a small but measurable performance improvement using the sparse matrix format with the 9S4R mechanism on the host CPU (i.e., MPI and OpenMP) with all ODE solver methods. The throughput was improved significantly more with the 45S201R mechanism due to the much higher matrix sparsity. The performance improvement with RKF45 was higher than the CVAN solver on the CPU since the matrix format and exponentiation optimizations were limited to the RHS function and a portion of the analytical Jacobian function. Other dense matrix operations in CVODE (e.g., factorization) were unaffected.

The sparse stoichiometric matrix optimizations had no impact on the GPU throughput with any ODE solver using the small 9S4R reaction mechanism. The GPU performance was increased with the larger 45S201R mechanism but to a lesser extent than that observed on the host CPU. The optimizations, particularly the *powi* optimization, reduced the floating-point intensity of the computations more than the bandwidth and this will have a greater impact on latency-optimized CPUs.

Overall, the reaction kinetics component with the reduced-order 9S4R reaction mechanism was accelerated 6.1 times over the baseline performance (MPI CVAN with dense format) using the best performing method (GPU RKF45 with sparse format) for the 128-k particle single-device runs. This performance improvement is significant and will have a direct impact on current DPD-RX simulations by permitting a reduction in run time or a proportional increase in the number of CG particles.

For the more chemically detailed 45S201R mechanism, the best method (OMP RKF45 on the host CPU using the *Sp-PowI* method) exceeded 60 times the 128-k particle single-device baseline performance. This level of acceleration indicates that the use of detailed reaction mechanisms—which have a broader modeling applicability and improved fidelity—are possible and attainable within the DPD-RX software. Moreover, these optimization efforts provide sufficient flexibility to optimally treat a given set of reaction equations, regardless of size, type, or stiffness and addresses a key simulation challenge for modeling the multiscale nature of the energy release and propagation mechanisms in advanced energetic materials.

Many of the optimizations presented in this report were incorporated into the USER-DPD package of the “28 Jun 2016” LAMMPS release. As a result of this work, the DPD-RX software enables efficient use of HPC resources to perform previously impractical micro- and mesoscale particle-based simulations. This allows new problems to be addressed and previously inaccessible phenomena to be studied within a simulation.

6. References

1. Maillet JB, Soulard L, Stoltz G. A reduced model for shock and detonation waves. II. The reactive case. Europhysics Letters. 2007;78(6).

2. Plimpton S. Fast parallel algorithms for short-range molecular dynamics. J of Computational Phys. 1995;117(1):1–19.

3. Brennan JK, Lisal M, Moore JD, Izvekov S, Schweigert IV, Larentzos JP. Coarse-grain model simulations of nonequilibrium dynamics in heterogeneous materials. J of Phys Chem Letters. 2014;5(12):2144–2149.

4. Hairer E, Wanner G. Solving ordinary differential equations II: stiff and differential-algebraic problems. Springer Series in Comp Math. 1996.

5. Linford JC, Michalakes J, Vachharajani M, Sandu A. Multi-core acceleration of chemical kinetics for simulation and prediction. SC09; Portland, OR. 2009.

6. Shi Y, Green WH, Wong H, Oluwole OO. Redesigning combustion modeling algorithms for the graphics processing unit (GPU): chemical kinetic rate evaluation and ordinary differential equation integration. Combustion and Flame. 2011;58(1).

7. Murray L. GPU acceleration of Runge-Kutta integrators. IEEE Transactions on Parallel and Distributed Systems. 2012;23(1).

8. Shi Y, Green WH, Wong H, Oluwole OO. Accelerating multi-dimensional combustion simulations using hybrid CPU-based implicit/GPU-based explicit ODE integration. Combustion and Flame. 2012;59:2388–2397.

9. Stone CP, Davis RL. Techniques for solving stiff chemical kinetics on graphical processing units. J of Propulsion and Power. 2013;67:764–773.

10. Niemeyer KE, Sung KE. Recent progress and challenges in exploiting graphics processors in computational fluid dynamics. J of Supercomputer. 2014;67:528–564.

11. Niemeyer KE, Sung CJ. Accelerating moderately stiff chemical kinetics in reactive-flow simulations using GPUs. J of Computational Phys. 2014;256:854–871.

12. Hindmarsh AC, Brown PN, Grant KE, Serban R, Shumaker DE, Woodward CS. SUNDIALS: suite of nonlinear and differential/algebraic equation solvers. ACM Transactions on Mathematical Software. 2005;31(3):363–396.

13. Fehlberg E. Low-order classical Runge–Kutta formulas with stepsize control and their application to some heat transfer problems. Marshall (AL): George C Marshall Space Flight Center; 1969 July. Report No.: NASA TR-R-315, 2969.

14. Prasad K, Yetter RA, Smooke MD. An Eigenvalue method for computing the burning rates of RDX propellants. Combustion Sci and Tech. 1997;124 (1–6):35–82.

15. Bell N, Garland M. Implementing sparse matrix-vector multiplication on throughput-oriented processors. Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC09); 2009; Portland, OR.

16. Nakashima H, Miyake Y, Usui H, Omura Y. OhHelp: a scalable domain-decomposing dynamic load balancing for particle-in-cell simulations. International Conference on Supercomputer; 2009.

17. LAMMPS. 2016 Sep 27 Patch. http://lammps.sandia.gov/bug.html.

List of Symbols, Abbreviations, and Acronyms

BDF	backwards differentiation formula
CG	coarse-graining
CUDA	Compute Unified Design Architecture
CVAN	CVODE with analytical Jacobian
CVDQ	CVODE with difference quotient
CVODE	C Variable Order Differential Equation
DPD	Dissipative Particle Dynamics
DPD-RX	Reaction Dissipative Particle Dynamics
EM	energetic materials
fs	femtosecond
GPU	graphics processing unit
HPC	high-performance computing
MPI	message passing interface
ODE	ordinary differential equations
RDX	crystalline cyclotrimethylenetrinitramine
RHS	right-hand-side
RKF	Runge-Kutta-Fehlberg
SIMT	single-instruction, multiple-thread
WI	workload intensity

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

2 (PDF) DIRECTOR US ARMY RSRCH LAB RDRL CIO L IMAL HRA MAIL & RECORDS MGMT

1 (PDF) GOVT PRINTG OFC A MALHOTRA

1 (PDF) BROWN DEER TECHNOLOGY D RICHIE

1 (PDF) GENERAL ELECTRIC M MASQUELET

6 (PDF) ENGILITY CORP J GRANICKI G KEDZIORA M LASINSKI J LILL G STATHOPOULOS H THORNBURG

1 (PDF) HÁLA LAB OF THERMODYNAMICS INSTITUTE OF CHEM PROC FUND OF THE ADV SCI COMP RSCH AND DEPT OF PHY JE PUKINJE UNIV M LÍSAL

2 (PDF) HPCMP F HILL R HEDGEPETH

1 (PDF) OAK RIDGE NAT LAB R SANKARAN

1 (PDF) OREGON STATE UNIV K NIEMEYER

3 (PDF) SANDIA NAT LAB S J PLIMPTON A P THOMPSON S MOORE

1 (PDF) US NAVAL RSRCH LAB I SCHWEIGERT

13 (PDF) US ARMY RSRCH LAB RDRL CIH C J KNAP K LEITER RDRL WML N J TRIVEDI RDRL WML B J K BRENNAN E F C BYRD J P LARENTZOS B RICE RDRL WML D C C CHEN M J MCQUAID M J NUSCA J VEALS RDRL WMP C R BECKER RDRL WMM G J ANDZELM