SAND2015-10550J

Optimizing the Performance of Reactive Molecular Dynamics Simulations for Multi-core Architectures

Hasan Metin Aktulga, Chris Knight, Paul Coffman, Tzu-Ray Shan, Wei Jiang

Abstract

Hybrid parallelism allows high performance computing applications to better leverage the increasing on-node parallelism of modern supercomputers. In this paper, we present a hybrid parallel implementation of the widely used LAMMPS/ReaxC package, where the construction of bonded and nonbonded lists and evaluation of complex ReaxFF interactions are implemented efficiently using OpenMP parallelism. Additionally, the performance of the QEq charge equilibration scheme is examined and a dual-solver is implemented. We present the performance of the resulting ReaxC-OMP package on a state-of-the-art multi-core architecture Mira, an IBM BlueGene/Q supercomputer. For system sizes ranging from 32 thousand to 16.6 million particles, speedups in the range of 1.5-4.5$\times$ are observed using the new ReaxC-OMP software. Sustained performance improvements have been observed for up to 262,144 cores (1,048,576 processes) of Mira with a weak scaling efficiency of 91.5% in larger simulations containing 16.6 million particles.

Index Terms

Reactive Molecular Dynamics, ReaxFF, LAMMPS-ReaxC-OMP, Multi-core architectures, Hybrid parallelism

1 Introduction

Molecular Dynamics (MD) simulation has become an increasingly important computational tool for a range of scientific disciplines including, but certainly not limited to, chemistry, biology, and materials science. In order to examine the microscopic properties of atomistic systems for many nanoseconds (and possibly microseconds) and distances spanning several nanometers, it is crucial to have a computationally cheap, yet sufficiently accurate interatomic potential to facilitate the required simulations. Several

- *H. M. Aktulga is with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824 and also with Lawrence Berkeley National Laboratory, Berkeley, CA 94720. E-mail: hma@cse.msu.edu*
- *C. Knight is with the Argonne Leadership Computing Facility, Argonne National Laboratory, Argonne, IL 60439. E-mail: knightc@anl.gov.*
- *P. Coffman is with the Argonne Leadership Computing Facility, Argonne National Laboratory, Argonne, IL 60439. E-mail: pcoffman@anl.gov.*
- *T-R. Shan is with the Materials Science and Engineering Center, Sandia National Laboratories, Albuquerque, NM 87185. E-mail: tnshan@sandia.gov.*
- *W. Jiang is with the Argonne Leadership Computing Facility, Argonne National Laboratory, Argonne, IL 60439. E-mail: wjiang@alcf.anl.gov.*

popular molecular force fields are readily available in the literature for modeling liquids, proteins, and materials (e.g. Charmm [1], Amber [2], and OPLS [3] to name a few). The computational efficiency of these models can be largely attributed to defining fixed bonding topologies within (and between) molecules, fixed partial charges, and the use of relatively simple functions to model the interatomic potential. While appropriate for many systems and problems, the use of fixed bonding topologies and charges prevents one from exploring microscopic processes involving chemical reactions or responses from environmental effects, which may be critical to properly understand the process of interest. Instead of resorting to computationally expensive quantum mechanical alternatives, which explicitly treat the electronic degrees of freedom and thereby are appropriate to model chemical reactions, one can employ simulation methods that include some degree of variable bond topology (e.g. multistate methods [4], [5]) or force fields that do not define a bonding topology. One class of the latter type of force fields are based on bond order potentials (e.g. ReaxFF [6], COMB [7], [8], AIREBO [9]). The goal of all such reactive methodologies and force fields is to model reactive systems at time and length scales that far surpass those currently impractical for exploration with electronic structure methods, but at the same time complementing these more accurate quantum mechanical methods. It is thus important to ensure that efficient implementations of a method are available in order to best address challenging scientific questions and best utilize available computational resources.

ReaxFF is a bond order potential that has been widely used to study chemical reactivity in a wide-range of systems. The PuReMD software [10], [11], [12] and the LAMMPS/ReaxC package [13], which is also based on PuReMD, provide efficient, open-source implementations of the ReaxFF model that have been beneficial to large communities of researchers. PuReMD and LAMMPS/ReaxC incorporate novel algorithms and data structures to achieve high performance in force computations while retaining a small memory footprint. The ability for a large community of researchers to efficiently carry out such simulations is becoming even more important as algorithms for the efficient fitting of ReaxFF models are made available [14], [15], [16], [17].

Just like strategies to accurately and efficiently model a challenging problem have evolved over time, so too has the translation of algorithms from paper to software matured to make optimal use of high-performance computing (HPC) resources. As a result of the physical limitations of the current chip technology, we have witnessed the emergence of multi-core architectures over the past decade. Hybrid parallelism (typically in the form of MPI/OpenMP) allows HPC applications to better leverage the increasing on-node parallelism. In this paper, we present hybrid parallel algorithms and their implementation for ReaxFF, where the construction of bonded and nonbonded lists and evaluation of complex interactions are implemented efficiently with a suitable choice of data structures and using thread parallelism provided by the OpenMP library. We present detailed performance analysis of the resulting LAMMPS/ReaxC-OMP package on a state-of-the-art multi-core system Mira, an IBM BlueGene/Q supercomputer. For system sizes ranging from 32 thousand to 16.6 million particles, speedups in the range of 1.5-4.5$\times$ are observed using the new hybrid parallel implementation. Sustained performance improvements have been observed for up to 1,048,576 processes in larger simulations.

2 ALGORITHMS AND IMPLEMENTATION

Algorithms, data structures and implementation details underlying the original ReaxFF software (PuReMD and LAMMPS/ReaxC package) with MPI parallelism are presented in detail by Aktulga *et al.* [11], [10]. In this paper, we focus on enabling efficient thread-parallelism for ReaxFF computations.

2.1 Motivation for a Hybrid Implementation

Parallel implementations based on spatial decomposition, where each MPI process is assigned to a specific region of the simulation box, is the most commonly used approach in MD simulations [10], [18], [19], [20]. The computation of bonded and short-ranged interactions then requires the exchange of atom position information near process boundaries, *a.k.a* the ghost region. The main benefits of a hybrid MPI/OpenMP implementation are the reduction of the number of MPI processes that must communicate, the amount of data exchanges between these processes and redundant computations at the ghost regions (if any). This is because nearest-neighbor communication (and related data duplication) is proportional to the surface area of the domain owned by an MPI process [18] and hybrid parallelization reduces the number of partitions for a given node count.

Below, we try to quantify this with a simple example where we assume a homogeneous (or random) distribution of atoms in a simulation box. The volume ratio of the ghost region to the original simulation domain in an MPI-only vs. MPI/OpenMP implementation would then reflect the relative communication and computation overheads in both schemes. For simplicity, let d denote the dimensions of the *cubic* region assigned to a process, g be the thickness of the ghost region (which is typically determined by the largest interaction cutoff distance), $t = c^3$ be the number of threads on a node for some integer $c \geq 1$ and n be the number of nodes used. Then the total ghost volume in MPI-only and MPI/OpenMP hybrid implementations would respectively be:

$$\begin{aligned} V_{mpi} &= nc^3((d+g)^3 - d^3) = n((cd+cg)^3 - (cd))^3 \\ V_{mpi-omp} &= n((cd+g)^3 - (cd)^3) \end{aligned} \tag{1}$$

Figure 1 shows the relative volume of the ghost region to the original simulation domain under weak-scaling scenario with increasing number of cores t and various d/g ratios. Using MPI-only parallelization, the relative volume is constant and it is considerably high for low values of d/g. Under MPI/OpenMP parallelization, there is a single partition per node and therefore the relative volume of the ghost region decreases as the number of cores increases. As we show in Figure 2, the reduction in the total ghost region volume can be significant for modern architectures like the IBM BG/Q and Intel Xeon Phi based systems.

It may be argued that in an MPI-only implementation, expensive inter-process communications may be turned into intra-node communications by mapping the c^3 nearby MPI processes onto the same node (*i.e.*, using topology aware mapping techniques [21], [22], [23]). As a result, for most classical molecular dynamics models, increased ghost region volume would result in memory overheads, but may not incur significant computational overheads except for building the neighbor lists. In such cases, a hybrid parallel implementation may not yield significant gains for small scale computations, but for capability scale simulations on large supercomputers, leveraging thread parallelism will still be very important.

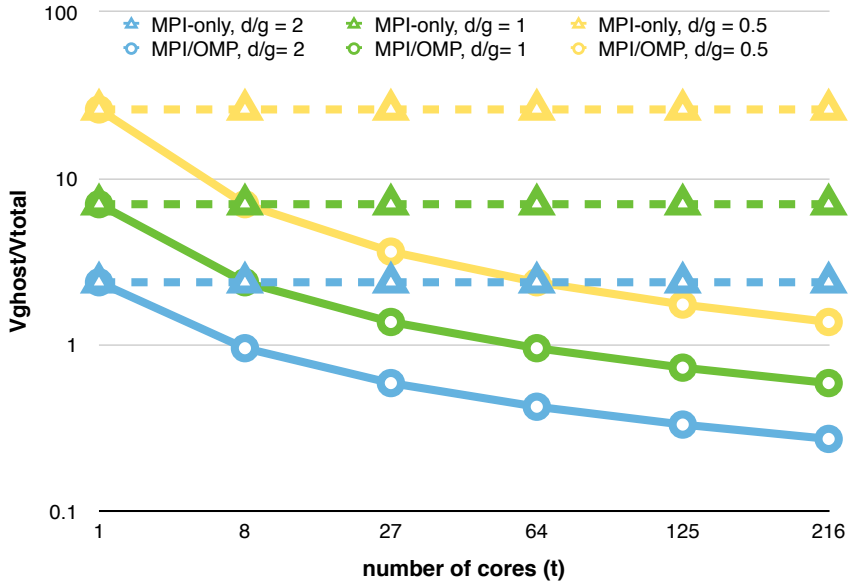

Fig. 1. Ratio of the ghost region to the actual simulation domain in MPI-only vs. MPI-OpenMP parallelization under the weak-scaling scenario with increasing number of cores on a single node.

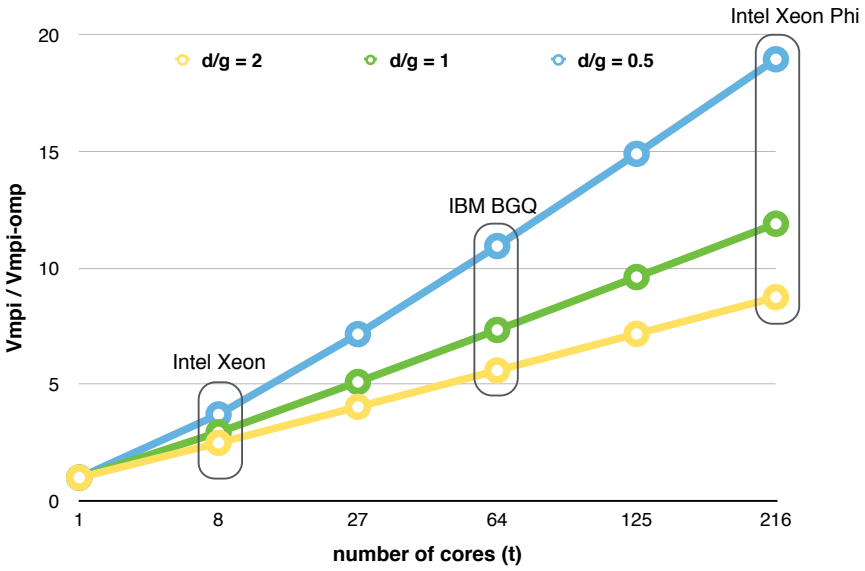

Fig. 2. Relative ghost region volumes in MPI-only vs. MPI-OpenMP parallelized molecular dynamics simulations with spatial decomposition. We mark the core counts for typical multi-core and many-core processors available today.

For ReaxFF computations, hybrid MPI/OpenMP parallelization is crucial in terms of performance for two unique reasons. First, the dynamic nature of bonds in ReaxFF and the presence of valence and dihedral interactions that straddle long distances into process boundaries require significant number of bonded computations to be repeated in the ghost regions of multiple processes. Unlike most classical MD packages, computational expense of bonded interactions in ReaxFF are comparable to that of non-bonded interactions [11]. Using the TATB benchmark example available in LAMMPS, we observe that the ratio of the computational expenses of bonded and non-bonded interactions is approximately 3/2 for one step, 60.77% vs. 39.23% to be exact (this ratio will show variations depending on the cutoffs used and the specific system being simulated). Therefore in the strong scaling limit as d gets comparable to or less than g, increased ghost region volumes are likely to cause significant computational overheads in ReaxFF computations.

A second reason is the inter-node communication overheads during the charge equilibration (QEq) procedure [24]. To determine partial charges on each atom, it is necessary to solve a large linear system of equations at each step in a ReaxFF simulation. For this purpose, iterative linear solvers that require a forward-backward halo-exchange of partial charges at each iteration are used [11]. These communications become a performance bottleneck on a large number of MPI ranks. Since hybrid parallelism will reduce the number of MPI ranks and ghost particles involved, it is expected to reduce the onset of communication-related performance bottlenecks.

These two limitations in current ReaxFF simulations have been our primary motivation for a hybrid parallel ReaxFF implementation. As we show through extensive tests in Section 3, we achieved significant performance improvements by leveraging the hybrid MPI/OpenMP programming model.

2.2 General Thread Parallelization Strategy

The challenge in a shared memory parallel implementation is eliminating *race conditions* and ensuring balanced workloads are distributed among threads. We may summarize the general implementation scheme in the new ReaxC-OMP package as partitioning of atoms among threads. To prevent race conditions, we use thread-private arrays during force updates and OpenMP reductions for energy updates. We first discuss the rationale for this choice and note specific implementation issues regarding each interaction later in this section.

The general work flow in ReaxFF is to compute various atomic interaction functions (bonds, lone pair, over-/under-coordination, valance angles, dihedrals, van der Waals and Coulomb) to the local atomic system (including ghost particles) and then sum various force contributions at the individual atomic level to obtain the net force on each atom for a given time step. Potential energies are computed at the system level. Energy and force computations, although disparate in their mathematical formulations, are aggregated in the same global data structures, with those related to forces being uniquely indexed for each (local and ghost) atom. The force computation functions all share a general methodology of computing the energies and forces in an atom centered fashion, defining an interaction list for each atom, calculating the force between a given atom and each of its neighbors, and then aggregating the forces on individual atoms and the potential energy of the system. More specifically, the algorithms consist of an

outer loop over a data structure containing all atoms in the local system, and an inner loop over the neighbors of a given atom where most of the computation takes place (see Algorithm 1 for an example). Performance counters instrumented within each function around these loops identified them as targets for performance improvements via OpenMP multi-threading. The ensuing tuning effort utilized these counters to precisely measure the OpenMP speedups.

Algorithm 1 Pairwise force computation

Input: Atom list and positions
Output: Potential energy and forces (partial)

```
1:  for (int i = 0; i < numAtoms; i++) do
2:      nbrList[] = getNeighbors(i);
3:      for (int j = 0; j < len(nbrList[]); j++) do
4:          k = nbrList[j];
5:          globalEnergy += computeE(atoms[i], atoms[k]);
6:          f_{i,k} = computeF(atoms[i], atoms[k]);
7:          globalForce[i] += f_{i,k};
8:          globalForce[k] -= f_{i,k};
9:      end for
10: end for
```

Data Privitization: As outer loops of interaction functions (line 1 in Alg. 1) were identified to be the targets for multi-threading, these loops were made OpenMP parallel by dividing the atoms among threads, and thread privatizing certain local variables. Interactions in the Reax force field are complex mathematical formulas that are expensive to compute. Therefore all interactions (pairwise, three-body and four-body) are evaluated once and the resulting forces are applied to all atoms involved in the interaction. In a thread parallel computation, this situation creates race conditions on the global force data structure, as atoms assigned to different threads may be neighbors of each other or they may have common neighbors. Incurring thread locks via the OpenMP critical directive within the inner loop was observed to be very inefficient due to the increasing overhead of using the lock. This wiped out most of the performance gains in our test systems when using more than a couple of threads. Therefore we employ a scheme based on the thread-privitization of force arrays. Instead of a thread updating the global force data structure directly at the inner loop level, each thread is allocated a private force array at the start of the simulation which they update independently during force computations. After all force computations are completed, thread-private force arrays are aggregated (reduced) into the global force array to compute the final total force on each atom. Despite the performance overhead of this additional reduction step, the data-privitization methodology was much more efficient than thread locks and scaled well with large numbers of threads (up to 16 as discussed in Section 3).

In our OpenMP implementation, system energy tallies are handled with relatively little performance overhead via the OpenMP reduction clause at the outer loop level. Additionally, electrostatic and virial forces need to be tallied for each pairwise interaction. The original MPI-only implementation utilized pre-existing serial functions within the pair-wise force field base class (Pair) in LAMMPS for this purpose. We now utilize the

threaded versions within the LAMMPS/USER-OMP package following a methodology consistent with other threaded force field implementations that employs setup, tally and reduction functions appropriately in place of the serial versions within the algorithms.
Thread Scheduling: In OpenMP, static scheduling is the default work partitioning strategy among threads. In our outer loop parallelization scheme described above, static scheduling would partition n atoms into t chunks (t being the number of threads) consisting of approximately n/t contiguous atoms in the list. While static scheduling incurs minimum runtime overheads, such a partitioning may actually lead to load imbalances because some atoms may have a significantly large number of interactions in comparison to others in a system where atoms are not distributed homogeneously throughout the simulation domain. Also some atoms may be involved in a large number of 3-body, 4-body and hydrogen bond interactions, while others may have none due to specifics of the ReaxFF model and relevant chemistry. As an illustrative example, a plot of the assignment of candidate valence angles to atoms from the LAMMPS/FeOH3 example on a single process is shown in Figure 3. In LAMMPS, atom list is reordered based on spatial proximity to improve cache performance, and in this particular case, the majority of valence angle interactions involve atoms appearing at the beginning of the atoms list. In this test case, more than 80% of atoms in the system do not own any angle interactions. As a typical simulation progresses and atoms migrate between MPI processes, the ownership of atoms between processes or the ordering within the atom list may change, but the imbalance of per-atom work would remain. Therefore statically scheduling the work across threads in contiguous chunks of the atom list can degrade performance as some (most) threads can own considerably more angles than the average number. This example focused on valence angle interactions, but similar workload distributions exist for other interactions due to the chemical nature of the species simulated, making this a general issue to be mindful of.

For the majority of cases, the use of the dynamic scheduling option in OpenMP was found to ensure a good balance of work among threads as opposed to explicitly assigning per-thread work beforehand. However, there exists an important trade-off regarding the chunk size. Smaller chunks are better for load balancing, but they may incur significant runtime overheads (default chunk size for dynamic scheduling is 1). Larger chunks reduce scheduling overheads, but with larger chunks load balancing is harder to achieve and the number of tasks that can be executed concurrently decreases. In the new ReaxC-OMP package, we empirically determined the scheduling granularity. For example, comparing the performance for the 16.6 million particle benchmark on 8,192 BG/Q nodes, a chunksize of 20 atoms gives slightly better performance using 8 MPI ranks and 8 OpenMP threads per rank on each node (Figure 4). As the number of threads per MPI rank increases (and number of MPI ranks per node decreases), a chunksize of 25 was found to be optimal. For the chunksizes sampled in the range 10-50, a maximum deviation of 6% in performance was observed relative to the performance with the smallest chunksize when running 64 threads per MPI rank. While this parameter needs to be tuned for ideal performance depending on the simulated system and the architecture, its default value is set to 20.
Memory Overheads: One potential drawback to our thread-private arrays approach is the increased memory utilization because the force array is duplicated for each thread. For a big local system using a simple force field, this approach might result

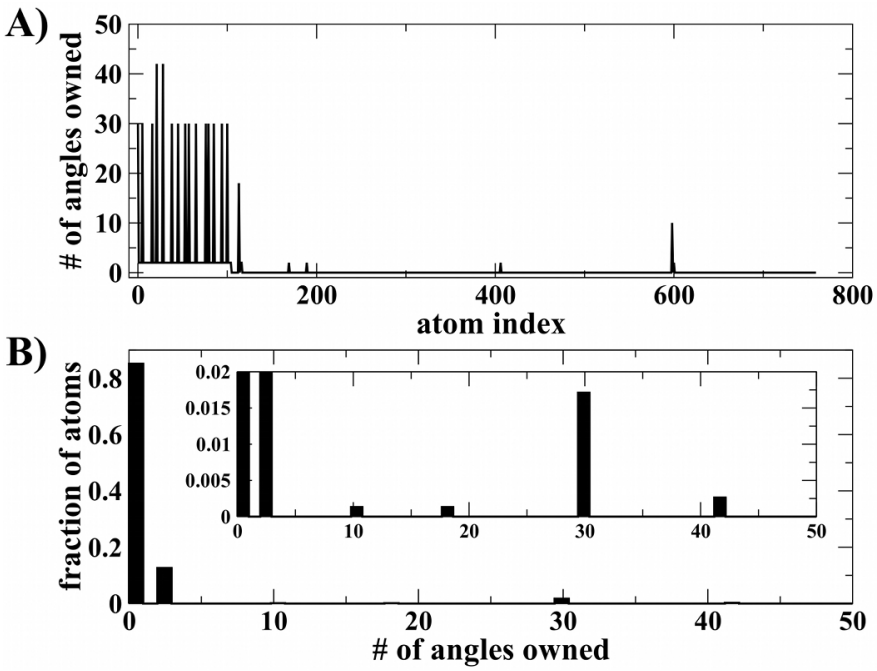

Fig. 3. A) Illustrative example of how the assignment of valence angle interactions to atoms generates an imbalanced distribution for the LAMMPS FeOH3 example. B) Fraction of atoms with specified count of angles owned. The inset shows a magnified view of the fraction of atoms with the majority of assigned work. This is one example where a naive assignment of work to threads is inefficient and degrades performance when scaling to a large number of threads.

in significant memory overheads if the number of threads is large. In ReaxFF, the data structures that require major memory space are the neighbor, bonds, 3-body and hydrogen bond lists. In these lists, the number of interactions per atom may range from tens to hundreds and as we discuss below there is no duplication of these data structures in the ReaxC-OMP package (except for the optional duplication of the neighbor list). In comparison, the force array only stores the force on an atom in x, y, z dimensions. So under typical simulation scenarios, the duplication of force arrays are not likely to cause significant overheads in terms of memory usage. Also note that the main challenge for ReaxFF simulations is to access sufficiently long timescales where interesting scientific phenomena can be observed. Increasing the number of timesteps that can be simulated per day requires keeping the number of atoms per MPI rank (the local system) relatively small, therefore the memory footprint of ReaxFF simulations is typically not a major bottleneck.

2.3 Implementation Details

In this section, we present implementation details regarding key computational kernels and data structures in ReaxC-OMP.

Neighbor and Interaction Lists: Neighbor lists generated by LAMMPS at the request of a force field contains only the neighboring pair information. ReaxC-OMP maintains a separate neighbor list with more detailed information like pair distance and distance

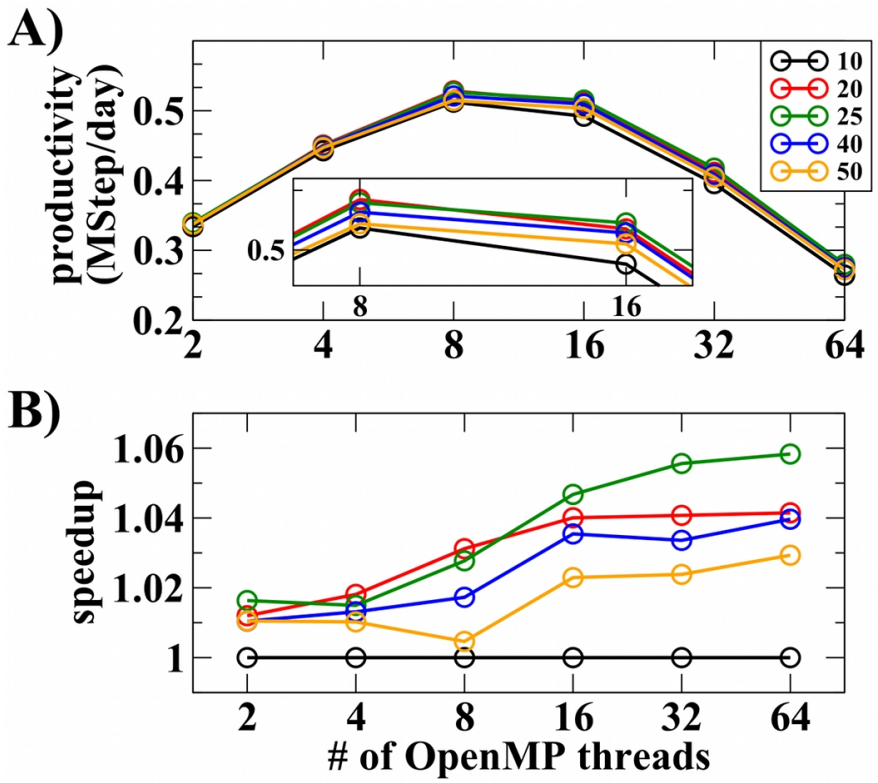

Fig. 4. A) Productivity as a function of number of OpenMP threads for a range of chunksizes observed for the 16.6 million particle PETN benchmark on 8,192 BG/Q nodes. B) The observed speedup relative to a chunksize of 10. The inset in A) shows a magnified view for the performance of 8 and 16 threads.

vector as these quantities are needed multiple times during the construction of the bond list and the hydrogen bond list, as well as during force computations. The neighbor list is stored by default as a half list, *i.e.*, for neighboring atoms i and j, only a single record is kept. A compact adjacency list format (similar to the compressed row format in sparse matrices) is used for storing the neighbor list.

While a half list is advantageous for reducing the computational and storage costs of the neighbor list, it brings challenges in generating the bond and hydrogen bond lists. Efficient on-the-fly construction of 3-body and 4-body interaction lists requires the bond list to be a full list with both i-j and j-i bonds available. The hydrogen bond list is generated based on the surrounding atom information of a covalently bonded H atom; but this information would be spread throughout the neighbor list if it is stored as a half list.

In ReaxC-OMP, bond and hydrogen bond lists are generated by making the outer loop over the neighbor list OpenMP parallel. For efficiency, both of these lists are generated by making a single pass over the neighbor list and, if needed, updating the bond or hydrogen bond lists of atoms i and j concurrently. As in the force computations, the challenge is in the inner loop where race conditions may arise due to updates to the bond or hydrogen bond lists of common neighbors. In both cases, race conditions are prevented by introducing *critical regions* that can be executed by a single thread at any

given time. For a thread which needs to update the bond or hydrogen bond list of atom j while processing the neighbors of atom i, the critical region only includes the reservation of a slot in the relevant list. Once a slot is reserved, all subsequent updates are performed outside the *critical region*. In this way, performance penalties associated with *critical regions* is reduced by limiting them to be very short code sequences. We have found the combination of a half-list to store neighbors and the use of critical regions to give good overall performance on moderate number of threads (up to 16) as discussed in the performance evaluation section.

Pairwise Interactions: Bond order correction, bond energy and non-bonded interaction computations (*i.e.* van der Waals and Coulomb interactions) constitute the pair-wise interactions in ReaxFF. As described above, these interactions are made OpenMP parallel at the outer loop level and race conditions are resolved through OpenMP reductions for energies and thread-privitization of the force array. In Alg. 2, we give a simple pseudo-code description of the van der Waals and Coulomb interactions in the non-bonded force computations to illustrate this idea.

Algorithm 2 Threaded non-bonded pairwise force computation

Input: Atom list and positions
Output: Potential energy and forces (partial)

```
1: #pragma omp parallel reduction (+:PotEng) {
2: tid ← omp_get_thread_num();
3: PairReaxC–>evThreadSetup(tid);
4: #pragma omp for schedule(dynamic)
5: for (int i ← 0; i < numAtoms; i++) do
6:     nbrList[] ← getNeighbors(i);
7:     for (int j ← 0; j < len(nbrList[]); j++) do
8:         k ← nbrList[j];
9:         evdW, fvdW ← computeVDWaals(atoms[i], atoms[k]);
10:        eClmb, fClmb ← computeCoulomb(atoms[i], atoms[k]);
11:        PotEng += (evdW + eClmb);
12:        tprivForce[tid][i] += (fvdW+fClmb);
13:        tprivForce[tid][k] -= (fvdW + fClmb);
14:        PairReaxC–>evThreadTally(tid);
15:    end for
16: end for
17: PairReaxC–>evThreadReduction(tid);
18: Reduce tprivForces onto globalForce array
19: }
```

Three-body Interactions: One particular challenge in ReaxFF is the dynamic nature of the three-body interactions list. Whether an atom contributes to a three-body valence angle interaction depends on the molecular identity and the surrounding environment of the atom. As such, not all atoms in a system may be involved in a three-body interaction. Additionally, depending on the nature of the molecular species being simulated, only a subset of atoms in the system are designated as the *central atom* of an angle (e.g., see Figure 3).

The three-body interactions are dynamically formed based on the bonds of *central atoms*; and they need to be stored in a separate list because four-body interactions are generated based on the three-body interactions present at a given time step. Storing a three-body interaction information is expensive in terms of memory, and the number of interactions per atom can vary significantly as shown in Figure 3. Therefore, we first identify which angles are present at a time step without storing them. After all angles have been identified, a per-atom prefix sum is computed. The 3-body interactions are then computed and stored using the global array offsets to eliminate memory clashing between threads.

QEq: The dynamic bonding in ReaxFF requires the re-distribution of partial charges at every step. LAMMPS/ReaxC uses the the charge equilibration method (QEq) [11], [24] which models the charge re-distribution as an energy minimization problem. Using the method of Lagrange multipliers to solve the minimization problem, two linear systems of equations are obtained with a common kernel H, an $N \times N$ sparse matrix where N is the number of atoms. H denotes the coefficient matrix generated from a truncated electrostatics interaction and well-known Krylov subspace methods (CG [25] and GMRES [26]) can be used to solve the charge re-distribution problems [11].

An effective extrapolation scheme that we developed for obtaining good initial guesses and a diagonally preconditioned parallel CG solver yield satisfactory performance for charge equilibration. This QEq solver has been implemented as the `fix qeq/reax` command in LAMMPS. In the new ReaxC-OMP package, we adopted a concurrent iteration scheme [10] which combines the sparse matrix multiplication and orthogonalization computations for the two linear systems involved in charge equilibration. This concurrent iteration scheme helps reducing communication overheads.

OpenMP threading was applied to several computational loops within the QEq solver, most significantly the sparse matrix vector multiplication and the construction of the Hamiltonian matrix from the neighbor list. Taking advantage of the fact that the QEq Hamiltonian is symmetric, only unique, non-zero elements of the sparse matrix are computed and stored. Using an atom-based prefix sum, the effort to compute the Hamiltonian matrix is efficiently distributed across threads avoiding potential race conditions to improve performance.

2.4 Experience with Transactional Memory

As an alternative to the thread data privitization strategy detailed previously in Section 2.2, the usage of transactional memory was also explored on Blue Gene/Q. Transactional memory (TM) atomics (tm_atomic directive) are supported by the IBM XLC compiler and allow their nesting within OpenMP parallel regions. A typical implementation consists of essentially replacing OpenMP critical directives with tm_atomic in the application code, and passing -qtm on the command line during compilation. Blue Gene/Q implements TM support at the hardware level within the L2-cache by tracking memory conflicts for the atomic transaction group. If conflicts are found a rollback is executed for the entire transaction, restoring the state of the memory, and then the transaction is re-executed. In this fashion multiple threads can execute the code on shared memory data concurrently without incurring the overhead of locks. However, there are other performance factors to consider. There is some performance overhead in generating the atomic transaction each time it executes, and significant overhead if

a conflict is found and a rollback occurs. So the key to TM performance is to have a significant amount of work in the transaction while avoiding frequent conflicts with other threads. There are runtime environment variables supported by the XL compiler that tell the application to generate reports detailing the runtime characteristics of the transactions. These reports can give clues regarding the impact of TM on performance and be used to guide further tuning of the code.

This approach was attempted in several of the energy and force computation functions in ReaxC-OMP, where a tunable number of iterations in the inner-loop pairwise computations was chunked together into one transaction. However, no significant performance improvement over the baseline OpenMP implementation, where race conditions were resolved using *critical* sections, could be attained with any number of iterations. With a small number of iterations in a chunk, there were few conflicts but a lot of transactions, so the transaction generation overhead prevented any speedup. When the chunk size was increased, there were larger but fewer transactions. In this case the increased number of conflicts resulted in a significant number of rollbacks which again prevented any speedup. In these computations, the TM conflicts arose because disparate threads were executing pairwise computations with common neighbors based on the division of labor occurring on the outer atomic index loop, but the atomic index has little correlation with spatial decomposition. As a result, the ReaxC-OMP package does not use the TM strategy.

2.5 A Tool for Molecular Species Analysis

An important need in reactive molecular simulations is the analysis of molecular species. The conventional way for this task is to do post-processing using the trajectory output, however, there are several disadvantages to this approach for ReaxFF simulations. First, each snapshot is rather large; bonding information typically requires 100-1000 bytes/atom. Second, the trajectory output frequency must be higher than the fastest reactive process in the system, even if this process only involves a small subset of all atoms. Third, sub-sampling and time-averaging of bonding information is required in order to distinguish persistent bonds from transient encounters due to thermal and ballistic collisions. Lastly, post-processing software typically only operates on single processor. Due to all these factors, even for modest systems sizes (*e.g.* less than 100,000 atoms and 100 cores), time spent performing I/O and post-processing chemical species analysis greatly exceeds the time spent running the MD simulation itself.

To cope with this problem, we developed a real-time *in situ* molecular species analysis capability integrated within the LAMMPS simulation, `fix reax/c/species` command. This command uses the same spatial decomposition parallelism taking advantage of the distributed data layout and achieves comparable scaling performance to the MD simulation itself. Bonds between atoms, molecules and chemical species are determined as the simulation runs, and concise summary that contains information on the types, numbers and locations of chemical species is written to a file at specific time steps. As a result, users are now able to monitor the chemical species and chemical reactions in *real time* during large-scale MD simulations with reactive potentials, instead of attempting to analyze huge trajectory files after the simulations have finished. The *in situ* molecular species analysis algorithm can be summarized with the following steps:

- A pair of atoms, both with unique global IDs, are deemed to be bonded if the bond order value between the two atoms is larger than a threshold specified for this interaction type (default value is 0.3). A molecule ID, that is the smaller value of the two global IDs, is assigned to the pair of atoms. This process is repeated till every atom has been assigned a molecule ID.
- Sorting is performed for all molecule IDs and molecule IDs are reassigned from 1 to N, where N corresponds to the maximum number of molecule IDs. This number N also indicates the number of molecules in the system.
- Unique molecular species are determined by looping through each of the molecules and counting the number of atoms of each element. Molecules with the same number of atoms of each element are identified as the same species. One drawback of this algorithm is that it does not distinguish isomers.
- Finally, each distinct species and their counts are printed out in a concise summary.

In-situ analysis of physical observables such as bond order, bond lengths and molecular species can be beneficial so long as the analysis time remains a small percentage of the simulation time. In this case, one benefits from the distribution of data structures within the simulation code to compute observables in parallel while data is still in memory. As we show in the performance evaluation section, the analysis through the `fix reax/c/species` command exhibits a similar scaling behavior as the simulation itself and incurs only 5-25% performance overhead which is a significant improvement over the cost of the I/O intensive post-processing method. Additionally, the physical observables can be stored and averaged to determine bonds between atoms based on time-averaged bond order and/or bond lengths, instead of bonding information of specific, instantaneously sampled time steps. Such a capability is provided by the time-averaging of per-atom vectors function (`fix ave/time`) in LAMMPS.

3 Performance Evaluation

Performance benchmarks were executed on the 48-rack IBM Blue Gene/Q Mira system at Argonne. Each compute node on Mira contains a PowerPC A2 1.6 GHz processor with 16 cores per node (4 hardware threads per core for a total of 64 threads per node) and 16 GB RAM. Mira's 49,152 compute nodes are connected with each other using a proprietary 5D torus interconnection network to provide a peak computing speed of 10 petaflops. Calculations in this study utilized up to 16 BG/Q racks with 1024 compute nodes per rack.

The PETN crystal benchmark available on the LAMMPS website was used in performance evaluation studies. Replicas of the PETN system containing up to 16.6 million particles were examined. In all benchmark tests, charge equilibration (QEq) was invoked at every step with a convergence threshold of 10^{-6}.

Active Idling: Since the general threading scheme in ReaxC-OMP consists of several parallel regions independently implemented across disparate functions, the execution path of the code oscillates between threaded and non-threaded regions. The shared memory parallel (SMP) runtime treatment of idle threads could have a significant impact on performance. It is optimal in this case for the threads to continue spinning and remain as active as possible in between the thread-parallel regions, so that when they again have work to do, they can resume efficiently. In OpenMP, this is achieved by setting the runtime environment variable OMP_WAIT_POLICY to be ACTIVE, which is

applicable on all platforms with OpenMP support. In addition, BG/Q systems provide the BG_SMP_FAST_WAKEUP environment variable to reduce the thread wake up time which has been set to YES for performance evaluations reported in this paper.

3.1 Performance, Scalability and Validation

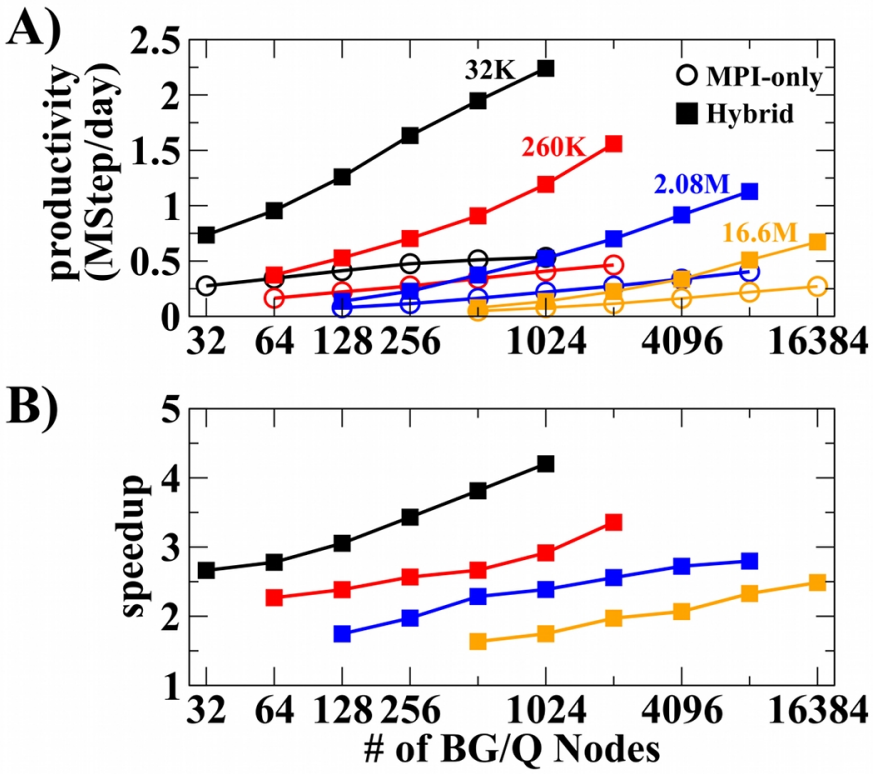

Fig. 5. A) Measured productivities of original MPI-only (open circles) and newest hybrid (solid squares) Reax/C implementations reported as millions of timesteps (MSteps) per day for four systems sizes of a PETN crystal benchmark: 32,480 (black), 259,840 (red), 2,078,720 (blue), and 16,629,760 (orange) particles. B) Relative speedups of hybrid vs. MPI-only implementations.

To examine the performance as a function of the number of MPI processes used per node and OpenMP threads used per MPI process, we performed a benchmark test on the PETN crystal replicated up to 16.6 million atoms. A range of 32 to 16,384 BG/Q nodes (each with 16 cores and 64 hardware threads) was used to best sample a representative range of HPC resources typically available to a user. The number of MPI processes per node varied in powers of 2 from 1 to 64 while the number of OpenMP threads per MPI process varied from 1 to $64/N_{MPI}$, where N_{MPI} is the number of MPI processes (so all of the available 64 hardware threads were used). 100-2000 MD step simulations were performed depending on system size with a standard setup, i.e., 0.1 fs time step size and re-neighboring checked every 10 MD steps. Trajectory files are not written (per the original benchmark available on the LAMMPS website [13]), thus, the timings reported in this section are representative of only the computation and communication costs of the ReaxFF simulations. With support in LAMMPS for MPI-IO and the writing

of trajectory files per subset of MPI ranks, the performance costs associated with I/O are expected to be in the 1-5% for these system sizes.

3.1.1 Performance Improvements

To quantify the performance improvements achievable with the hybrid parallel ReaxC-OMP package, the original MPI-only ReaxFF implementation in LAMMPS, i.e. USER-ReaxC package, was used as the baseline case. Performance results for both codes on the PETN crystal benchmark are plotted in Figure 5 with systems ranging from 32 thousand to 16.6 million particles. Each data point for the hybrid implementation in Figure 5 represents the optimal timing observed with respect to the number of MPI ranks and OpenMP threads per node. Optimal performance was typically observed using 8 MPI ranks with 8 OpenMP threads each. In some cases, a configuration of 4 MPI ranks with 16 threads has given slightly better performance (see below for a more detailed discussion). For our smallest system (32K atoms), the overall execution time on 1,024 BG/Q nodes for the hybrid code was 4.2 times faster than the MPI-only code. With the larger system sizes (2.08 M and 16.6 M atoms), consistent speedups of 1.5x to 3x (Figure 5b) were observed. Note that the higher speedups achieved with smaller systems is due to the higher communication and redundant computation to useful computation ratio in these systems as discussed in Section 2.1.

Performance numbers given in Figure 5a shows that the hybrid code also exhibits excellent weak scaling efficiency. Taking the performance of the 32K particle system on 32 nodes as our base case, for the hybrid implementation we have observed a weak scaling efficiency of 96% with 260 K particles on 256 nodes, 93% with 2.08 M particles on 2,048 nodes and 91.5% with on 16.6 M particles on 16,384 nodes.

Overall, the productivity (number of steps per day) gains with the original MPI-only code remains modest even on large systems. On the other hand, we observed that productivity with the hybrid parallel implementation continues to improve with the usage of more resources (number of nodes). Since one of the main bottlenecks in computational studies using MD simulations is the extremely long wall-clock times needed to reach simulation time-scales where interesting scientific phenomena can be observed (nanoseconds and beyond), from users' perspective, this is a very important capability provided by the hybrid implementation.

3.1.2 Detailed Performance Analysis

Next, we compare the speedups on a kernel basis obtained by the hybrid implementation running with the ideal number of threads over its single threaded execution. Single threaded execution is used here as a proxy for the MPI-only version which lacks the concurrent iteration scheme for QEq computations and indeed provides better overall performance than the MPI-only version. In single-threaded simulations, 32 cores out of the 64 available have been used, as we observed that 32 MPI ranks per node yielded better overall performance in comparison to using all available cores. This is likely due to the increased overheads on large number of MPI ranks, as well as limited cache space available on the IBM BG/Q architectures, which has 16 KB private L1 cache per core and 32 MB shared L2 cache. Note that limited cache space per core is a commonly observed trend on modern multi-core and many-core systems. Simulations with the hybrid implementation, on the other hand, used 4 MPI processes with 16 OpenMP threads per node.

	32 Nodes			1024 Nodes		
Kernel	**Single**	**Hybrid**	**Speedup**	**Single**	**Hybrid**	**Speedup**
Write Lists	34.7	6.5	5.3x	19.6	2.3	8.5x
Init. Forces	29.8	12.1	2.5x	16.7	2.6	6.4x
Bond Orders	11.3	1.8	6.3x	6.3	0.53	11.9x
3-body Forces	5.4	2.0	2.7x	2.8	0.34	8.2x
4-body Forces	3.7	2.3	1.6x	1.9	0.40	4.8x
Non-Bonded For	6.9	6.7	1.03x	0.48	0.46	1.04x
Aggregate For	19.8	3.6	5.4x	11.7	1.7	6.9x
QEq	15.5	22.4	0.7x	12.2	12.0	1.02x
Other	0.27	0.69	0.4x	0.07	0.17	0.4x
Total time	131.3	59.3	2.2x	77.9	21.8	3.6x

TABLE 1
Timing breakdown in seconds for the single-threaded and ideal configuration executions for the hybrid implementation on 32 and 1024 BG/Q nodes for key phases of ReaxFF simulation with the PETN crystal benchmark containing 32 thousand particles. These simulations have been executed for 500 steps. Single-threaded calculations used 32 MPI ranks per node, while the ideal configuration was determined to be 4 MPI ranks per node and 16 OpenMP threads per rank.

Table 1 gives a breakdown of the timings for the key phases in the PETN crystal benchmark containing 32 thousand particles. For this system, the thickness of the ghost region is 10 Å and the size of the simulation box is 66.4 Å x 75.9 Å x 69.9 Å. The kernels that involve significant redundant computations at the ghost regions are *write lists*, which computes neighbor atom information and distances, *init forces*, which initializes the bond and hydrogen bond lists, *bond orders*, *3-body forces*, *aggregate forces* and to some extent *4-body interactions*. Note that most of these kernels are bond related computations. With the hybrid implementation, we observe significant speedups in all these kernels as the hybrid implementation reduces redundancies at process boundaries. On 1024 nodes, the achieved speedups increase even further, as the ratio of ghost region to the actual simulation domain increases considerably when using only 1 thread.

We do not observe any significant speedup for *nonbonded forces*, which is expected because this kernel avoids redundant computations in ghost regions as described in Section 2.3. Contrary to our expectations though, for the *QEq* kernel, our hybrid implementation has performed worse than the single-threaded execution on 32 nodes (15.5 s vs. 22.4 s), and only slightly better on 1024 nodes (12.2 s vs. 12.0 s). The QEq kernel is an iterative solver consisting of expensive distributed sparse matrix vector multiplications (SpMV) in the form of $Hx_i = x_{i+1}$ followed by a halo exchange of partial charges at each step. The QEq matrix H is a symmetric matrix, and the original MPI-only implementation exploits this symmetry for efficiency. In the hybrid implementation, we opted to continue exploiting the symmetry and resolved race conditions between threads by using private partial result vectors for each thread. Our tests show that this is computationally more efficient than not exploiting the symmetry at all (which would increase SpMV time by a factor of 2), but still does not perform as well as the MPI-only SpMV computations due to increased memory traffic and cache contentions associated with private result arrays. This performance degradation in SpMV computations takes away the gains from reduced communication overheads achieved with the hybrid im-

	64 Nodes			2048 Nodes		
Kernel	**Single**	**Hybrid**	**Speedup**	**Single**	**Hybrid**	**Speedup**
Write Lists	11.1	2.7	2.3	4.8	0.6	8.0
Init. Forces	9.6	6.6	1.4	4.1	0.8	5.1
Bond Orders	3.4	0.7	4.8	1.6	0.2	8.0
3-body Forces	1.8	0.8	2.2	0.7	0.2	3.5
4-body Forces	1.4	0.8	1.8	0.4	0.1	4.0
Non-Bonded For	5.8	4.3	1.3	0.3	0.3	1.0
Aggregate For	5.5	1.3	4.2	2.8	0.4	7.0
QEq	7.5	7.7	0.97	2.4	2.8	0.85
Other	0.18	0.34	0.53	0.01	0.06	0.17
Total time	46.4	25.5	1.8	17.7	5.7	3.1

TABLE 2
Timing breakdown in seconds for the single-threaded and ideal configuration executions for the hybrid implementation on 64 and 2048 BG/Q nodes for key phases of ReaxFF simulation with the PETN crystal benchmark containing 260 thousand particles. These simulations have been executed for 100 steps. Single-threaded calculations used 32 MPI ranks per node, while the ideal configuration was determined to be 4 MPI ranks per node and 16 OpenMP threads per rank.

plementation. As a result, for smaller node counts, the QEq computations are carried out more efficiently using single-threaded execution.

Note that the increased memory traffic and cache contention issues are also present in other kernels due to the use of thread-private arrays. However, those kernels perform several floating point operations per force or bond update, and the number of threads in these tests have empirically been optimized for best performance. On the other hand, in SpMV computations, only two floating point operations (multiply and add) are needed for each non-zero matrix element. The relatively low arithmetic intensity of the QEq kernel explains the poor performance obtained in this kernel. Our future work on the LAMMPS/ReaxC code will focus on the development of more efficient SpMV algorithms that eliminate the use of thread-private arrays and are customized based on the sparsity structures of QEq matrices.

In Table 2, we present a similar breakdown for the PETN crystal benchmark with 260 thousand atoms on 64 and 2,048 BG/Q nodes. In this case, the observed speedups on a per kernel basis are relatively lower. Note that, the simulation domain is 8 times larger than that of the 32 thousand atom case, whereas the number of nodes used is only doubled (from 32 nodes to 64 nodes, and 1,024 nodes to 2,048 nodes). Therefore the ratio of the ghost region to the actual simulation domain is lower in this case, resulting in reduced, but still significant, performance gains.

3.1.3 Number of MPI Ranks vs. Threads per Rank

Based on Figures 1 and 2, one would expect the best productivity to be achieved using a single MPI process per node and 64 OpenMP threads per process. However, in our tests we observed that the productivity initially increases with the number of threads, but starts decreasing after 8 or 16 threads. A detailed examination of performance with respect to the number of OpenMP threads for the PETN benchmark with 2.1 million particles is shown in Figure 6. Using 4 MPI processes with 16 threads or 8 MPI processes

with 8 OpenMP threads per node offers the best performance in the range of 512 to 8,192 nodes for this system. The improved performance between 2 and 8 threads as the number of nodes increases is a result of the 4x fewer spatial decomposition domains (and MPI processes) and decreased volume of MPI communication to keep all domains synced at each step in the simulation. In general, we observed that the productivity gains from using hybrid parallelism is more pronounced especially on larger node counts.

We believe that the main reason for the poor thread scalability of our approach is the increased memory traffic and cache contention when using a large number of threads. Note that in the hybrid parallel version, we are partitioning atoms to threads using dynamic scheduling for load balancing purposes. This scheme does not necessarily respect data locality. Therefore, each thread needs to update the force on potentially a large number of neighbor atoms in its private force array that is used to prevent race conditions. When using a large number of threads, there are relatively fewer MPI ranks, and there is a higher number of atoms per such MPI domain. This situation further increases the memory traffic and the pressure on the limited cache space. To take full advantage of the multi-core and many-core parallelism on current and future hardware, our future efforts will focus on improving data locality of workloads across threads. In this regard, the spatial partitioning of the process domain to threads in a load-balanced way, for example by using the nucleation growth algorithm presented by Kunaseth *et al.* [27], is a viable route forward. Also note that unlike classical MD methods where non-bonded computations are the dominating factor for performance, ReaxFF contains a number of computationally expensive kernels such as bond interactions, dynamic 3-body and 4-body lists and hydrogen bonds. To expose a high degree of parallelism and improve thread scalability, we will explore the use of separate teams of threads that asynchronously progress through these key phases of the ReaxFF calculation.

3.1.4 Performance and Scaling with Molecular Species Analysis

Computational expense of the real-time molecular species analysis is illustrated in Figure 7. In this benchmark test of a PETN crystal with 9 million atoms, bond order values between pairs of atoms are stored at every 10 MD steps. Then on the 1000th step, stored bond order values are averaged, molecules and molecular species are determined, and a concise output is written. On 4,096 BG/Q nodes, a simulation including the real-time molecular species analysis is approximately 7-19% slower than the simulation without species analysis, depending on the number of MPI processes and OpenMP threads. On a more general note, a 5-25% reduction in productivity is typically observed with the real-time molecular species analysis on other node counts and machines. In comparison, post-processing a large volume of trajectory files with a serial analysis code can be orders of magnitude more costly and time-consuming after accounting for precious simulation time spent writing large trajectory files at high frequency. So overall, the molecular species analysis tool introduced in this study is expected to yield further productivity gains for the users of the LAMMPS/ReaxC-OMP package.

3.2 Validation through Science Applications

The developments in the LAMMPS/ReaxC-OMP package have been performed in close collaboration with domain scientists at Sandia and Argonne National Laboratories. The two science cases used for validation included the study of the effects of material defects

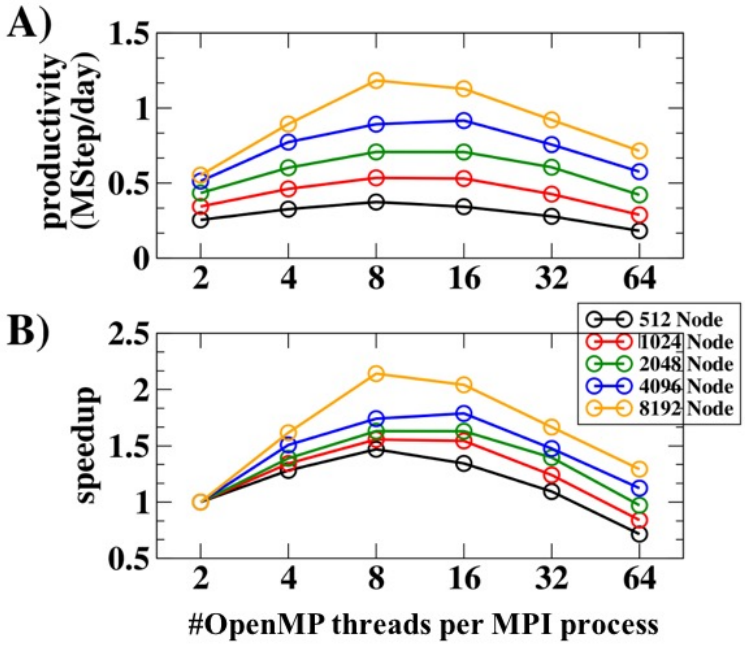

Fig. 6. Productivity (A) and parallel speedup (B) of the hybrid LAMMPS/ReaxC as a function of OpenMP threads per MPI process on 512 to 8,192 BG/Q nodes for a PETN system with 2.1 million particles. In all cases, the number of MPI processes per node times the number of threads per MPI process is 64 which is the total thread count on a BG/Q node.

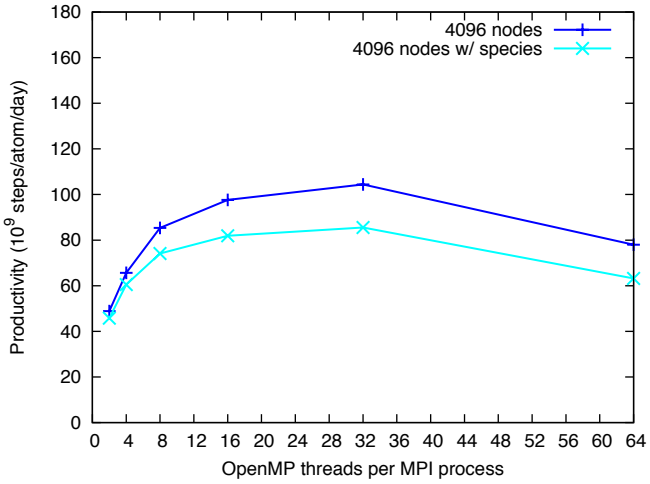

Fig. 7. Performance of real-time molecular species analysis as a function of MPI processes per node and OpenMP threads per MPI process on 4,096 BG/Q nodes for a PETN crystal containing 9 million atoms.

and heterogeneities in energetic materials, and investigation of graphene superlubricity. For both cases, energies and forces computed at each step, as well as the overall progression of molecular trajectories have been validated against the original MPI-only implementation in LAMMPS. Further validations by comparisons to experimental results could be performed thanks to the enhanced simulation time-ranges made possible by the new LAMMPS/ReaxC-OMP package. Below we give a brief summary of both studies, and provide references for further information.

3.2.1 Energetic Materials

Material defects and heterogeneities in energetic materials play key roles in the onset of shock-induced chemical reactions and the ignition of hotspots by lowering initiation thresholds [28], [29], [30], [31], [32]. These defects and heterogeneities span a range of length scales including dislocations, shear bands, grain boundaries, and porosities/cavities/voids. A hot spot with increased temperature/stress and enhanced chemical reactivity was previously observed in a micron-scale, 9-million-atom PETN single crystal containing a 20 nm cylindrical void [33]. In that study, the PETN crystal was shocked with a mild impact velocity (1.25 km/s) and the hot spot was formed after 50 ps due to the collapse of the cylindrical void. To enable the study of larger simulations required to model PETN hot spots under more realistic situations, Shan *et al.* used the hybrid parallel LAMMPS/ReaxC-OMP code presented in this paper. This study used 8,192 IBM BlueGene/Q nodes on the supercomputer Mira at Argonne National Laboratory for a total of approximately 400 hours (approximately 50 million core-hours). As a result of the performance and scalability improvements in the new package, the investigation by Shan et al. were able to continue the PETN simulation for another 450 ps, which was sufficiently long enough to estimate the hot spot growth rate and elucidate its mechanism. Temperature plots of the shock-induced void collapse and hot spot growth processes are illustrated in Figure 8. A manuscript discussing the results and findings from this work is under preparation for publication.

3.2.2 Graphene Superlubricity

In a separate study, ReaxFF simulations were used to assist with obtaining atomistic-level insight into the mechanism for macroscale superlubricity enabled by graphene nanoscroll formation [34]. With the efficient hybrid parallel ReaxFF implementation, exploration of large-scale systems under conditions of ambient humidity were enabled on leadership class computing resources, *i.e.* the Mira supercomputer. These ReaxFF simulations helped to shed light on and attribute superlubricity to a significant reduction in the interfacial contact area due to the scrolling of nanoscale graphene patches and incommensurability between graphene scroll and diamond-like carbon [34].

4 Related Work

The first implementation of ReaxFF is due to van Duin et al. [35]. After the utility of the force field was established in the context of various applications, this serial implementation was integrated into LAMMPS by Thompson et al. as the REAX package [36]. Nomura *et al.* have reported on the first parallel implementation of ReaxFF [37], but this codebase remains private to date.

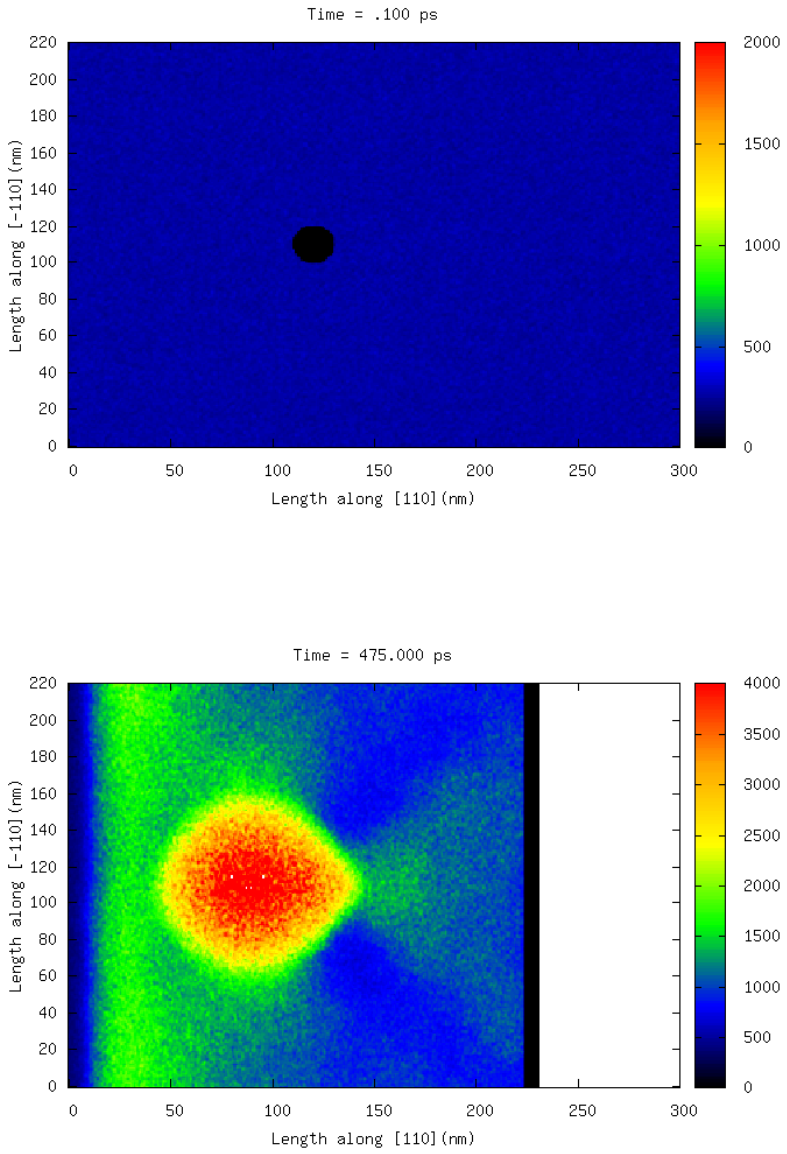

Fig. 8. Temperature maps of a PETN single crystal containing a 20 nm cylindrical void shock at 1.25 km/s impact velocity, at 0 ps (top) and 475 ps after shock (bottom). 20 nm void turned into a hot spot with an exponential growth rate.

The widely used LAMMPS/USER-REAXC package and the new LAMMPS/REAXC-OMP package described here are based on the PUREMD code developed by Aktulga *et al.* The PUREMD codebase [12] contains 3 different packages to ensure architecture portability: SPUREMD [11], PUREMD [10] and PUREMD-GPU [38]. SPUREMD, a serial implementation of ReaxFF, introduced novel algorithms and numerical techniques to achieve high performance, and a dynamic memory management scheme to minimize

its memory footprint. Today, sPuReMD is being used as the ReaxFF backend in force field optimization calculations [17] where fast serial computations of small molecular systems are crucial for extending the applicability of the Reax force field to new chemical systems. PuReMD is an MPI-based parallel implementation of ReaxFF, and exhibits excellent scalability. It has been shown to achieve up to 5× speedup over the LAMMPS/Reax on identical machine configurations. PuReMD code has been integrated into LAMMPS as the USER/ReaxC package [13], which actually constitutes the MPI-only version used for comparisons in this study.

Acceleration of ReaxFF simulations through the use of GPUs have also been explored recently. Zheng *et al.* also report a single GPU implementation of ReaxFF, called GMD-Reax [39]. PuReMD-GPU, a GP-GPU implementation of ReaxFF, achieves a 16× speedup on an Nvidia Tesla C2075 GPU over a single processing core (an Intel Xeon E5606 core). To the best of our knowledge, PuReMD-GPU is still the only publicly available GPU implementation of ReaxFF.

There has been a number of other efforts within LAMMPS to better leverage the performance capabilities of multi-core architectures. The LAMMPS OMP package uses the OpenMP interface for multi-threading, The Intel package provides improved performance on Intel CPUs and Xeon Phi accelerators [40]. The Kokkos package is a C++ library under active development to provide parallelism across different many-core architectures (including multi-core CPU, GPGPU, and Intel Xeon Phi). The force fields supported by these packages remain limited to simple force fields. None of these packages support the ReaxFF force field, which has a fairly complex formulation and therefore its hybrid parallelization has been a major challenge to this date.

5 Conclusions

We presented a hybrid MPI-OpenMP implementation of the ReaxFF method in the LAMMPS simulation software and analysis of its performance on large-scale simulations and computing resources. On Mira, a state-of-the-art multicore supercomputer, we observed significant improvements in the computational performance and parallel scalability with respect to the existing MPI-only implementation in LAMMPS. We also presented the implementation and performance results of a tool for in-situ molecular species analysis tailored for reactive simulations. While performance results obtained using a large number of OpenMP threads (e.g. 64) have come short of expectations, the threading model employed in this work serves as a useful starting point for extending the thread scalability even further (e.g. Intel Xeon Phi many-core architectures). The current hybrid implementation, however, has already been invaluable in a couple studies involving large-scale, multi-million particle simulations on leadership computing resources. It is expected that a wide community of researchers will have similar successes in their own fields of study as a result of this effort and the performance benefits will be improved further through our plans for future work.

Acknowledgements

TRS would like to acknowledge Oleg Sergeev of VNIIA for fixing several bugs in the LAMMPS' implementation of the real-time molecular species analysis. An award of computer time was provided by ALCF's Director's Discretionary program. This

research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

REFERENCES

[1] A. D. MacKerell, Jr., D. Bashford, M. Bellott, R. L. Dunbrack, Jr., J. D. Evanseck, M. J. Field, S. Fischer, J. Gao, H. Guo, S. Ha, D. Joseph-McCarthy, L. Kuchnir, K. Kuczera, F. T. K. Lau, C. Mattos, S. Michnick, T. Ngo, D. T. Nguyen, B. Prodhom, W. E. Reiher, III, B. Roux, M. Schlenkrich, J. C. Smith, R. Stote, J. Straub, M. Watanabe, J. Wiórkiewica-Kuczera, D. Yin, and M. Karplus, "All-atom empirical potential for molecular modeling and dynamics studies of proteins," *J. Phys. Chem. B*, vol. 102, no. 18, pp. 3586–3616, 1998.

[2] W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, Jr., D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell, and P. A. Kollman, "A second generation force field for the simulation of proteins, nucleic acids, and organic molecules," *J. Am. Chem. Soc.*, vol. 117, no. 19, pp. 5179–5197, 1995.

[3] W. L. Jorgensen and J. Tirado-Rives, "The opls potential functions for proteins. energy minimizations for crystals of cyclic peptides and crambin," *J. Am. Chem. Soc.*, vol. 110, no. 6, pp. 1657–1666, 1988.

[4] A. Warshel and R. Weiss, "An empirical valence bond approach for comparing reactions in solutions and in enzymes," *J. Am. Chem. Soc.*, vol. 102, no. 20, pp. 6218–6226, 1980.

[5] C. Knight and G. Voth, "The curious case of the hydrated proton," *Acc. Chem. Res.*, vol. 45, no. 1, pp. 101–109, 2012.

[6] A. C. T. van Duin, S. Dasgupta, F. Lorant, and W. A. Goddard III, "Reaxff: A reactive force field for hydrocarbons," *J. Phys. Chem. A*, vol. 105, pp. 9396–9409, 2001.

[7] T.-R. Shan, B. D. Devine, T. W. Kemper, S. B. Sinnott, and S. R. Phillpot, "Charge-optimized many-body potential for the hafnium/hafnium oxide system," *Phys. Rev. B*, vol. 81, p. 125328, 2010.

[8] T. Liang, T.-R. Shan, Y.-T. Cheng, M. Devine, B. D. Noordhoek, Y. Li, Z. Lu, S. R. Phillpot, and S. B. Sinnott, "Classical atomistic simulations of surfaces and heterogeneous interfaces with the charge-optimized many body (comb) potentials," *Mat. Sci. Eng. R*, vol. 74, pp. 235–279, 2013.

[9] S. J. Stuart, A. B. Tutein, and J. A. Harrison, "A reactive potential for hydrocarbons with intermolecular interactions," *J. Chem. Phys.*, vol. 112, no. 14, pp. 6472–6486, 2000.

[10] H. M. Aktulga, J. C. Fogarty, S. A. Pandit, and A. Y. Grama, "Parallel reactive molecular dynamics: Numerical methods and algorithmic techniques," *Parallel Comput.*, vol. 38, no. 4-5, pp. 245–259, 2012.

[11] H. M. Aktulga, S. A. Pandit, A. C. T. van Duin, and A. Y. Grama, "Reactive molecular dynamics: Numerical methods and algorithmic techniques," *SIAM J. Sci. Comput.*, vol. 34, no. 1, pp. C1–C23, 2012.

[12] A. Grama, H. M. Aktulga, and S. B. Kylasa, "PuReMD, Purdue Reactive Molecular Dynamics package." https://www.cs.purdue.edu/puremd, 2014. Accessed on June 8, 2015.

[13] H. M. Aktulga, "LAMMPS/User-ReaxC package." http://lammps.sandia.gov/doc/pair_reax_c.html, 2010. Accessed on June 8, 2015.

[14] J. D. Deetz and R. Faller, "Parallel optimization of a reactive force field for polycondensation of alkoxysilanes," *J. Phys. Chem. B*, vol. 118, no. 37, pp. 10966–10978, 2014.

[15] H. R. Larsson, A. C. T. van Duin, and B. Hartke, "Global optimization of parameters in the reactive force field reaxff for sioh," *J. Comp. Chem.*, vol. 34, pp. 2178–2189, 2013.

[16] A. Jaramillo-Botero, S. Naserifar, and W. A. Goddard, III, "General multiobjective force field optimization framework, with application to reactive force fields for silicon carbide," *J. Chem. Theory Comput.*, vol. 10, no. 4, pp. 1426–1439, 2014.

[17] M. Dittner, J. Muller, H. M. Aktulga, and B. Hartke, "Efficient global optimization of reactive force-field parameters," *J. Comput. Chem.*, p. DOI: 10.1002/jcc.23966, 2015.

[18] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *J. Comp. Phys.*, vol. 117, pp. 1–19, 1995.

[19] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl, "Gromacs 4: Algorithms for highly efficient load-balanced, and scalable molecular simulation," *J. Chem. Theory Comput.*, vol. 4, no. 3, pp. 435–447, 2008.

[20] K. J. Bowers, R. O. Dror, and D. E. Shaw, "Zonal methods for the parallel execution of range-limited n-body simulations," *J. Comp. Phys.*, vol. 221, pp. 303–329, 2007.

[21] A. Bhatelé, L. V. Kalé, and S. Kumar, "Dynamic topology aware load balancing algorithms for molecular dynamics applications," in *Proceedings of the 23rd International Conference on Supercomputing*, pp. 110–116, ACM, 2009.

[22] T. Hoefler and M. Snir, "Generic topology mapping strategies for large-scale parallel architectures," in *Proceedings of the International Conference on Supercomputing*, pp. 75–84, ACM, 2011.
[23] H. M. Aktulga, C. Yang, E. G. Ng, P. Maris, and J. P. Vary, "Topology-aware mappings for large-scale eigenvalue problems," in *Proceedings of the Euro-Par 2012 Parallel Processing Conference*, pp. 830–842, Springer, 2012.
[24] A. K. Rappe and W. A. Goddard, III, "Charge equilibration for molecular dynamics simulations," *J. Phys. Chem.*, vol. 95, no. 8, pp. 3358–3363, 1999.
[25] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, 1952.
[26] Y. Saad and M. H. Schultz, "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 3, pp. 856–869, 1986.
[27] M. Kunaseth, D. F. Richards, J. N. Glosli, R. K. Kalia, A. Nakano, and P. Vashishta, "Analysis of scalable data-privatization threading algorithms for hybrid mpi/openmp parallelization of molecular dynamics," *The Journal of Supercomputing*, vol. 66, no. 1, pp. 406–430, 2013.
[28] F. Bowden, M. Stone, and G. Tudor, "Hot Spots on Rubbing Surfaces and the Detonation of Explosives by Friction," *Proc. R. Soc. Lond. A*, vol. 188, no. 1014, p. 329, 1947.
[29] J. Johnson, P. Tang, and C. Forest, "Shock-wave initiation of heterogeneous reactive solids," *J. Appl. Phys.*, vol. 57, no. 9, pp. 4323–4334, 1985.
[30] J. Dear, J. Field, and A. Walton, "Gas-compression and jet formation in cavities collapsed by a shock-wave," *Nature*, vol. 332, no. 6164, pp. 505–508, 1988.
[31] D. Dlott and M. Fayer, "Shocked molecular-solids - vibrational up pumping, defect hot spot formation, and the onset of chemistry," *J. Chem. Phys.*, vol. 92, no. 6, pp. 3798–3812, 1990.
[32] S. Walley, J. Field, and M. Greenaway, "Crystal sensitivities of energetic materials," *Mater. Sci. Technol.*, vol. 22, pp. 402–413, APR 2006.
[33] T.-R. Shan and A. P. Thompson, "Micron-scale Reactive Atomistic Simulations of Void Collapse and Hotspot Growth in Pentaerythritol Tetranitrate," *Proc. 15th International Detonation Symposium*, vol. in press, 2014.
[34] D. Berman, S. Deshmukh, S. Sankaranarayanan, A. Erdemir, and A. Sumant, "Macroscale superlubricity enabled by graphene nanoscroll formation," *Science*, vol. 348, no. 6239, pp. 1118–1122, 2015.
[35] A. C. Van Duin, S. Dasgupta, F. Lorant, and W. A. Goddard, "Reaxff: a reactive force field for hydrocarbons," *The Journal of Physical Chemistry A*, vol. 105, no. 41, pp. 9396–9409, 2001.
[36] A. Thompson, "LAMMPS/reax package." http://lammps.sandia.gov/doc/pair_reax.html, 2009. Accessed on June 8, 2015.
[37] K. ichi Nomura, R. K. Kalia, A. Nakano, and P. Vashista, "A scalable parallel algorithm for large-scale reactive force-field molecular dynamics simulation," *Comp Phys Comm*, vol. 178, pp. 73–87, Januaury 2008.
[38] S. B. Kylasa, H. Aktulga, and A. Grama, "Puremd-gpu: A reactive molecular dynamics simulation package for gpus," *Journal of Computational Physics*, vol. 272, pp. 343–359, 2014.
[39] M. Zheng, X. Li, and L. Guo, "Algorithms of gpu-enabled reactive force field (reaxff) molecular dynamics," *Journal of Molecular Graphics and Modelling*, vol. 41, pp. 1–11, 2013.
[40] W. Brown, J.-M. Carrillo, N. Gavhanec, F. Thakkar, and S. Plimpton, "Optimizing legacy molecular dynamics software with directive-based offload," *Comp. Phys. Commun.*, vol. 195, pp. 95–101, 2015.

Hasan Metin Aktulga is an Assistant Professor in the Department of Computer Science and Engineering at Michigan State University. His research interests are in the areas of high performance computing, applications of parallel computing, big data analytics and numerical linear algebra. Dr. Aktulga received his B.S. degree from Bilkent University in 2004, M.S. and Ph.D. degrees from Purdue University in 2009 and 2010, respectively; all in Computer Science. Before joining MSU, he was a postdoctoral researcher in the Scientific Computing Group at the Lawrence Berkeley National Laboratory (LBNL).

Chris Knight is an Assistant Computational Scientist at the Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory (ANL). His research interests include development of scalable algorithms for molecular simulations to model soft condensed matter and interfaces using classical and first principle methods. He recieved the B.Sc degree in Chemistry from Eastern Michigan University in 2003 and Ph.D degree in Chemistry from The Ohio State University in 2009 under the supervision of Sherwin J. Singer. Before joining the ALCF, he was a post-doctoral researcher with Gregory A. Voth at ANL and the University of Chicago.

Paul Coffman is a Principal Scientific Applications Engineering Specialist at the Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory (ANL). Since receiving his B.Sc degree in Computer Science from the University of Minnesota - Duluth in 1993 he has had a 22-year technical career at IBM until most recently joining the ALCF. His experience spans software development, testing, performance engineering, data management and analytics across multiple architectures, languages and environments, with focus on HPC scientific applications and the IBM Blue Gene /Q system software stack for the most recent several years.

Tzu-Ray Shan is a computational materials scientist with experience in implementing software for multi-core CPU, GPU and MIC computing architectures. He earned a Ph.D. from the University of Florida in materials science and engineering and joined Sandia National Laboratories in 2011. He develops advanced force fields for molecular dynamics (MD) simulations and runs reactive MD simulations on HPC systems. He is a core developer of the LAMMPS software, primarily responsible for maintaining, developing, and implementing advanced reactive force fields. He is now part of a team adopting Kokkos in LAMMPS to achieve performance portability across different computing architectures.

Wei Jiang got Physics BS at University of Science & Technology of China, and Chemistry PhD from University of Utah. He worked as a computational postdoc fellow at Argonne national Laboratory between 2008 and 2011. Currently he is assistant computational scientist of Argonne leadership computing facility, focusing on computational chemistry simulations for soft condensed matters.