PG-PuReMD: A Parallel-GPU Reactive Molecular Dynamics Package

Sudhir B Kylasa, Hasan M Aktulga, Ananth Y Grama

Abstract—We present a parallel/GPU implementation of our open-source reactive molecular dynamics code, PG-PuReMD (Parallel GPU-Purdue Reactive Molecular Dynamics). Using a variety of innovative algorithms and optimizations, PG-PuReMD achieves over 350x speedup compared to a single CPU implementation on a cluster of 36 state of the art GPUs. This is a significant development, since it enables simulations of over 0.5M atoms in under 0.5 seconds per time-step of simulation time. We report on various design choices and implementation details of PG-PuReMD in this paper. PuReMD, on which this code is based, along with its integration into LAMMPS, is currently used by over 100 research groups worldwide and is an important community resource. PG-PuReMD is currently being independently validated at a small number of institutions and is in limited release.

Index Terms—Reactive Molecular Dynamics; Parallel GPU Implementations; Material Simulations;

1 INTRODUCTION

There has been significant effort aimed at atomistic modeling of diverse systems – ranging from materials processes to biophysical phenomena. Parallel formulations of these methods have been shown to be among the most scalable applications. Classical molecular dynamics (MD) techniques typically rely on static bonds and fixed partial charges associated with atoms. These constraints limit their applicability to non-reactive systems. ReaxFF, which is an effort aimed at addressing this limitation, is a novel reactive force field developed by van Duin et al. [1]. ReaxFF bridges quantum-scale and classical MD approaches by explicitly modeling bond activity (reactions) and charge equilibration. The flexibility and transferability of the force field allows ReaxFF to be easily extended to systems of interest. ReaxFF has been successfully applied to diverse systems [1]–[4].

Our prior work in the area led to the development of the PuReMD (Purdue Reactive Molecular Dynamics) code, along with a comprehensive evaluation of its performance. PuReMD incorporates several algorithmic and numerical innovations to address significant computational challenges posed by ReaxFF. It achieves excellent per time-step execution times, enabling nanosecond-scale simulations of large reactive systems. Using fully dynamic interaction lists that adapt to the specific needs of simulations, PuReMD achieves low memory footprint. Our tests demonstrate that PuReMD is up-to five- to six times faster than competing implementations, while using significantly lower memory. PuReMD has also been integrated with the LAMMPS (LAMMPS/User-ReaxC) software package for atomistic simulations. PuReMD and its LAMMPS version has been used by several research groups around the world on diverse systems, ranging from strain relaxation in Si–Ge nanobars [5] and water-silica systems [6] to explosives (RDX) and biomembranes (lipid bilayers). PuReMD has a large number of downloads and an active developer community.

Two important challenges in molecular simulation are the large number of time-steps required and the size of the systems that can be simulated. Time-steps in ReaxFF are of the order of tenths of femtoseconds, but several important physical analyses require simulation data spanning nanoseconds (millions of time-steps) and beyond. GPU clusters provide significant processing power in small, affordable hardware systems. These considerations provide compelling motivations for parallel/GPU acceleration of PuReMD. A production code with good scalability properties presents tremendous scientific opportunities for a large community.

The highly dynamic nature of interactions and the memory footprint, the diversity of kernels underlying non-bonded and bonded interactions, the complexity of functions describing the interactions, the charge equilibration procedure, which requires the solution of a large system of linear equations, and high numerical accuracy requirements pose significant challenges for parallel-GPU implementations of ReaxFF. Effective use of shared memory to avoid frequent global memory accesses and configurable cache to exploit spatial locality during scattered memory operations are essential to the performance of various kernels on individual GPUs. These kernels are also optimized to utilize GPUs’ capability to spawn thousands of threads, and coalesced memory operations are used to enhance performance of specific kernels. The high cost of double precision
arithmetic on conventional GPUs must be effectively amortized/masked through these optimizations. These requirements are traded-off with increased memory footprint to further enhance performance. The significant increase in performance from use of GPUs puts tremendous pressure on parallel-GPU implementations, since faster computations without commensurate reductions in communication costs result in lower efficiencies. We address these challenges through a sequence of design trade-offs of communication and redundant storage, along with alternate algorithmic choices for key kernels.

In this paper, we present in detail, the design and implementation of all phases of PG-PuReMD (Parallel GPU PuReMD). Comprehensive experiments on a state-of-the-art GPU cluster are presented to quantify accuracy as well as performance of PG-PuReMD. Our experiments show over 350x improvement in runtime on a cluster of 36 GPU-equipped nodes, compared to a highly optimized CPU-only PuReMD implementation on model systems (water). These speedups have tremendous scientific impact for diverse simulations. PG-PuReMD is the first production code of its kind. It is currently being validated, and in limited release.

The rest of the paper is organized as follows: Section 2 discusses the related work on parallel ReaxFF. Section 3 presents a brief overview of our research in this area. We briefly describe our prior research; sPuReMD - a single threaded implementation, PuReMD - multi core implementation and PuReMD-GPU - a single GPU implementation of ReaxFF. We comprehensively evaluate the performance of PG-PuReMD in Section 4. And section ?? provides the concluding remarks and future work and enhancements with the research presented in this paper.

2 RELATED EFFORTS

The first-generation ReaxFF implementation of van Duin et al. [1] strongly established the utility of the force field in the context of various applications. This serial implementation was integrated into the parallel molecular dynamics (MD) package LAMMPS [7] by Thompson et al. [8]. Except for the charge equilibration part, this integration of ReaxFF into LAMMPS was based on the original Fortran code of van Duin [1]. In [9], [10], we describe PuReMD, which features novel algorithms and numerical techniques to achieve high runtime performance, and a dynamic memory management scheme to minimize memory footprint. PuReMD exhibits excellent scalability, and has been shown to achieve up to 5x speedup over the parallel ReaxFF code in LAMMPS on identical machine configurations of hundreds of processors and beyond. In [11], we describe PuReMD-GPU, a GP-GPU implementation of ReaxFF and, achieves a speedup of $16 \times$ w.r.t single core PuReMD’s performance. To our knowledge PuReMD-GPU is the only publicly available GPU implementation of ReaxFF which has been validated by independent groups on various systems.

Zheng et al. recently reported a single GPU implementation of ReaxFF, called GMD-Reax [12]. This is the closest effort in literature to the PG-PuReMD code presented in this paper. GMD-Reax is reported to be up to 6 times faster than the User-Reax/C package in LAMMPS. However, this performance is heavily influenced by the use of single-precision arithmetic operations. Single-precision arithmetic is faster compared to double-precision arithmetic operations. Our experiments suggests that in real applications single precision arithmetic does not yield acceptable accuracy (gross energy drifts are observed at picosecond scales and beyond), and for this reason reliable implementations of ReaxFF avoids using single precision arithmetic. On a C2075 GPU, single precision arithmetic is twice as fast as double precision arithmetic because of its implementation on hardware resources which are shared by the processing cores.

3 OVERVIEW OF PuReMD

Our research in this area led to the development of PuReMD [9], [10]. sPuReMD is a serial implementation of ReaxFF and offers high modeling accuracy and linear scaling in terms of memory and run-time characteristics. It is significantly faster than comparable implementations of ReaxFF (5-6× over LAMMPS) and has been demonstrated in diverse applications contexts ranging from biomembranes (lipid layers) to explosives (PETN).

sPuReMD is the serial version of our implementation of ReaxFF, a single threaded application. Key components of a ReaxFF simulation include generating neighbors, computing energy and forces, and moving atoms under the effect of net forces until the desired number of steps is reached. Each of these components, however, is considerably more complex compared to a nonreactive classical MD implementation due to dynamic bonding and charge equilibration requirements.

Due to the truncated nature of bonded and non-bonded interactions, binning (or link-cell method) is used for neighbor generation. First, a three-dimensional grid structure is built by dividing the domain of simulation into small cells. Atoms are then binned into these cells based on their spatial coordinates. Using this technique, we can realize $O(k)$ neighbor generation complexity for each atom, $k$ is the average number of neighbors per atom. A number of parameters, such as, Cell dimensions, Atoms to cell distance, regrouping and verlet-lists can be varied to achieve tremendous performance benefits.

Major steps for computing forces in ReaxFF are determining the bond orders between atom pairs, computing forces due to those bonds, updating the partial charges on each atom and the computation of non bonded forces. Since bonded potentials depend primarily on the strength of the bonds all forces arising from the bonded interactions depends on the derivative of the bond order terms. Instead of evaluating the bond order derivative expressions at the beginning of every time
step (and using them repeatedly), they can be delayed until the end of time step. This also has the added benefit of saving frequent memory lookup at the expense of few additional book-keeping instructions. During the computation of the bonded potentials, coefficients for the corresponding bond order derivative terms arising from various interactions are accumulated into a scalar variable. After all the bonded interactions are computed, this term is used to compute the net force due to bonded interactions.

Non-bonded force computation is relatively expensive because of longer cutoff radii associated with non-bonded interactions. ReaxFF formulation of non-bonded interactions is complex compared to its classical counterparts. Using a lookup table and approximating complex expressions by means of interpolation is a very common optimization technique employed by various MD realizations of ReaxFF (offering considerable performance gains on conventional hardware architectures). sPuReMD uses a cubic spline interpolation to achieve accurate approximations of non bonded energies and forces.

(Re)assigning partial charges to atoms at each time step is the most time consuming computation of ReaxFF implementation, which does exist in classical MD techniques because they rely on static charges on atoms. sPuReMD employs krylov subspace solver with ILUT preconditioner for solving charge equilibration problem. Preconditioned GMRES offers better accuracy and less number of iterations for convergence compared to other solvers for the divergent systems studied.

In a reactive force field the dynamic nature of bonds, three-body and four-body interactions together with significant amount of book keeping necessitates large memory regions and sophisticated memory management techniques. A three phase memory management technique is effectively used by sPuReMD to estimate, (re)allocate and manage evolving data structures as the simulation progresses.

Structural comparisons between sPuReMD, GROMACS and CPMD tell us that results of sPuReMD with hexane structures are in near perfect agreement. The MACS and CPMD tell us that results of sPuReMD with bond order derivatives are computed at the end of the time-step while computing the energy and forces of bonded interactions. Lookup tables are used to aid the non-bonded interactions. Diagonally scaled conjugate gradient (PCG) is employed as the krylov subspace method for charge equilibration. Dynamic memory management techniques, similar to sPuReMD, are used by PuReMD.

PuReMD has been demonstrated to scale to more than 3K computational cores under weak-scaling scenarios, yielding over 78% efficiency. Bonded and Non-bonded interactions scale well compared to communication heavy charge equilibration. As a result of key optimizations, PuReMD achieves very good weak scaling performance.

Bonded and non-bonded interactions scale well with the increasing number of cores in the strong scaling scenarios. Because of the outer-shell choice the initialization phase at each time step consumes significant portion of the per time step time.

PuReMD achieves significant performance gains compared to LAMMPS package. PuReMD is three to five times faster than LAMMPS and with smaller memory footprint.

PuReMD-GPU [11] is our realization of ReaxFF on a single GPU. To our knowledge, this is the only publicly available GPU implementation of ReaxFF. It maintains redundant forms of data structures (neighbor lists, bond lists, EEq matrix) in order to exploit the GPUs Single Instruction Multiple Thread (SIMT) programming model. Augmented data structures help avoid (most) floating point atomic operations on GPUs yielding tremendous speedup at the expense of additional memory on the GPUs. Interestingly the use of lookup tables on the GPUs slow down the speedup by a factor of two and this can be attributed to the fact that memory operations on global memory on GPUs (200 cycles) are about 10$\times$ slower compared to local memory (20 cycles). Also since thousands of threads are executing in parallel computations are an order of magnitude faster compared to memory access.

SIMT execution model of GPU’s enables spawning thousands of thread computations simultaneously yielding significant performance gains compared to single threaded implementations. Effective modeling the data structures layout in memory, using various types of available memory on the GPU, coupled with slicing computations so that multiple threads work on the same computation simultaneously can yield an order of magnitude performance gains over conventional hardware architectures.

Depending on the amount of computation involved one can either use single thread or multiple threads per computation. Generation of neighbor lists, hydrogen bonds and non bonded interactions use multiple thread per atom kernels (and other computations use single thread per atom kernels) in PuReMD-GPU. Note that in all these computations we have relatively large computa-
tions and with suitable data layout in memory multiple threads can use coalesced read/write operations and avoid use of atomic operations yielding tremendous performance benefits. For instance, neighbor list generation of each atom may be performed by one or more threads. The case of single thread per atom is relatively straightforward. Each thread runs through all neighboring cells of the given atom and identifies neighbors which are inserted into the neighbor lists. Since shared data structures (cells and their atoms) are only read and there are no shared writes no synchronization is required. Contrast this with multiple threads per atom which requires suitable partition of the computation, as well as synchronization of writes into the neighbor list. Each thread takes as argument the atom-id for which it computes neighbors. This atom-id is used by all threads to concurrently identify neighbor cells. Each of these neighbor cells is processed by the threads in a lock-step fashion. In the case of four threads per atom, every fourth atom in a neighbor cell is tested by the same thread; i.e., thread 0 is responsible for testing whether atoms 0, 4, 8 and so on are in the neighbor list, thread 1 is responsible for atoms 1, 5, 9, .. In this scenario one may note that atoms 0, 1, 2 and 3 of the first neighbor cell will be processed concurrently by threads with ids 0, 1, 2 and 3 in the first step; atoms 4, 5, 6 and 7 are processed concurrently by threads 0, 1, 2 and 3 in the second step respectively and so on. Since the threads themselves are executed in SIMT fashion, synchronization among these threads is arbitrated using prefix sum of the generated neighbors and the final insertion into the neighbor list is performed after the relative index is determined using the prefix sum computed earlier.

For bonded interactions (lone-pair, two-body, three-body and four-body) the size of the computation is relatively small and some of these suffers from thread-divergence because of which multiple threads per computation does not yield better speedups compared to single thread per computation. Speedups from 20% to 6× can be realized using this type of implementation. Atomic operations can be avoided using shared memory to store intermediate results which can be later processed to generate the final result. Bonded and Non-bonded interactions compute forces and energies contributed by each atom using this mechanism yield tremendous performance gain compared to the use of atomic operations (upto 40× speedup).

PuReMD-GPU achieves a speedup of 16× compared to sPuReMD and 2-3× compared to 8-core PuReMD. PuReMD-GPU has been validated for accuracy by independent research groups and is publicly available.

4 Experimental Results

We report on our comprehensive evaluation of the performance of PG-PuReMD. For single core simulations using sPuReMD, we used our development testbed with the following configuration: two intel Xeon CPU E5606 processors (four cores per processor) operating at 2.13 GHz and 24 GB of memory, running Linux OS. GCC 4.6.3 compiler was to build the PuReMD executable with the following options: “-O3 -funroll-loops -fstrict-aliasing”. We extensively used Lawrence Berkeley National Laboratory’s GPU Cluster (DIRAC) for evaluating the performance of PG-PuReMD and Carver cluster for PuReMD using single- and multiple-cores. Carver is a liquid cooled IBM iDataPlex system with 1202 nodes (9984 processor cores). We used nodes with two quad-core intel Xeon X5550 (“Nehalem”) 2.67 GHz processors and 24 GB of DDR3 RAM for our testing. All nodes are interconnected by 4X QDR InfiniBand switch. PuReMD is compiled with openmpi compiler version 1.4.5 and with following options: “-O3 -funroll-loops -fstrict-aliasing”. PuReMD simulations were run on DIRAC cluster, which is a 50 GPU node cluster interconnected with Quad Data Rate (QDR) InfiniBand switch. Each GPU node contains 2 Intel 5530 2.4 GHz, 8MB cache, 5.86GT/sec QPI Quad core Nehalem processors (8 cores per node) and 24GB DDR3-1066 Reg ECC memory. Out of the 50 nodes, 44 nodes have NVIDIA Tesla C2050 Fermi GPUs with 3GB of memory (this pool was used extensively for the evaluation of PG-PuReMD). PG-PuReMD is compiled in CUDA 5.0 environment with the following compiler options “-arch=sm_20 -funroll-loops -O3” and openmpi (version 1.4.5) is used to link all the object files to produce the final executable. All the arithmetic operations are double precision. Thread block size for all the kernels in this section is 256 except matrix-vector dot product which uses a block size of 512 threads. Neighbor-list kernel uses 16 threads per atom, while hydrogen-bonds, coulombs/van der waals force and matrix-vector dot product kernels uses 32 threads per atom. The model systems in all these simulations use a time-step of 0.25 femtoseconds, a tolerance of 10−6 for the QEq solver, and Berendsen NVT ensemble.

We used water systems of various sizes for in-depth analysis of performance, since it represents diverse stress points for the code. For the weak scaling tests, we use water systems with 10,000 atoms and 16,000 atoms per GPU, and for strong scaling analysis, we use water systems with 80,000 and 200,000 atoms.

To better understand the results of our experiments, we identify six key parts of PG-PuReMD:

- **comm**: initial communications step with neighboring processors for atom migration and boundary atom information exchange.
- **nbrs**: neighbor generation step, where all atom pairs falling within the interaction cut-off distance $r_{nbrs}$ are identified.
- **init_forces**: generation of the charge equilibration (QEq) matrix, bond list, and H-bond list based on the neighbors list.
- **QEq**: is the charge equilibration part that solves a large sparse linear system using Conjugate Gradients with a diagonal preconditioner. This involves costly matrix-vector multiplications and both local
and global communications.
- **bonded**: is the part that includes computation of forces due to all interactions involving bonds (hydrogen bond interactions are included here as well). This part also includes identification of 3-body and 4-body structures in the system.
- **nonb**: is the part that computes nonbonded interactions (van der Waals and Coulomb).

Each of these parts has different characteristics: some are compute-bound, some are memory-bound while others are interprocess communication-bound. Together they comprise almost 99% of the total computation time for typical systems. We perform detailed analysis of these major components to better understand how PG-PuReMD responds to increasing system sizes and increasing number of processors. We also use these results to infer the impact of various machine parameters on performance.

In this section PuReMD-1-core refers to simulations run with PuReMD using 1-core per node and PuReMD-8-core refers to 8-cores per node.

**4.1 Strong Scaling Results**

Figures 1(a) and 1(b) presents timings of PG-PuReMD, PuReMD-1-core and PuReMD-8-core for the two water systems. For both the water systems, we notice that there is a large drop in the total time per time step for PuReMD-1-core compared to others when the number of nodes is increased and this is because only one-core of the available cores is used for computation (computation size is relatively large per MPI process). As the number of nodes increases, number of atoms processed per node decreases resulting in decrease in the time per time step. When all the available cores are utilized, with PuReMD-8-cores, we notice that the drop in the total time per time step becomes less significant as we increase the number of nodes used. This is because of the following reasons; (i) Number of atoms processed per node is significantly less compared to when one core is used (roughly 8 × less), (ii) as the number of nodes increases the number of
MPI processes increase as well contributing to increases communication costs (iii) number of atoms processed per MPI process drops down significantly from 2500 per process when 4 nodes are used to 278 when 36 nodes are used and this explains why this curve becomes flatter as the number of nodes used is increased and, (iv) the QEq computation, which is communication-bound and is executed in a lock-step fashion because of the linear solve, tends to dominate when large number of nodes are used and this explains why this curve becomes flatter as the number of nodes used is increased. The curve for PG-PuReMD behaves similar to PuReMD with 8-cores curve but with subtle differences. Number of processes spawned by PG-PuReMD is always equal to the number of nodes used and for PuReMD-8-cores implementation. Because of limited available memory on the GPUs (3GB of global memory per GPU), the water-200K system can only be run on 12 GPUs and beyond.

Figure 2(a) illustrates the timings of major components of a typical time step for water-80K system. init kernel, which generates the bond list, hydrogen-bond list and QEq matrix, sees a consistent drop as the number of GPUs are increased. This kernel uses single thread per atom implementation and with a thread block size of 256 when 4 nodes are used PG-PuReMD spawns 79 blocks which is the reason why a steep drop is noticed for this kernels’ timing when the number of GPUs are increased (number of thread blocks available for execution is significantly more than the available number of SM’s on the GPUs). As we move further along the x-axis, the curve tends to become flat and this is because with the increasing number GPUs atoms processed per node will decrease and at a certain point the number of thread blocks created is less than the number of SM’s.
on the C2070 GPUs (from 27 GPUs onwards). After this point, some of the SMs are idle on the GPU, while other SMs are busy executing the kernel (because of the SIMT execution model of CUDA). Note that the time for the init kernel still drops after this point because the size of the computation decreases with the increasing number of GPUs but this decrease is significantly less compared to the drop which is noticed when the thread blocks created were more than the available number of SMs on the GPUs.

Similar arguments can be made about the bonded interactions curve in this figure. Except hydrogen bonds interactions all the other interactions use single thread per atom kernels which explains why bonded curve displays similar behavior as init curve. Note that three-body and four-body interactions dominate the bonded interactions for the water system. Hydrogen-bonds interactions comprise about 25% of the bonded interactions in a typical time step for the water system. Hydrogen-bonds kernel uses multiple threads per atom implementation because of which this kernel exhibits a better scaling performance compared to kernels which employ single thread per atom implementation.


the neighbor list of each atom is large (order of several hundreds) and this computation can be easily adopted to use multiple threads per atom, as discussed in section 3. Compared to single threaded implementation this kernel achieves a speedup of 9×. Clearly it shows a consistent drop in the time as the GPUs are increased. When more GPUs are used (27 and beyond) this curve tends to become flatter because of the following reason: when 36 GPUs are used this kernel spawns 138 thread blocks at 16 threads per atom, which is just above the maximum number of thread blocks each GPU can execute at maximum efficiency. With an efficiency of 33%, ideally each SM can execute a maximum of 3 thread blocks compared to several tens of thread blocks when smaller number of nodes are used for simulation.


Comparing figures 2(a) and 2(b) following observations can be made; (i) nonb computation achieves excellent strong scaling results when GPUs are used as opposed to CPUs (it achieves a speedup of over 40 × in PuReMD-GPU) and this is the most expensive computation in PuReMD, (ii) init tends to be the most expensive computation on GPUs because of the size of the computation done per thread and single thread per atom implementation (iii) QEq starts to dominate with increasing number of nodes on the GPUs because the computation is relatively faster on the GPUs as opposed to CPUs without a commensurate decrease in the communication costs; and (iv) exchange of boundary atoms dominate when CPUs are used where it stays relatively constant on the GPUs (PuReMD spawns 288 MPI processes on a 36-node simulation as opposed to 36 with PG-PuReMD).
<table>
<thead>
<tr>
<th>GPUs</th>
<th>QEq_TotalTime %</th>
<th>Efficiency</th>
<th>QEq_TotalTime %</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11.80</td>
<td>100.00</td>
<td>10.99</td>
<td>100.00</td>
</tr>
<tr>
<td>2</td>
<td>13.11</td>
<td>95.97</td>
<td>11.66</td>
<td>99.17</td>
</tr>
<tr>
<td>4</td>
<td>14.61</td>
<td>93.46</td>
<td>12.33</td>
<td>97.10</td>
</tr>
<tr>
<td>8</td>
<td>15.45</td>
<td>91.52</td>
<td>12.93</td>
<td>96.33</td>
</tr>
<tr>
<td>12</td>
<td>16.10</td>
<td>91.21</td>
<td>13.61</td>
<td>95.27</td>
</tr>
<tr>
<td>18</td>
<td>16.47</td>
<td>90.17</td>
<td>13.70</td>
<td>94.74</td>
</tr>
<tr>
<td>27</td>
<td>20.53</td>
<td>84.68</td>
<td>17.78</td>
<td>89.92</td>
</tr>
<tr>
<td>36</td>
<td>21.04</td>
<td>84.02</td>
<td>18.25</td>
<td>89.04</td>
</tr>
</tbody>
</table>

**TABLE 1**

Efficiency results of Water system for weak scaling simulations

### 4.2 Weak Scaling Results

We now discuss the weak scaling results of PG-PuReMD and compare it with sPuReMD, and PuReMD. We used water systems with 10K and 16K atoms per GPU for benchmarking PG-PuReMD application. Figures 3(a) and 3(b) illustrates the speedup and total time per time step for the two water systems respectively. For water system with 16K atoms per GPU, PG-PuReMD achieve a speedup of 359× on 36 GPUs when compared to sPuReMD, 11× when compared to PuReMD-1-core and 1.6× when compared to PuReMD-8-cores. Note that performance of the GPUs is significantly dependent on the available global memory. With 3GB of global memory, we can only run a maximum of 16K atoms per node and with 6GB of global memory PuReMD-GPU can accommodate water system of size 50K atoms. Outer shell maintained by PG-PuReMD, which enables parallelization of ReaxFF, each process needs to maintain significant portion of boundary atoms, which are not present in PuReMD-GPU. These contribute to decreased number of atoms in each GPUs sub-domain can accommodate as well as achievable speedup per process (PuReMD-GPU achieves a speedup of 13× with 10K atom water system as opposed to 16× with 50K water system).

Figures 4(a) and 4(b) presents the timings of individual components of ReaxFF under weak scaling scenarios for PG-PuReMD and PuReMD-8-core for 16K atoms per node. All the components, except comm and qeq are constant because of constant sub-domain size per node. comm, which measure the exchange of boundary atoms, increases with increasing number of nodes. Note that comm and qeq increases relatively less in PG-PuReMD compared to PuReMD-8-core because of number of MPI processes used in these two implementations and, QEq, which is a linear solve, is executed in a lock step fashion and during each iteration four communications are involved which act as barriers for the MPI processes.

Table 1 presents the efficiency results of PG-PuReMD for water systems under weak scaling scenarios. PG-PuReMD achieves a speedup of about 10× per MPI process when compared to PuReMD single core implementation. This results in decreased parallel efficiency as we increase the number of GPUs as shown in the above table. PuReMD [10] achieves an efficiency of 99% with 32 MPI processes, while PG-PuReMD achieves 84% efficiency for comparable number (36) of MPI processes. This decrease in efficiency for PG-PuReMD is a consequence of the faster execution by the GPUs.

### 5 Conclusion

In this paper, we presented an efficient and scalable parallel implementation of ReaxFF using MPI on CUDA platforms. Our open-source implementation is shown to achieve a 350x speed up compared to single CPU implementation under weak-scaling scenarios. PG-PuReMD’s accuracy has been verified against the benchmark production PuReMD code by comparing various energy and force terms for large numbers of time-steps under diverse application scenarios and systems.

**ACKNOWLEDGMENTS**

We thank Adri van-Duin for significant help in validating our software on a variety of systems. We also thank Joe Fogarty at the University of Southern Florida for constructing model systems for testing and validation and Lawrence Berkeley National Laboratories for providing us access to their GPU cluster for benchmarking our application.

**REFERENCES**