Final Report

DoE Award Number: ER25789

Name of Recipient: Purdue University

Project Title: Hierarchical Petascale Simulation Framework for Stress Corrosion Cracking

1 Project Accomplishments

A number of major accomplishments resulted from the project. These include:

- Data Structures, Algorithms, and Numerical Methods for Reactive Molecular Dynamics. We have developed a range of novel data structures, algorithms, and solvers (amortized ILU, Spike) for use with ReaxFF and charge equilibration.
- Parallel Formulations of Reactive MD (Purdue Reactive Molecular Dynamics Package, PuReMD, PuReMD-GPU, and PG-PuReMD) for Messaging, GPU, and GPU Cluster Platforms. We have developed efficient serial, parallel (MPI), GPU (Cuda), and GPU Cluster (MPI/Cuda) implementations. Our implementations have been demonstrated to be significantly better than the state of the art, both in terms of performance and scalability.
- Comprehensive Validation in the Context of Diverse Applications. We have demonstrated the use of our software in diverse systems, including silica-water, silicon-germanium nanorods, and as part of other projects, extended it to applications ranging from explosives (RDX) to lipid bilayers (biomembranes under oxidative stress).
- Open Source Software Packages for Reactive Molecular Dynamics. All versions of our software have been released over the public domain. There are over 100 major research groups worldwide using our software.
- Implementation into the Department of Energy LAMMPS Software Package. We have also integrated our software into the Department of Energy LAMMPS software package.

In the rest of this report, we elaborate on these accomplishments.

2 Performance and Scalability of Purdue Reactive Molecular Dynamics (PuReMD) Package

In our previous reports, we have demonstrated the superlative performance of our serial and parallel PuReMD releases compared to other implementations. Here, we summarize the performance of our most recent releases – PuReMD GPU for Nvidia GPUs and PG-PuReMD for GPU clusters. For a detailed discussion of the serial and parallel (MPI) formulations, we kindly refer the reader to [3, 4].

System size	Look-up tables	W/o look-up tables
6540	7.13	13.6
10008	7.61	13.04
25050	9.06	16.02
36045	9.16	16.36
50097	8.15	16.55

Table 1: Speedup achieved for various sizes of water systems

2.1 PuReMD-GPU: Experimental Results

Our detailed benchmarking of PuReMD-GPU demonstrates a speedup of 16x for a model water system, leveraging various algorithms and optimizations [2].

Here, we summarize our evaluation of the performance, stability, and accuracy of PuReMD-GPU. All simulations are performed on an Intel Xenon CPU E5606 operating at 2.13 GHz processor, 24GB of RAM with Linux OS and equipped with Tesla C2075 GPU. The PuReMD code is compiled with the following compiler options - "-funroll-loops -fstrict-aliasing -O3". PuReMD-GPU is compiled in CUDA 5.0 environment with the following compiler options "-arch=sm_20 -funroll-loops -O3". *All arithmetic operations are double precision.* Fused Multiplication Addition (*fmad*) operations are not used in the PuReMD-GPU implementation so that results perfectly match those from PuReMD execution. All simulations use a block size of 256 threads. Some kernels are designed to use multiple threads per atom, and the block size for these kernels is mentioned where necessary. The model systems in all these simulations use a time-step of 0.1 *femtoseconds*, a tolerance of 10^{-10} for the QEq solver, and NVE ensemble unless otherwise noted.

We use water and silica model systems for in-depth analysis of accuracy and performance, since they represent diverse stress points for the code. Water systems of sizes 6,540 atoms, 10,008 atoms, 25,050 atoms, 36,045 atoms and 50,097 atoms are used. Silica systems of sizes 6,000 atoms, 12,000 atoms, 24,000 atoms, 36,000 atoms and 48,000 atoms are used.

2.1.1 Scaling Studies of Performance

Water systems with different sizes are used to test scaling properties of PuReMD-GPU. Table 1 shows the speedup achieved for water systems of various sizes. We clearly observe that as the system size increases, the effective speedup also increases. For the water system we achieve a speedup of 16x compared to the PuReMD implementation. *This is the key result of our development effort.* In all these simulations, *neighbor-list, hydrogen-bond , matrix-vector, and coulombs/van der waals force* kernels use multiple threads per atom, and all the other kernels use one thread per atom.

2.1.2 Performance Across Diverse Systems

Apart from the water system, we also used silica systems of different sizes to further validate the performance of PuReMD-GPU. From Table 2, we observe that PuReMD-GPU yields a similar performance improvement as observed with water systems, compared to baseline PuReMD implementation. Speedup achieved with silica systems is lower compared to water systems. In the silica system, there are no hydrogen bonds. *Four-body* interactions dominate in these systems, as every atom participates in four-body interactions. Even with the use of auxiliary memory in the bond-list, atomic operations cannot be avoided in this kernel. This is because of the following reason: assuming that *atoms (i, j, k, l)* participate in a four-body interaction, we store *three-body*

System size	look-up tables	W/o look-up tables
6000	5.8	9.64
12000	5.951	10.13
24000	6.5	10.86
36000	6.6	11.39
48000	6.6	11.45

Table 2: Speedup achieved for various sizes of silica systems

	water-6540		silica-6000	
Data-structure	PuReMD	PuReMD-GPU	PuReMD	PuReMD-GPU
Neighbor-list	75.1	150.09	43.32	86.54
Hydrogen-bond list	7.12	32.8	4.5	13.73
QEq Matrix	25	54.4	14	32.77
Bond-list	27.7	42.6	26.03	40.11
Three-body-list	9.7	9.7	27	27
Total Memory	144.62	289.59	114.85	200.15

Table 3: Memory footprint for water-6540 system. (All the memory footprints are at system initialization. They may grow as the simulation progresses.)

interactions among *atoms (i, j, k) and atoms (j, k, l)* in the *three-body-list* and atoms *j and k* are central atoms. The thread processing the four-body interactions for *atom j*, needs to update the force on *atom l* during the processing of this interaction. We use an atomic operation in this case to avoid maintaining another data structure for resolving this dependency. Avoiding this atomic operation may improve the performance of this kernel for systems where four-body interactions are dominant, at the expense of additional memory usage. However, this significantly increases complexity of the kernel (overheads such as use of additional local variables and increased use of shared memory needs to taken into account).

2.1.3 Performance Analysis of PuReMD-GPU

We now present a detailed analysis of our PuReMD-GPU implementation.

Memory Footprint of PuReMD and PuReMD-GPU Table 3 presents memory usage of various data structures used in PuReMD and PuReMD-GPU. *Neighbor-list* memory usage in PuReMD-GPU is doubled, compared to PuReMD. This is because PuReMD-GPU maintains redundant neighbors in this list. *Bond-list* and *hydrogen-bond list* data structures (in PuReMD-GPU) are augmented with additional memory to store temporary results during various computations to help resolve dependencies. This explains the increased memory usage by these lists compared to PuReMD. Separate kernels process this auxiliary memory, utilizing coalesced global memory access and shared memory resulting in better performance. QEq matrix storage is doubled in PuReMD-GPU because both upper and lower triangular parts of the symmetric matrix are stored; this is not the case in PuReMD. Three-body list storage is identical in PuReMD and PuReMD-GPU. Overall memory footprint of PuReMD-GPU is roughly double compared to PuReMD. However, this increase in memory usage allows us to achieve significant speedups on the GPU.

Computation	Water-6540			Silica-6000		
	CPU Time	GPU Time	GPU Overhead	CPU Time	GPU Time	GPU Overhead
Neighbor list generation	117	14	-	85.3	11.9	-
Initialization	232.8	31.1	12	163	10.8	0.57
Bond Order	4.1	0.5	-	10.9	0.7	-
Bond Energy	2.4	0.2	0.03	4.5	0.313	0.03
Lone Pair Interactions	2.7	0.8	0.1	4.8	1.25	0.09
Three Body Interactions	37.4	5.9	0.4	113.5	13.2	0.56
Four Body Interactions	34.7	8.1	0.5	252.8	58.9	0.89
Hydrogen Bonds	133.1	6.2	4.9	0.05	0.9	0.5
Charge Equilibration	589.9	43.0	-	97.5	21.9	-
Couloumbs Forces	1001.9	24.2	0.1	587.2	14.9	0.1

Table 4: Timings for various computational components of a single time-step for the water-6540 system and silica-6000 systems. All the timings are in milliseconds.

Detailed Performance Analysis Table 4 presents a comparison of PuReMD and PuReMD-GPU implementations and overheads associated with various computations in the GPU implementation. Overheads are incurred in PuReMD-GPU because it uses additional kernels to perform computations that are not present in PuReMD. It is evident from this table that the overhead incurred by most kernels is much less than the cost of (essential) computation in the kernel, thus resulting in good speedups. An exception to this is the hydrogen bond kernel, which takes 6.2 msecs and has an overhead of 4.9 msecs. This overhead is because of processing the *hydrogen-bonds list*, which is large compared to the *bond-list*.

Neighbor-list generation does not have any overhead when compared to PuReMD implementation, since it is implemented in a single kernel. The *initialization* step, during which *bond-list, QEq matrix and hydrogen-bond-list* are generated, uses separate post processing kernels to make the bond-list and hydrogen-bond list symmetric. *Bond-list* is made symmetric in PuReMD-GPU in two stages. *Bond-list* is generated during the *initialization* step and a separate kernel iterates over the *bond-list* and updates pointers so that *bond (i, j)* and *bond (j, i)* point to each other. A similar approach is taken for *hydrogen bonds* list construction as well.

Bond-energy interaction includes overhead of separate kernels that compute E_{be} of the system. *Lone-pair* interaction has the overhead of three kernels, which compute E_{lp}, E_{ov}, and E_{un}. A post-processing step is used to compute the force on each atom due to this interaction. *Three-body* interaction has the overhead of kernels that perform reduction on lists to compute E_{ang}, E_{pen}, and E_{coa}. Another post-processing kernel is used to compute the force for each atom due to this interaction and intermediate derivatives. *Four-body* interaction has the overhead of the kernels that perform reduction for computing E_{tor} and E_{con}. A post-processing step is also used for computing the force on each atom due to this interaction and intermediate derivatives. *Hydrogen-bond* interaction has the overhead of the kernel for computing E_{hb}. Two separate kernels are used for post-processing, one to iterate over the *hydrogen-bond-list* and one iterating over *bond-list* to compute the force on each atom due to this interaction.

Of all the bonded interactions, *hydrogen-bond* interaction is the most expensive, followed by *three-body* and *four-body* interactions for the water system. *Four-body* interactions dominate among the bonded interactions in silica systems. *Bond-order, bond-energy and lone-pair* interactions are simpler kernels, costing only a fraction of the total time spent in the bonded interaction compu-

	Kernel	1/ atom	2 / atom	4 / atom	8 / atom	16 / atom	32 / atom
Water 6540 At.	Neighbor-generation	23.1	16.2	13.9	13.7	14.7	22.2
	Three-body	6.1	6.4	5.7	6.3	9.3	13.9
	Four-body	8.1	11.2	11.7	13.3	17.8	24.9
	Hydrogen-bonds	7.6	6.6	6.1	5.7	5.9	6.2
	GMRES	193.0	100.0	74.5	58.9	53.9	54.7
	Coulombs/ VDW	30.4	28.5	26.8	24.5	23.9	24.3
Water 36045 At.	Neighbor-generation	127.2	85.6	73.2	73.7	80.1	120.5
	Three-body	23.0	25.3	25.5	31.9	47.3	71.9
	Four-body	39.9	55.7	59.7	69.7	98.4	138.9
	Hydrogen-bonds	38.0	32.8	31.1	30.4	31.6	33.1
	GMRES	1132.9	562.8	329.5	220.0	183.8	178.5
	Coulombs/VDW	162.4	148.6	136.9	132.1	128.7	132.5

Table 5: Effect of multiple threads per atom on the kernel performance. All the timings are in msecs

Computation	atomic	add. memory
Bond Order	0.5	0.5
Bond Energy	7.6	0.2
Lone Pair Interactions	122.1	0.8
Three Body Interactions	265.3	6.0
Four Body Interactions	55.9	8.1
Hydrogen Bonds	157.5	6.2
Coulombs/van der Waals	24.2	24.2

Table 6: Timings for each interaction when using atomic operations vs additional memory to resolve dependencies for the water-6540 system. All the timings are in msecs.

tations. *Three-body* and *four-body* interactions are implemented using multiple threads per atom. Table 5 presents the associated timings. Multiple threads per atom kernels for *three-body and four-body* interactions are not significantly better than single thread per atom implementations, because of the fact that *bond-lists* of each atom have only a few entries, and if multiple threads per atom are used, a large number of thread-blocks are created, which affects the performance of these kernels because of low kernel occupancy.

For non-bonded interactions, Coulombs and Van Der Waals forces are computed by iterating over the *neighbor-list* of each atom. Table 6 presents the timings of each of these kernels using multiple threads per atom. We clearly observe from this table that using multiple threads per atom yields better performance because of coalesced memory operations and effective use of shared memory to process intermediate results.

The sparse matrix-vector kernel is implemented using a 2D partitioning of the QEq matrix. Table 5 presents the impact of varying number of threads per atom for this kernel.

Atomic operations vs. additional memory to resolve dependencies Table 6 presents timings for each of the bonded and non-bonded interaction kernels when atomic operations are used to update global memory locations (updating the energies of the system). We observe from this table that the performance of all of the kernels is significantly impacted by the use of atomic operations to update the global memory.

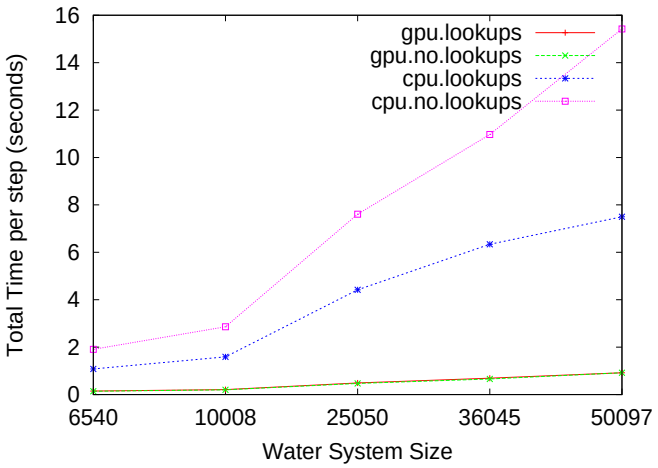

Figure 1: Time per time-step for the water systems.

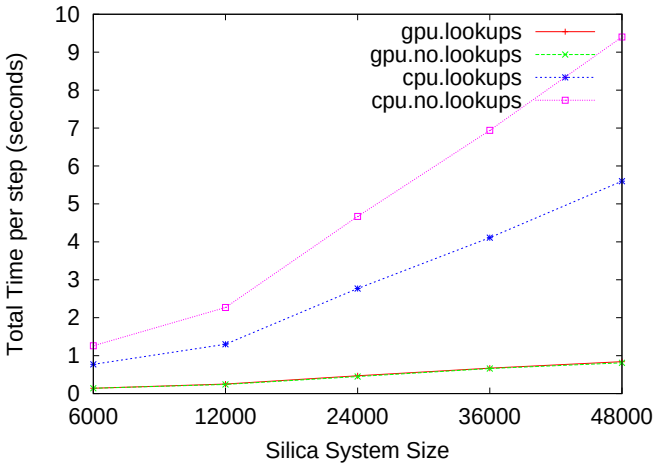

Figure 2: Time per time-step for silica systems.

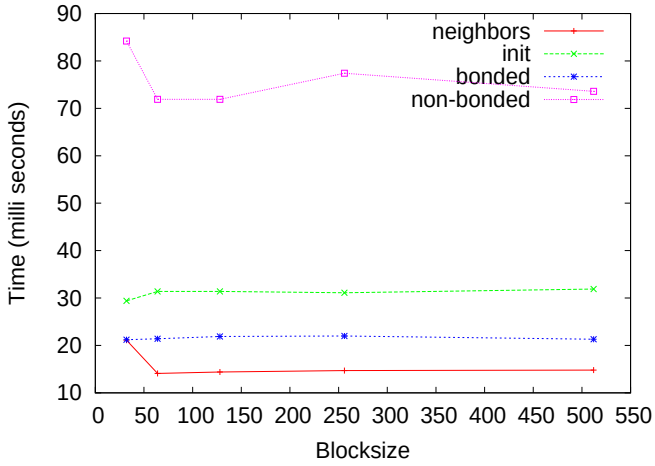

Figure 3: Effect of block size on the performance of kernels.

Using Lookup Tables. Figures 1 and 2 show the performance of PuReMD and PuReMD-GPU with and without the use of lookup tables for various sizes of water and silica model systems. We observe from these figures that PuReMD's performance is significantly impacted if lookup tables are not used, whereas PuReMD-GPU's performance remains inert to this change. Since GPU's run kernels in *SIMT* fashion, it represents a few instruction overhead compared to serial execution of these instructions on the CPU, which consume a large number of instruction/memory cycles. This explains why lookup tables yield significant performance benefits on CPU's and not so on GPUs.

Block Size and System Performance GPU kernel performance is a function of block size, as well as resources (shared memory and registers) available at each *SM*. The Tesla C2075 GPU has a 32K register file and 48KB of shared memory per block. To hide memory access latency, CUDA switches between blocks, which means that a single multiprocessor can run up-to 8 blocks of threads (this may vary depending on the GPU used), provided it has the resources it needs. Each resident or active block is assigned its own set of resources, facilitating fast switch among the resident blocks on the multiprocessor. Each block is bound to the multiprocessor on which it is scheduled to execute until its completion. Recall that the occupancy of a CUDA kernel is defined as the ratio of active warps to maximum number of active warps. Occupancy is limited by the resource constraints on the SM (shared memory usage per block, number of registers used per block, and block-size). Kernels with low occupancy have a harder time hiding latency. Consequently, it is desirable to have high occupancy in order to achieve better performance, especially for memory-bound applications.

Kernels with fewer threads in a block may have high occupancy, but the performance degrades because the SM cannot schedule enough instructions to hide latency (only 8 blocks can be scheduled at any point of time). As a result, the SM waits on the instructions to complete before scheduling the rest of the blocks. Larger number of threads limit the occupancy because of resource constraints. This decreases the performance of the kernel. Figure 3 shows the timings of each of the kernels with different block sizes. We observe that the performance of kernels varies with block size, which can be directly attributed to the available resources to that specific kernel. Each kernel uses different amounts of resources and is tuned to yield peak performance at a certain thread block size.

	After 10K steps		After 100K steps	
Energy	CPU-GPU(%) for water	CPU-GPU(%) for silica	CPU-GPU (%) for water	CPU-GPU(%) for silica
E_{be}	-3.904e^{-02} (6.836e^{-06}%)	-1.289e^{-00} (1.319e^{-04}%)	1.454e^{02} (-2.547e^{-02}%)	1.824e^{02} (-1.866e^{-02}%)
$E_{ov} + E_{un}$	-2.901e^{-03} (1.214e^{-05}%)	-5.645e^{-02} (7.307e^{-05}%)	9.831e^{01} (-5.706e^{-01}%)	-9.744e^{01} (1.281e^{-01}%)
E_{lp}	2.600e^{-09} (4.277e^{-03}%)	5.602e^{-06} (-4.599e^{-01}%)	-6.540e^{-08} (4.838%)	3.705e^{-05} (-2.298%)
$E_{ang} + E_{con}$	-2.073e^{-02} (-2.512e^{-04}%)	-1.348e^{-01} (-2.581e^{-04}%)	-2.430e^{01} (-2.949e^{-01}%)	-3.137e^{01} (-6.113e^{-02}%)
E_{hb}	4.985e^{-03} (-5.601e^{-05}%)	0 (0%)	1.616e^{02} (-1.816%)	0(0%)
$E_{Tor} + E_{conj}$	-1.396e^{-04} (-7.211e^{-04}%)	7.294e^{-02} (3.680e^{-03}%)	-1.694e^{-01} (-8.562e^{-01}%)	5.825e^{0} (2.939e^{-01}%)
E_{vdw}	2.077e^{-02} (1.779e^{-05}%)	-1.461 (-4.124e^{-04}%)	3.029e^{02} (2.600e^{-01}%)	2.357e^{01} (6.641e^{-03}%)
E_{coul}	-2.348e^{-01} (8.731e^{-05}%)	-1.385 (2.647e^{-04}%)	4.538e^{02} (-1.694e^{-01}%)	-8.138e^{01} (1.550e^{-02}%)
E_{pol}	-2.013e^{-01} (-1.114e^{-04}%)	-1.187 (-3.430e^{-04}%)	3.835e^{02} (2.133e^{-01}%)	-7.095e^{01} (-2.041e^{-02}%)

Table 7: Differences in various energies between PuReMD and PuReMD-GPU after 100,000 step simulation on water-6540 and silica-6000 systems. At the beginning of the simulation the energies between CPU and GPU match perfectly to machine precision

2.1.4 Accuracy of PuReMD-GPU

We perform extensive validation tests of PuReMD-GPU's accuracy compared to PuReMD. Table 7 presents the deviation in various energies between PuReMD and PuReMD-GPU after the 0^{th} step, 10,000 steps, and 100,000 steps. After the 0^{th} step, various energies are identical between PuReMD and PuReMD-GPU. After 10,000 steps, the deviation in various energies is negligible. After 100,000 steps, the maximum deviation observed is about 4.8%. This can be attributed to the fact that arithmetic operations on the GPU are not performed in the same order as the CPU.

2.2 Experimental Results: GPU Cluster Formulation

We report here on our comprehensive evaluation of the performance of PG-PuReMD. All the simulations are performed on Lawrence Berkeley National Laboratory's GPU Cluster (DIRAC). This is a 50 GPU node cluster inter-connected with Quad Data Rate (QDR) InfinitiBand 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 MPI is used to link all the object files to produce the final executable. All the arithmetic operations are double precision. Fused Multiplication Addition (*fmad*) operations are not used in the PG-PuReMD implementation so that the results perfectly match the serial implementation. 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 NVE 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 per GPU and 16,000 atoms per GPU, and for strong scaling analysis, we use water systems with 80,000 and 200,000 atoms.

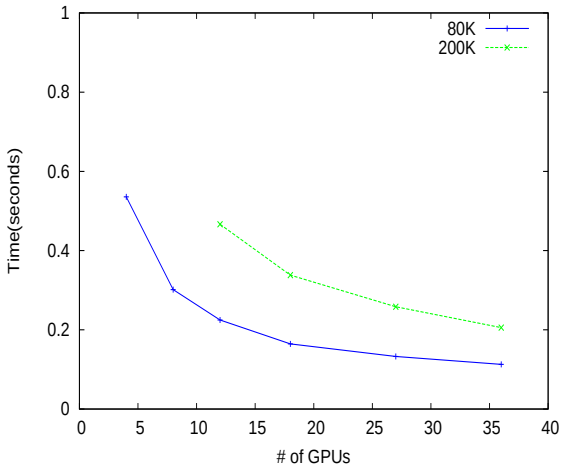

Figure 4: Strong scaling results for Water systems (Water-80K and Water-200K).

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 analyses of these major components to better understand how 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.

2.2.1 Strong Scaling Results

Figure 4 presents timings of PG-PuReMD for two water systems. It can be seen that as the number of GPUs increases, the time per time step for the water systems decreases consistently, suggesting that PG-PuReMD scales well when the system size is constant while increasing the number of

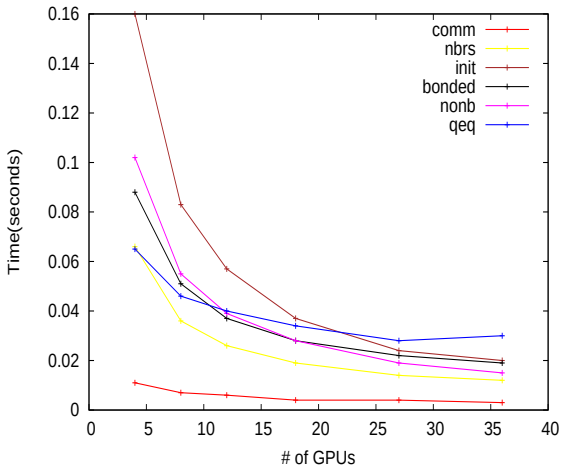

Figure 5: Strong scaling results for key components of ReaxFF of Water-80K systems.

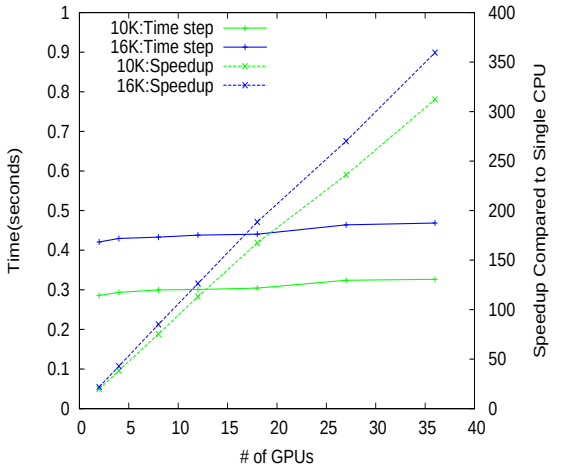

Figure 6: Weak scaling results for Water systems (10K/GPU and 16K/GPU).

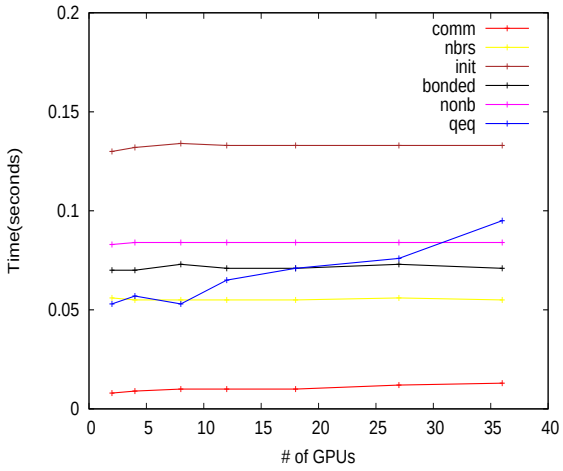

Figure 7: Weak Scaling results for key components of ReaxFF of Water systems (16K/GPU)

GPUs, at least to moderate configurations. Because of limited available memory on GPUs (3GB of global memory per GPU), the water-200K system can only be run on 12 GPUs and beyond.

Figure 5 presents the timings of major components (per time-step) for the water 80K system. *comm*, *nbrs*, *init* and *nonb* scale well as the number of GPU's is increasing for this system. The time for *comm* is almost constant after 4 GPUs. *nbrs* and *nonb* use multiple threads per atom, when using 36 GPUs, each GPU is processing about 2222 atoms. With 16 threads per atom and a block size of 256 *nbrs* kernel has about 139 blocks and *nonb* kernel (32 threads/atom) has about 278 blocks yielding enough thread blocks to keep the *SM*'s saturated because of which we see the curves for these two kernels trending downwards consistently as the GPUs are increasing for this system. Because of multiple threads per atom, the number of thread blocks decreases sharply for these two kernels when increasing the number of GPUs, which is the reason why steep drop is noticed for these two curves for small number of GPUs. The time for the *bonded* kernel is almost constant beyond 27 GPUs. This is because all the bonded interactions except uses hydrogen-bond interaction uses single thread per atom implementation. With 27 GPUs, each GPU processes about 2962 atoms, resulting in 12 blocks each of 256 threads. From this point onwards we have less number of thread blocks than *SM*'s indicating that it reached the lower bound on its timing per time-step. Since all the bonded interactions, except hydrogen-bond interactions, uses single thread per atom implementation, we see consistent drop when using small number of GPUs and it tends to become flat after 18 GPUs, after this point the number of thread blocks created for these kernels become less than the number of *SM*'s on the GPUs. The *init* kernel, which builds bond-list, hydrogen-bond list and QEq matrix, drops sharply compared to other kernels when small number of GPUs are used, because each GPU is processing large number of atoms(quantum of work per each thread is also large) and we have large number of thread blocks. As we increase the number of GPUs, this curve tends to become flatter since the number of thread blocks becomes less than the number of *SM*'s on the GPU.

The communication bound parts of PG-PuReMD do not scale as well. *QEq* is one of the most expensive parts of PG-PuReMD. Since QEq involves four communication operations (two message exchanges and two global reductions) in every iteration of the linear solver, as the number of GPUs increases the communication overhead increases as well. Moreover, as the sub-domain size decreases, the amount of computation decreases and the communication overhead starts to dominate.

2.2.2 Weak Scaling Results

We used water systems with 10,000 atoms per GPU and 16,000 atoms per GPU for benchmarking the PG-PuReMD application under weak scaling scenarios. For water systems with 16,000 atoms per GPU, we achieve a speedup of 359x on 36 GPUs when compared to single CPU time for the same water system. Figure 6 illustrates the weak scaling performance and speedup's achieved.

Figure 7 presents weak scaling results of water system with 16K atoms/GPU. For all the components, we observe that the time per time-step is almost constant, yielding excellent scaling over the range of GPUs used. An interesting observation can be made about the *QEq* computation – this computation is heavily dependent on the communication. Furthermore, it is executed in a lockstep fashion for every iteration of the linear solve. The *QEq* time almost doubles from 4 GPUs case to 36 GPUs. This can be attributed to the amount of communication and execution of each iteration in a lock step fashion.

Table 8 presents efficiency results for water systems under weak scaling. We achieve a speedup of about 10x per MPI process when compared to PuReMD implementation. This results in decreased

	10K per GPU		16K per GPU	
GPU's	$\frac{QEq}{TotalTime}$%	Efficiency	$\frac{QEq}{TotalTime}$%	Efficiency
1	11.80	100.00	10.99	100.00
2	13.11	95.97	11.66	99.17
4	14.61	93.46	12.33	97.10
8	15.45	91.52	12.93	96.33
12	16.10	91.21	13.61	95.27
18	16.47	90.17	13.70	94.74
27	20.53	84.68	17.78	89.92
36	21.04	84.02	18.25	89.04

Table 8: Efficiency results of Water system for weak scaling simulations

parallel efficiency as we increase the number of GPUs. PuReMD [4] achieves an efficiency of 99% with 32 MPI processes; PuReMD runs on CPUs and it takes 16 MPI processes to completely use a single CPU. In comparison PG-PuReMD achieves about 84% efficiency for comparable number of MPI processes. The reduction in efficiency is a consequence of the faster serial execution by the GPUs.

3 Technology Transfer

Three major outcomes of the project are:

- Public domain release of the PuReMD, Parallel PuReMD, and PuReMD-GPU, packages. These packages are used by over 100 research groups, worldwide.
- Integration of PuReMD into LAMMPS (fix Reax/C). This fix is used by a large number of users (we do not track this user group, but provide timely response to the user group feature requests and bug tracking).
- We are currently working with Scientific Computing and Modeling Inc (a company based in Amsterdam) to incorporate PuReMD into the ADF Molecular Modeling Suite. While we make the software available to them free of cost, SCM optimizes it, validates it, and supports it for its paying customers.

4 Human Resources Development

The project funded two students, Hasan Metin Aktulga and Joseph Fogarty. Dr. Aktulga has graduated and joined Lawrence Berkeley Lab.

References

[1] S. Kylasa, H. Aktulga, and A. Grama, *PG-PuReMD: A Parallel-GPU Reactive Molecular Dynamics Package*, *Proceedings of IPDPS*, submitted, 2013.

[2] S. Kylasa, H. Aktulga, and A. Grama, *PuReMD-GPU: A Reactive Molecular Dynamic Simulation Package for GPUs*, *Journal of Molecular Graphics and Modeling*, submitted, 2013.

[3] H.M. Aktulga, S. Pandit, A.C.T. van Duin, A. Grama *Reactive Molecular Dynamics: Numerical Methods and Algorithmic Techniques SIAM J. Scientific Computing*, 34(1), 2012.

[4] H.M. Altulga, J.C. Fogarty, S.A. Pandit, A. Y. Grama, *Parallel Reactive Molecular Dynamics: Numerical Methods and Algorithmic Techniques*, *Parallel Computing*, 38(4-5): 245-259, 2012.

[5] J.C. Fogarty, H.M. Aktulga, A.C.T. van Duin, A.Y. Grama, S.A. Pandit, *A Reactive Simulation of the Silica-Water Interface*, *J Chem Phys.*, 132(17):174704, 2010.

[6] Y. Park, H.M. Aktulga, A.Y. Grama, A. Strachan, *Strain relaxation in Si/Ge/Si nanoscale bars from MD simulations*, J Appl Phys, 106, 034304, 2009.

[7] H.M. Altulga, A. Y. Grama, *PuReMD User Guide*, Technical Report, Purdue University, 2010.