LAMMPS strong scaling performance optimization on Blue Gene/Q

Leadership Computing Facility

About Argonne National Laboratory

Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The Laboratory's main facility is outside Chicago, at 9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

DOCUMENT AVAILABILITY

Online Access: U.S. Department of Energy (DOE) reports produced after 1991 and a growing number of pre-1991 documents are available free via DOE's SciTech Connect (<http://www.osti.gov/scitech/>)

Reports not in digital format may be purchased by the public from the National Technical Information Service (NTIS):

U.S. Department of Commerce National
Technical Information Service 5301 Shawnee
Rd
Alexandra, VA 22312
www.ntis.gov
Phone: (800) 553-NTIS (6847) or (703) 605-6000
Fax: (703) 605-6900
Email: **orders@ntis.gov**

Reports not in digital format are available to DOE and DOE contractors from the Office of Scientific and Technical Information (OSTI):

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
www.osti.gov
Phone: (865) 576-8401
Fax: (865) 576-5728
Email: **reports@osti.gov**

Disclaimer

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor UChicago Argonne, LLC, nor any of their employees or officers, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of document authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, Argonne National Laboratory, or UChicago Argonne, LLC

LAMMPS strong scaling performance optimization on Blue Gene/Q

Paul Coffman
IBM
3605 Hwy 52N
Rochester, MN 55901
608-519-0874
pkcoff@us.ibm.com

Wei Jiang
Argonne National Laboratory
Lemont, IL 60439
630-252-8688
wjiang@alcf.anl.gov

Nichols A. Romero
Argonne National Laboratory
Lemont, IL 60439
630-252-3441
naromero@alcf.anl.gov

Leadership Computing Facility

November 10, 2014

ABSTRACT

LAMMPS "Large-scale Atomic/Molecular Massively Parallel Simulator" is an open-source molecular dynamics package from Sandia National Laboratories. Significant performance improvements in strong-scaling and time-to-solution for this application on IBM's Blue Gene/Q have been achieved through computational optimizations of the OpenMP versions of the short-range Lennard-Jones term of the CHARMM force field and the long-range Coulombic interaction implemented with the PPPM (particle-particle-particle mesh) algorithm, enhanced by runtime parameter settings controlling thread utilization. Additionally, MPI communication performance improvements were made to the PPPM calculation by re-engineering the parallel 3D FFT to use MPICH collectives instead of point-to-point. Performance testing was done using an 8.4-million atom simulation scaling up to 16 racks on the Mira system at Argonne Leadership Computing Facility (ALCF). Speedups resulting from this effort were in some cases over 2x.

General Terms

Performance.

Keywords

LAMMPS, performance, strong-scaling, PPPM.

1. INTRODUCTION

LAMMPS is a classical molecular dynamics code, and an acronym for Large-scale Atomic/Molecular Massively Parallel Simulator. LAMMPS has potentials for soft materials (biomolecules, polymers) and solid-state materials (metals, semiconductors) and coarse-grained or mesoscopic systems. It can be used to model atoms or, more generically, as a parallel particle simulator at the atomic, meso, or continuum scale.[1] It has an extensive user base, and may be favored over other packages because of its extensive array of force fields, particularly in the domain of material science research. During the production readiness effort for the 48-rack Blue Gene/Q system named Mira at Argonne National Laboratory it was identified as an application with sufficiently high user demand but strong-scaling deficiencies to warrant a performance optimization effort. A benchmark was identified using the short-range Lennard-Jones term of the CHARMM force field (hereafter referred to as the pairlist computation) and the long-range Coulombic interaction implemented with the PPPM algorithm (hereafter referred to as the PPPM computation) which, if successfully optimized, could benefit a significant user base, effectively making the scope of the project the strong-scaling optimization of these two force fields. I/O optimization was not part of this project.

2. BUILDING AND RUNNING LAMMPS ON BLUE GENE/Q

All building and testing of LAMMPS for this optimization effort was done on Blue Gene/Q software release V1R2M0. The LAMMPS code base is C++, therefore the XL C++ cross-compiler for Blue Gene/Q version 12.1 was used to compile all the modules. Compiler optimization level 3 (-O3) was used for all modules, including the -qhot (performs high-order loop analysis and transformations) and -qunroll (tells the compiler to more aggressively search for opportunities for loop unrolling) optimization options. The multi-threading is all done via

OpenMP so the -qsmpt=omp option was also specified to the compiler to enable it. The PPPM computation uses fast Fourier transforms (FFTs) with functions from the single-precision version of FFTW3 (<http://www.fftw.org/>), therefore the fftw3f library is linked in to build the stand-alone binary. All communication is done with MPI, and MPICH-2 is the current standard supported on Blue Gene/Q V1R2M0. All benchmarks were run with the stand-alone LAMMPS binary, which takes a parameter file and various runtime arguments detailing the simulation based on data files existing in the current working directory. Since I/O was not part of this performance optimization project, no checkpoint or restart files were utilized after the start of the simulation.

3. LAMMPS AND THE BLUE GENE/Q ARCHITECTURE

Blue Gene/Q is the follow-on to Blue Gene/P in the family of supercomputers from IBM and the next step toward exascale computing in price performance, scalability, power efficiency, and system reliability. The Blue Gene/Q Compute chip is a SystemonChip (SoC) design combining CPUs, caches, network, and a messaging unit on a single chip. [2] There are 16 64-bit PowerPC A2 cores available for application usage, each running 4 hardware threads which can be oversubscribed with up to 5 software threads. Additionally, each core has a SIMD quad floating point unit (Quad Processor eXtension, QPX), an L1 prefetching unit (L1P) supporting both linear stream and list (perfect) prefetching, and a wake-up unit to reduce certain thread-to-thread interactions. [2] Each core has its own 16 KB L1 data cache and shares a 2MB slice of L2 cache connected by a crossbar which supports transactional memory and speculative execution at the hardware level. The memory hierarchy latency is as follows: L1 is 6 cycles, L1P is 24 cycles, L2 is 84 cycles, and main memory is 346 cycles. [3] The A2 core runs at 1.6 GHz and if the QPX is fully utilized allows for 4 FMAs per cycle, translating to a peak performance per core of 12.8 GFlops, or 204.8 GFlops for the chip. [2] Each node contains a compute chip and 16 GB of DDR3 memory. 32 nodes compose a node board, 16 node boards compose a midplane, and 2 midplanes compose a rack, totalling 1024 nodes. Node-to-node communication is facilitated by a 5-d torus network, and each compute node has 10 communication links with a peak total bandwidth of 40 GB/second.[2].

From an MPI perspective an application can partition each node from 1 to 64 MPI ranks (ranks-per-node) by multiples of 2. Intra-node communication between ranks is done via shared memory whereas node-to-node communication utilizes the messaging unit and torus. The less ranks-per-node means a greater number of hardware threads available per rank, so rank-per-node/number-of-hardware-thread permutations equal 64, ie 1/64, 2/32, 4/16, 8/8, 16/4, 32/2 and 64/1. The multi-threading code in LAMMPS does not lend itself to make effective use of thread oversubscription, so the number of software threads must match the number of available hardware threads within each rank for optimum performance. As the proceeding sections will illustrate, from the outset it became apparent the main bottleneck to strong-scaling performance with large multi-rack block sizes was going to be MPI communication. Baseline measurements also indicated the optimum ranks-per-node/number-of-threads ratio decreased as the block size increased, beyond 4 racks the baseline ran optimally at 1 rank-per-node and 64 threads. In the scope of the domain decomposition of this application, one way to minimize the communication bottleneck is to run with fewer ranks-per-node,

thereby lessening communication between the ranks at the expense of computation within the ranks. Therefore, the focus of the performance effort became maximizing performance running at 1 rank-per-node and 64 threads, which then also meant maximizing the computational scaling of multiple threads on the chip and making optimal use of the chip architecture described above.

4. LAMMPS BENCHMARK

The benchmark chosen for this performance optimization effort was an 8.4 million atom brush structure simulation of poly-isopropyl-acrylamide (PNIPAM). The CHARMM force field implemented by the pairlist computation is well-suited for this material simulation and is one of the most accurate all-atom parameter force fields for biological and material molecular dynamics simulations - it contains complete additive bonded (bond, angle, dihedral and improper dihedral) and nonbonded (Lennard-Jones and Coulombic) interaction terms.[4] PNIPAM is a temperature sensitive polymer that undergoes a coil-to-globule transition as the temperature is raised. Investigating such a large scale conformational change is one of the most challenging tasks in soft condensed matter simulations.[5] PNIPAM requires accurate all-atom force fields whose interaction types must cover all atomic length scales, making the long-range scalability of the Coulombic PPPM computation of critical importance. A structural material simulation of PNIPAM utilizing the CHARMM force field is more scientifically relevant than many existing benchmark systems utilizing simplified force fields, and presents a highly non-trivial as well as accurate all-atom benchmark system upon which to investigate the performance of LAMMPS. Unless otherwise noted, all investigation, data and results are in the context of the benchmark described above running at 1 rankpernode using 64 OpenMP threads.

5. PROFILING AND ANALYSIS TOOLS, METHODOLOGIES AND ENVIRONMENTS

To minimize computational resource utilization and maximize turnaround, an abbreviated version of the benchmark was provided which would run sufficiently on 256 nodes in the IBM development lab where the primary author has easy access, so a majority of the profiling and tuning effort was done on this small scale and then applied to the large scale benchmark running on multi-rack blocks on Mira at the ALCF. LAMMPS provides it's own set of internal performance profiling metrics which are printed out after the results at the end of the simulation, including the actual wall-time and the percent of total calculation. These metrics were used as the first level of profiling. The code was then compiled, linked and run with -pg to instrument the GNU-profiler, which gave a decent profile of the serial functional components but a less-than-useful profile of the multi-threaded functional components. Beyond that was the use of internally developed profiling libraries using PMPI for the MPI communication and hardware performance counters for the

computation, which provided a very insightful view of computational issues within the threads. Finally, manual timebase instrumentation of the code, consisting of collecting the data in structures throughout the simulation and then printing out the values at the end proved very valuable in terms of determining the overall wall-time contribution of various functions to the overall benchmark time across all timesteps in the simulation.

6. OPTIMIZATION INVESTIGATION

6.1 LAMMPS internal profiling initial analysis

LAMMPS provides it's own set of internal performance profiling metrics which is printed after the results at the end of the simulation, including the actual wall-time and the percent of total calculation. The 2 metrics of interest for this project were the 'Pair' metric which was the total cumulative time for the pairlist computation across all timesteps and the 'Kspce' time which, for this benchmark, was the PPPM computation time

A baseline analysis of the MPI communication scaling was accomplished by holding the number of threads at 64 and running at 1 rank-per-node and then scaling the number of nodes from 1 to 16 racks (wall times are in seconds):

Table 1. MPI communication scaling of force fields baseline

Number of Racks	Pairlist wall time	Pairlist percent	PPPM wall time	PPPM percent
1	117.04	38.32	136.47	44.69
2	59.72	31.53	92.16	48.65
4	31	23.88	72.8	56.13
8	17	14.8	74	64.72
16	9.48	9.01	74.18	70.52

This data indicated the pairlist mpi communication was scaling pretty well, but the PPPM mpi communication was not. Beyond 4 racks scaling completely stopped, as the wall time was staying basically the same, but the percent of the total simulation time increased.

A baseline analysis of the algorithmic thread scaling was accomplished using the abbreviated version of the benchmark running on a fixed number of nodes (256) and running at 1 rank-per-node, scaling from 1 to 64 threads (wall time is in seconds):

Table 2. Multi-threaded scaling of force fields baseline:

Number of threads	Pairlist wall time	PPPM wall time
1	524.03	123.81
2	265.05	66.49
4	133.88	38.49
8	67.97	22.32
16	34.78	15.04
32	18.59	12.72
64	11.47	19.39

Analysis of this data again showed the pairlist multi-threaded computation was scaling pretty well, but the PPPM was not, and actually got worse at 64 threads. Therefore, to get stronger scaling of the benchmark, the PPPM MPI communication and multi-threaded computation needed to scale better, and additionally the overall time-to-solution could be improved if the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Conference '10, Month 1-2, 2010, City, State, Country.

Copyright 2010 ACM 1-58113-000-0/00/0010 ...\$15.00.

serial code within the threads of the pairlist computation was better optimized.

6.2 Thread control environment variable tuning

There are several environment variables that control OpenMP and thread utilization, all of which were thoroughly investigated. The following combination had the greatest positive effect on performance, particularly at 64 threads:

Table 3. Optimum thread control settings:

Environment variable name	Value
BG_SMP_FAST_WAKEUP	YES
OMP_WAIT_POLICY	ACTIVE
BG_THREADLAYOUT	1

The explanation for this is the LAMMPS code is interspersed with serial and multi-threaded code fragments all within the same timestep, so the combination of BG_SMP_FAST_WAKEUP=YES and OMP_WAIT_POLICY=ACTIVE settings effectively hold the threads in a barrier during the serial execution sections as opposed to having to respawn them when a multi-threaded code section is again entered. Furthermore, for 64 threads the BG_THREADLAYOUT=1 setting changes how the threads are assigned to the processor, going depth-first within the cores instead of the default breadth-first across the cores, resulting in different data being accessed by threads on the same core which in turn happened to have less contention for L1 and L2 cache (ie less cache evictions of one thread's data by another's), an explanation the performance counter data seemed to support. [6] [7]

6.3 Transactional Memory investigation

The OpenMP code was analyzed to see if any areas could benefit from the utilization of Transactional Memory (TM). BG/Q implements hardware support for memory speculation. It is controlled by the shared multi-versioning L2-cache. During memory speculation, the L2 cache tracks state changes caused by the speculative thread, and keeps them separate from the main memory state. [7] The only areas of shared data (omp critical sections) among threads was for the memory management of thread data, so TM was attempted in these cases but was not successful in that the performance did not improve.

6.4 Thread data reduction optimization

The algorithmic methodology of the multi-threaded implementation is to replicate all the atomic-level data for the multi-threaded functions within the pairlist and PPPM computations into a one-dimensional array strided by the thread id, so that as each thread runs it updates its own copy of data. When the function completes, the separate data from each thread is brought together and reduced. The time to perform this thread data reduction increased with the number of threads, so at 64 threads this amount of time was considerable and contributed significantly to the loss of multi-threaded scaling. Data reduction for the pairlist and PPPM computations were handled by the global function 'data_reduce_thr' and additionally the charge density discretization within the PPPM computation was handled by the global function 'data_reduce_fft'. Optimizations of these functions were based on the architecture of the Blue Gene/Q chip

- the L1 data cache line is 2x64 bytes wide, so 64 contiguous bytes are loaded at a time. The thread data is an array of double-precision floating point numbers, which are 8-bytes wide. Therefore, the first optimization was to manually unroll the loops to access and process the data 8 elements at a time. Although this code change was made to match the Blue Gene/Q architecture, many other architectures would benefit from this code, or some slight permutation of it. Additionally, the xLC built-in L1 register prefetching function `__prefetch_by_load` was utilized at key points in the loop. These optimizations in the context of the thread runtime options discussed in Section 6.2 moved the data access for the cpu's further up the memory latency hierarchy and thereby improved performance. Performance counter analysis revealed the effect of the optimizations running on 4 racks - this sample is for the 'data_reduce_thr' function on rank 0 but the data is similar on other ranks and for the 'data_reduce_fft' function as well:

Table 4. Performance counter metrics for thread data reductions

Metric	Baseline value	Optimized value
Average cycles	12.37 billion	7.54 billion
L1P Misses	4.77 billion	3.38 billion
L2 Misses	327.88 million	105.14 million
L2 lines loaded from memory	318.64 million	294.76 million
L2 lines stored to memory	190.22 million	157.55 million

This performance counter analysis indicates that the optimized function uses 39% fewer cycles because there are about 10% less loads and stores to main memory, 68% less L2 cache misses and 29% less L1P misses

6.5 Pairlist computation optimization

Given the need to run the application with multiple threads for optimum scaling, the multi-threaded (as opposed to the serial) version of the pairlist computation was the focus of the optimization effort. A previous performance engineering effort at Sandia National Laboratory had optimized the serial version of this code, so the first step was to apply the serial optimizations done as part of that effort to the OpenMP version.[8] Once those optimizations were in place, further tuning was investigated. The manual insertion of qpx vector instructions were investigated, but their successful implementation was prohibited due to the structure of the data and the algorithm. Also due to the non-sequential nature of the data access from the atomic data arrays, the L1P was highly ineffective - therefore utilization of the xLC built-in L1 register prefetching function `__prefetch_by_load` was successfully utilized at key points in the computational loops which, in combination with Sandia optimizations and the thread runtime options discussed in Section 6.2, moved the data access for the cpu's further up the memory latency hierarchy and thereby improved performance. Performance counter analysis revealed the effect of the optimizations running on 4 racks - this sample is from rank 0 but the data is similar on other ranks:

Table 5. Performance counter metrics for pairlist computation

Metric	Baseline value	Optimized value
Average cycles	40.84 billion	35.35 billion
L1P Misses	26.9 billion	23.97 billion
L2 Misses	186.52 million	142.79 million

L2 lines loaded from memory	190.16 million	145.97 million
L2 lines stored to memory	146.52 million	91.81 million

This performance counter analysis indicates that the optimized function uses about 14% fewer cycles because there are about 57% less loads and stores to main memory, 24% less L2 cache misses and 11% less L1P misses.

6.6 PPPM computation optimization

Multi-threaded scaling issues in the PPPM computation were addressed in several ways. The optimization of the 'data_reduce_fft' thread data reduction function described previously had a significant effect. Additionally, during the discretization of the charge density to the 3-D spatial grid decomposition, there were many nested loops in the computation, and at key points the L1 data cache block touch function (`_dcbt`) was utilized to improve the L1 data cache utilization. Finally, there is an `smp` version of FFTW3 which uses OpenMP which was analyzed and tested for performance, however the actual amount of data and time spent in each FFTW call was insufficient to make adequate use of additional threads.

MPI communication scaling issues were addressed by re-engineering the parallel Fast-Fourier Transform (FFT) implementation to use an MPICH collective instead of point-to-point communication. The PPPM implementation utilizes a parallel 3D-FFT methodology for calculating the net force of long-range Coulombics on individual atoms. The algorithm first discretizes the charge density for the unit cells into single points on a 3-D spatial grid decomposition, and then this discretized charge density is moved to an FFT-decomposition where each rank owns a set of 1-D columns of the 3-D FFT mesh. This requires each rank to send all of its grid data to a handful of different ranks. Similarly each rank will fill in its FFT columns with the data received from several other ranks.[9] The plan for the exchange of data between ranks is created during PPPM initialization in a structure called 'remap_plan_3d' and is used unchanged for each timestep. During the PPPM computation for each timestep, 3 separate 1-D FFTs are performed using the FFTW3 package. Interleaved with the 1-D FFT calls are transposes of the 3-D grid to bring new sets of columns onto the processors by calling a function named 'remap_3d' which uses information in the precomputed 'remap_plan_3d'. The existing 'remap_3d' code accomplished this by using point-to-point communication with an `MPI_Irecv / MPI_Send / MPI_Waitany` loop with information from the 'remap_plan_3d' structure telling it what data to exchange where. Performance analysis revealed that most of the time was spent in the 'remap_3d' function and very little relative time was spent in the actual FFTW3 calls. Further analysis revealed that the list of ranks were fully connected communication rings, where every rank within the ring exchanges data with every other rank in the ring. This was a case where an `MPI_alltoallv` collective could replace the `MPI_Irecv / MPI_Send / MPI_Waitany` loop much more efficiently. This was accomplished with changes to the 'remap_3d_create_plan' function during the PPPM initialization within which the 'remap_plan_3d' structure was populated and then later utilized during the PPPM computation in the 'remap_3d' function. During plan creation, an MPI sub-communicator was created based off an `MPI_Group` composed of the ranks within the send and receive lists for the original plan. The send and receive lists were then

altered to include the originating rank and sorted, so all ranks sharing the sub-communicator would have the same data in the same order from which the same buffer displacements could be computed. Then in the 'remap_3d' function, the send and receive buffers, counts and displacements could all be computed off these same lists and an `MPI_Alltoallv` collective could be used in place of the `MPI_Irecv / MPI_Send / MPI_Waitany` loop. PMPI analysis isolated to the PPPM computation revealed the effect of the collective running on 4 racks (total times are in seconds).

Table 6. PMPI metrics for PPPM computation

MPI Routine	Baseline # of calls	Baseline total time	Optimize # of calls	Optimize total time
MPI_Send	2.64 million	6.84	558k	3.44
MPI_Irecv	2.64 million	1.26	558k	0.39
MPI_Wait	49k	6.32	49k	4.75
MPI_Waitany	2.59 million	30.62	508k	4.73
MPI_Allreduce	217	0.06	217	0.02
MPI_Alltoallv	0	0	33k	10.63
Total time		45.1		23.96

This data indicates that at 4 racks the effect of the collective essentially cut in half the amount of time spent in MPI communication for the PPPM computation, which has an even greater effect as the application scales to larger and larger blocks

7. BENCHMARK RESULTS

Testing of the cumulative effect of the optimizations described in Section 6 on the benchmark described in Section 4 was done on Mira at the ALCF. This first chart depicts the results of running the benchmark at 1 rank-per-node using 64 OpenMP threads comparing the baseline to the optimized version of the application, scaling from 1 to 16 racks:

To further exemplify the speedup of the optimizations over the baseline, this next chart depicts the speedup factor of the optimizations on the application, scaling from 1 to 16 racks:

These results show that the effect of the optimization effort significantly improves performance at any scale, and the improvement gets larger as the application scales - the baseline scaling slows to marginal amounts at 4 racks, whereas the optimized version keeps scaling to a far greater extent to 16 racks, albeit non-linearly.

Finally, this table depicts the optimized version MPI communication scaling from 1 to 16 racks for comparison against the baseline table in section 6.1 (wall times are in seconds):

Table 7. MPI communication scaling of force fields with optimizations

Number of Racks	Pairlist wall time	Pairlist percent	PPPM wall time	PPPM percent
1	98.50	41.91	101.43	43.16
2	50.75	37.30	61.74	45.37
4	25.70	33.53	38.10	49.71
8	13.41	24.11	31.27	56.19
16	7.18	17.52	25.10	61.27

This data indicates that pairlist computation time is still scaling linearly but is simply faster at any scale, whereas the PPPM computation keeps scaling past 4 racks instead of plateauing and is also faster at any scale, however it is still the bottleneck. Further performance optimization efforts could include creating MPI mapping files to minimize the number of hops in the torus between nodes that share the same communication ring within the FFT 'remap_3d' calls

8. SUMMARY

The goal of this performance optimization effort was to analyze the computational performance of LAMMPS on Blue Gene/Q, and then, based off that analysis, make general changes to the application to exploit the Blue Gene/Q architecture but still be applicable to other architectures, which could then be contributed to the open source code base, and then additionally make specific tuning changes to the code and runtime environment that were specific to the Blue Gene/Q architecture to further improve

performance. This paper detailed that process, including technical details of the Blue Gene/Q architecture and how it was leveraged to improve performance. Although the optimizations maximized performance the most when running at 1 rank-per-node and 64 threads, significant improvements over the baseline were still measured at other ranks-per-node/number-of-thread configurations. Final testing results showed significant optimized speedup factors over the baseline ranging from 1.3x to 2.56x, increasing as the application scaled.

9. ACKNOWLEDGEMENTS

We would like to thank Bob Walkup and John Gunnels from the IBM Watson Research Lab for their consultation on a variety of topics related to this performance optimization effort. We would also like to thank the staff of the ALCF for the reliable access to the Mira system, and to the IBM Rochester development lab and support staff for the reliable access to the development and test systems located there. Finally, we would like to thank Steve Plimpton from Sandia National Laboratories for his correspondence on the re-engineering effort for the PPPM computation.

10. REFERENCES

- [1] Plimpton, S. (n.d.) LAMMPS Molecular Dynamics Simulator. Retrieved April 21, 2013 from <http://lammps.sandia.gov/>.
- [2] Habib, S., Morozov, V., Finkel, H., Pope, A., Heitmann, K., Kumaran, K., Peterka, T., Insley, J., Daniel, D., Fasel, P., Frontiere, N., Luki, Z. (2012) The Universe at Extreme Scale: Multi-Petaflop Sky Simulation on the BG/Q. Retrieved March 4, 2013 from <http://conferences.computer.org/sc/2012/papers/1000a004.pdf>.
- [3] Chung, I., Kim, C., Wen, H., Cong, G. (2012) Application Data Prefetching on the IBM Blue Gene/Q Supercomputer. Retrieved April 20, 2013 from <http://conferences.computer.org/sc/2012/papers/1000a075.pdf>.
- [4] Brooks BR, Bruccoleri RE, Olafson BD, States DJ, Swaminathan S, Karplus M (1983). CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. *J Comp Chem* 4 (2): 187–217. doi:10.1002/jcc.540040211.
- [5] Schild, H. G. (1992) Poly(N-isopropylacrylamide): experiment, theory and application. *Progress in Polymer Science*, 17 (2), 163–249.
- [6] IBM Corp. (n.d.) IBM XL C/C++ for Blue Gene/Q, V12.1 Compiler reference. Retrieved February 18, 2013 from <http://www-01.ibm.com/support/docview.wss?uid=swg27027065&aid=1>.
- [7] Gilge, M. (2012) IBM Blue Gene/Q Application Development Redbook. Retrieved January 22, 2013 from www.redbooks.ibm.com/redpieces/pdfs/sg247948.pdf.
- [8] Fischer, J., Richie, D. and Natoli, V. (2006) PET Project Report: Optimization of LAMMPS Molecular Dynamics Code. Retrieved March 6, 2013 from http://www.sandia.gov/CSRF_report_2007/References/Section_14/malins_sec14_ref9_fischer_LAMMPS_june_2006.pdf.
- [9] Plimpton, S., Pollock R. and Stevens M. (1997) Particle-Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations. *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*

Leadership Computing Facility

Argonne National Laboratory
9700 South Cass Avenue, Bldg. 240
Argonne, IL 60439

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC