Early Experiences Porting the NAMD and VMD Molecular Simulation and Analysis Software to GPU-Accelerated OpenPOWER Platforms

John E. Stone[1(✉)], Antti-Pekka Hynninen[2], James C. Phillips[1], and Klaus Schulten[1,3]

[1] Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign, Urbana, IL, USA
jestone@illinois.edu, johns@ks.uiuc.edu
[2] Oak Ridge Leadership Computing Facility, Oak Ridge National Laboratory, Oak Ridge, TN, USA
[3] Department of Physics, University of Illinois at Urbana-Champaign, Urbana, IL, USA

Abstract. All-atom molecular dynamics simulations of biomolecules provide a powerful tool for exploring the structure and dynamics of large protein complexes within realistic cellular environments. Unfortunately, such simulations are extremely demanding in terms of their computational requirements, and they present many challenges in terms of preparation, simulation methodology, and analysis and visualization of results. We describe our early experiences porting the popular molecular dynamics simulation program NAMD and the simulation preparation, analysis, and visualization tool VMD to GPU-accelerated OpenPOWER hardware platforms. We report our experiences with compiler-provided autovectorization and compare with hand-coded vector intrinsics for the POWER8 CPU. We explore the performance benefits obtained from unique POWER8 architectural features such as 8-way SMT and its value for particular molecular modeling tasks. Finally, we evaluate the performance of several GPU-accelerated molecular modeling kernels and relate them to other hardware platforms.

1 Introduction

Atomic-detail molecular dynamics (MD) simulation provides researchers with a powerful computational microscope that permits the study of biomedically-relevant processes that are too fast to observe first-hand, and that occur in the crowded molecular environment of living cells, that cannot be seen with even the most advanced experimental microscopes. Many societal challenges are addressed by biomolecular modelers employing state-of-the-art parallel computing platforms, for example, treatment of viral infections [1,2] and addressing the antibiotic resistance crisis [3]. The cellular processes of interest to biomedical researchers take place in molecular assemblies made of millions to hundreds of

M. Taufer et al. (Eds.): ISC High Performance Workshops 2016, LNCS 9945, pp. 188–206, 2016.
DOI: 10.1007/978-3-319-46079-6_14

millions of atoms. Such simulations are extremely demanding in terms of preparation, computation, storage, analysis, and visualization, and they continue to push the limits of parallel computing.

State-of-the-art petascale supercomputing platforms such as Blue Waters [4] and Titan [5] contend with significant challenges posed by constraints on space, power, and cooling. To achieve higher application performance, future systems must directly address these challenges with performant and energy efficient computing technologies that take maximal advantage of many-core CPUs, GPU accelerators, non-volatile storage systems, and new intra- and inter-node interconnects. One of the upcoming leadership-class computing systems for open science will be Summit, a pre-exascale supercomputer to be composed of roughly 3,400 GPU-accelerated compute nodes, fielded by the U.S. Department of Energy, and housed at the Oak Ridge Leadership Computing Facility. While a development environment directly comparable to the final Summit system is not yet available, existing OpenPOWER platforms support GPU accelerators and associated compilers and libraries, allowing application porting to begin today.

Below we describe the adaptation of representative and performance-critical algorithms in the widely used molecular dynamics program NAMD [6,7], and the molecular analysis and visualization tool VMD [8–10], on an IBM S822L OpenPOWER hardware platform with NVIDIA Tesla K40m GPUs.

2 Overview of ORNL Crest Test System

Except where noted, the application porting and performance evaluation activities reported here were performed using the “Crest” development systems made available through the Center for Accelerated Application Readiness (CAAR) at the Oak Ridge National Laboratory (ORNL), in preparation for the next-generation Summit supercomputer. The ORNL Crest system is currently composed of four compute nodes and an associated login node for software development and testing. The Crest compute nodes are IBM S822L servers configured with two 3.7 GHz 10-core POWER8 dual-chip CPU modules (DCMs), 256 GB of RAM among four NUMA nodes, four NVIDIA Tesla K40m GPUs, and two Mellanox Connect-IB FDR InfiniBand (56 Gbit/s) NICs. Figure 1 shows a simplified block diagram of the S822L hardware used in the Crest compute nodes.

The current software environment on Crest supports several compilers including GCC 4.9.3 and IBM XLC 13.1.2, both supporting compile-time auto-vectorization of performance critical loops using POWER8 VSX SIMD instructions. Although NAMD and VMD are both endianism-independent, the little-endian byte ordering used by the Linux operating system for OpenPOWER is beneficial for applications originally developed on popular little-endian platforms such as Intel x86, and it is necessary for efficient use of GPU accelerators. The NVIDIA Tesla K40m GPUs are supported by natively-hosted CUDA 7.5 compilers, and an assortment of GPU-accelerated subroutine libraries. At the time of writing, the CUDA 7.5 implementation for OpenPOWER does not yet support peer-to-peer GPU transfers. Below we discuss the use of specific compiler optimization features in the context of NAMD and VMD algorithms.

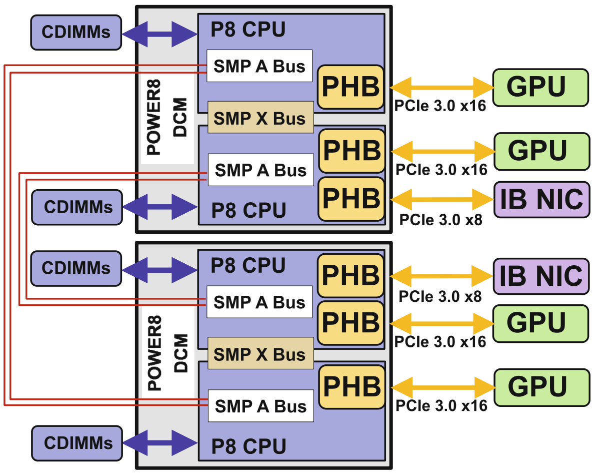


Fig. 1. Simplified block diagram of the IBM S822L compute nodes used in the ORNL Crest development system. Two POWER8 dual-chip CPU modules (DCMs) are shown with associated NUMA CPU and PCIe bus topology interconnecting the CPUs, PCIe host bridge (PHB) I/O channels, NVIDIA Tesla K40m GPUs, and Mellanox Connect-IB InfiniBand network adapters (IB NIC). The two DCMs contain two 5-core POWER8 processors, each with 8-way hardware simultaneous multithreading (SMT), for a total of 20 CPU cores and 160 SMT threads. Each CPU contains its own memory controller with a 96 GB/s bandwidth channel to a set of four CDIMMs (DIMMs paired with a so-called "Centaur" DIMM controller), giving each DCM 192 GB/s DRAM bandwidth, yielding a full system peak DRAM memory bandwidth of 384 GB/s. The two CPUs on a DCM are linked by a 32 GB/s SMP "X" bus, and the two DCMs communicate using four 12.8 GB/s SMP "A" bus links. Each of the four NVIDIA Tesla K40m GPUs are directly connected to the on-chip PCIe 3.0x16 PCIe host bridge (PHB) channel on a corresponding CPU. System components and peripherals less relevant to molecular dynamics simulation and analysis workloads are not shown.

3 NAMD: Parallel Molecular Dynamics Simulation

For the current Titan system and the future Summit system the majority of arithmetic performance is provided by GPU accelerators. NAMD currently exploits GPU acceleration for the two most compute intensive parts of molecular dynamics simulation: non-bonded force computation and Particle Mesh Ewald (PME) reciprocal computation. The remaining computations, e.g., bonded forces, are performed on the CPU. Continuing this existing GPU acceleration model, here we focus on further optimization of the two GPU parts. The starting point for our work is the NAMD CVS repository version 2015-04-23, which is essentially NAMD 2.10 with CUDA kernels modified to stream results incrementally to the CPU. In the following discussion, all changes and comparisons are made with respect to this version of NAMD. The algorithm changes described below are all new and are planned for inclusion in the NAMD 2.12 release.

3.1 Non-bonded Force Computation

Three major changes were made to the way non-bonded force computation is performed in NAMD. First, we broke up the compute objects that used to be on a single thread block, into "tile lists" that can be computed by any thread block. Second, Newton's third law was applied to eliminate duplicate calculation of pairwise forces. Third, we removed synchronization from the thread blocks to allow individual warps to perform the computation independently. We discuss each of these changes in further detail below.

Non-bonded forces in NAMD are calculated in compute objects that define all pairwise interactions between two sets of atoms. In the GPU-accelerated version of NAMD, computes are further split into 32×32 tiles as illustrated in Fig. 2(a). In Fig. 2(a), gray tiles do not have any atom pairs that interact and will be skipped. In the previous version of NAMD, a single thread block was assigned per compute. In the new version of the kernel, computes are broken into tile lists as shown in Fig. 2(b). Tile lists from different computes can be mixed and executed on any thread block. This gives rise to more flexible execution and a better opportunity for load balancing within warps in the thread block.

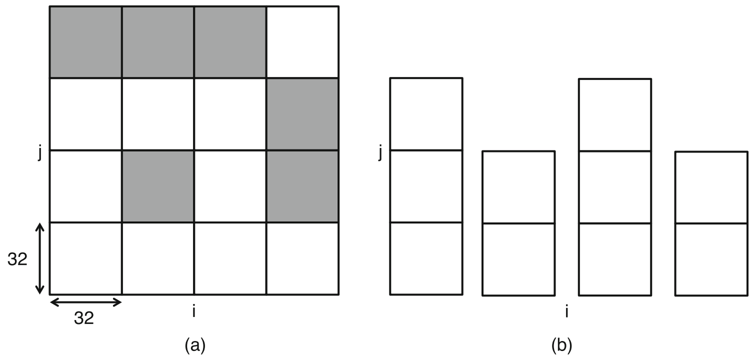


Fig. 2. (a) A so-called compute object consists of 32×32 tiles. Gray tiles have no interactions and are skipped. The previous version of the non-bonded force kernel executed an entire compute on a single thread block. (b) The new non-bonded kernel splits the compute into tile lists that can be executed on any thread block.

The total non-bonded force acting on atom i is given as a sum of all pairwise forces $\boldsymbol{f}_i = \sum_j \boldsymbol{f}_{ij}$. Due to Newton's third law, once $\boldsymbol{f}_{ij}$ are known, the force acting on atom j can be obtained from $\boldsymbol{f}_j = \sum_i -\boldsymbol{f}_{ij}$. In the previous version of the non-bonded force kernel, forces $\boldsymbol{f}_{ij}$ were computed on each thread of a warp and then accumulated to form $\boldsymbol{f}_i$. This is illustrated in Fig. 3(a) for warp size 4, where a warp of threads loops through the tile in horizontal lines starting at line $j = 1$ and ending at $j = 4$. Trying to accumulate to $\boldsymbol{f}_j$ during this loop would give rise to a race condition since all threads in the warp are storing to the same variable. Therefore, a second computation was necessary to compute forces $\boldsymbol{f}_j$

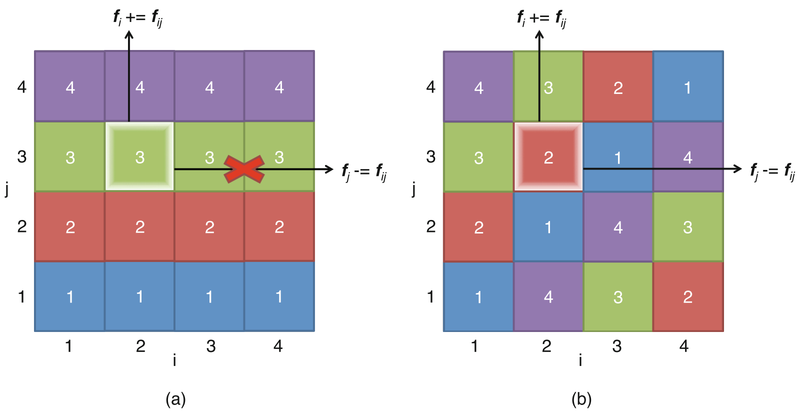


Fig. 3. Pairwise force computation in a 4 × 4 tile (a) previously and (b) in the new scheme. In (a) tile is looped in horizontal lines, while in (b) tile is looped in diagonal lines. The looping index is indicated on the tile elements and elements with the same index belong to the same warp.

by looping through the tile in vertical lines. In the new scheme illustrated in Fig. 3(b), we can avoid this second computation by looping though the tile in diagonal lines [11]. We start from the diagonal marked by "1", and then proceed to the sub-diagonal marked by "2", etc. Note how the diagonals wrap around the tile ensuring that all pairwise interactions are handled. In this scheme there is no race condition since every thread stores to different $\boldsymbol{f}_i$ and $\boldsymbol{f}_j$ variables. Within the 32 × 32 tile some pairwise forces are excluded based on the force field definition (such as atoms sharing a bond or an angle), the end of the atom list (when the patch has non-modulo 32 atoms), and the neighbor list cut-off radius. In addition, one half of interactions are excluded on "self" tiles where the two sets of atoms are the same. Exclusions tests are performed efficiently using an array of 32 × 32-bit unsigned int variables for each tile to keep track of the exclusions, where a 0-bit means "exclude" and 1-bit means "compute". The exclusion bit masks are created during the neighbor list build using a compressed lookup table [7]. For each of the 32 iterations through the tile, we must shift the input (atom coordinates, charges, etc.) and output (force) variables between threads within the warp. We use the `_shfl()` instructions first introduced in the Kepler GPU architecture to do this shifting with warp-synchronous programming entirely in GPU registers, thereby eliminating the need for shared memory and the additional synchronization operations it requires.

After the 32 × 32 tile is computed, forces $\boldsymbol{f}_j$ are stored in a global memory buffer using the `atomicAdd()` instruction and the kernel proceeds to the next tile with different j atoms but the same i atoms. Keeping i atoms constant reduces the number of memory accesses since for each tile only the j atoms must be loaded from global memory. After the warp finishes all tiles it was assigned, forces $\boldsymbol{f}_i$ are stored in global memory using the `atomicAdd()` instruction.

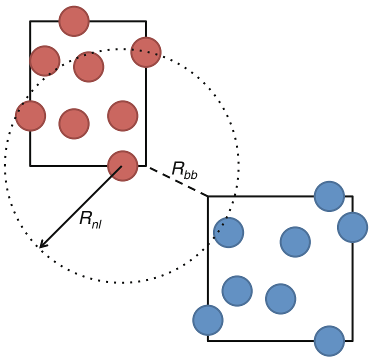
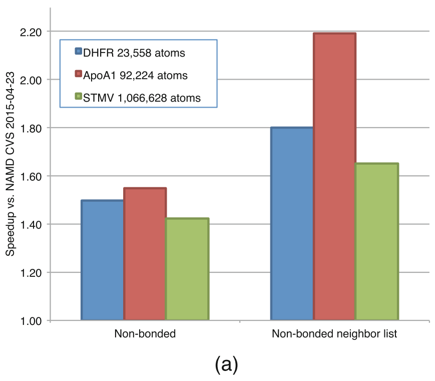


Fig. 4. Bounding boxes for two sets of eight atoms. Bounding box distance R_{bb} is well within the neighbor list cut-off distance R_{nl}, however none of the atoms are within the neighbor list cut-off.

In the previous version of NAMD, the neighbor list was built by the non-bonded force kernel by checking all atom pair distances in every 32×32 tile. We implemented a more efficient neighbor list builder that first builds an approximate neighbor list based on the bounding boxes of sets of 32 atoms, and then using that approximate list as input, builds the atom-based neighbor list in the non-bonded force kernel. The bounding-box approximation always overestimates the true neighbor list, as is illustrated in Fig. 4 where, although the bounding box distance R_{bb} is well within the neighbor list cut-off distance R_{nl}, none of the atoms are actually within the neighbor list cut-off. In the situation illustrated in Fig. 4, the bounding-box neighbor list would contain the two sets of atoms, but this entry would be pruned away in the atom-based distance check in the non-bonded force kernel.

We have also implemented the neighbor list for the Generalized Born implicit solvent (GBIS) kernels. Since the GBIS method does not use force-field exclusions and includes self-energies (explicit solvent calculations do not), a separate GBIS version of the neighbor list must be created. The bounding-box neighbor lists for both the GBIS and explicit solvent calculations are the same. During the atom-based neighbor list build in the non-bonded force kernel, we keep track of both the regular and the GBIS tile lists by assigning their indicators to the lower and upper 16-bits of a 32-bit integer, respectively. The two lists are finally separated in the neighbor list sorting step where the GBIS list is produced by choosing the upper bits, and the regular list by choosing the lower bits.

Benchmarks of the non-bonded force kernel were performed on a single Tesla K40m GPU of the Crest system for explicit solvent and GBIS systems. The explicit solvent systems included DHFR with 23,588 atoms, ApoA1 with 92,224 atoms, and STMV with 1,066,628 atoms, while the implicit solvent systems included GP (rabbit muscle glycogen phosphorylase, PDB ID 2GJ4) with 13,340 atoms and an empty HIV capsid with 5,662,344 atoms. For the explicit solvent cases, the benchmarks were performed using the non-streaming versions



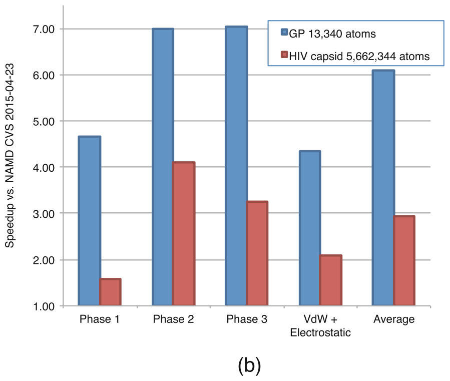


Fig. 5. Speedup of non-bonded force kernels vs. NAMD CVS version 2015-04-23 for (a) explicit solvent regular and neighbor list versions and (b) Generalized Born implicit solvent kernels.

of the kernels and we disabled the GPU PME reciprocal computation (NAMD script command `PMEOffload no`) to ensure that it did not interfere with the non-bonded force kernel timings. The kernel timings were obtained from the average kernel run time reported by the CUDA `nvprof` profiler.

Figure 5 shows the speedup observed for both explicit and implicit solvent cases. In the explicit solvent case shown in Fig. 5(a) we observe 1.4× to 1.6× speedup for the regular non-bonded force kernel and 1.7× to 2.2× speedup for the non-bonded neighbor list builder kernel. The neighbor list version gets a larger speedup due to the use of the bounding box neighbor list approximation, which reduces the number of tiles that must be checked at atom level.

In the case of explicit solvent, shown in Fig. 5(b), we observe much larger speedups: 6× speedup for the smaller GP system and 3× speedup for the HIV capsid system. Figure 5(b) also shows the breakdown of speedups for the three phases of the GBIS computation as well as the Van der Waals (VdW) and electrostatic parts.

3.2 PME Reciprocal Force Computation

NAMD uses the smooth particle mesh Ewald version [12] of the famous particle mesh Ewald (PME) method [13]. The PME method consists of five main parts: (1) spread charges on to grid, (2) 3-D Fast Fourier Transform (FFT) from direct to reciprocal space, (3) solve Poisson equation on the grid in the reciprocal space, (4) 3-D FFT back from reciprocal to direct space, and (5) gather atom forces from the direct space grid. In the previous version of NAMD, only charge spreading (1) and force gathering (5) were performed on the GPU while the 3-D FFT and Poisson solve were performed on the host CPUs. In the new version, we moved all of the PME computation to the GPU and changed the way the charge spreading and force gathering is done, as detailed below. We implemented a multi-GPU 3-D FFT solver that uses a pencil, slab, or box decomposition depending on the

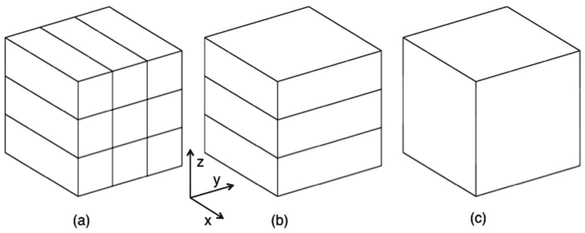


Fig. 6. PME (a) pencil, (b) slab, and (c) box decompositions for the direct to reciprocal space FFT. A single GPU is assigned to each pencil, slab, or box, respectively.

number of GPUs in use. These decompositions are illustrated in Fig. 6 for the direct to reciprocal space FFT step. We assign a single GPU for each pencil, slab or box. This means that we would use nine GPUs for the pencil decomposition in Fig. 6(a), three GPUs for the slab decomposition in Fig. 6(b), and a single GPU for the box decomposition in Fig. 6(c).

For the pencil decomposition in Fig. 6(a), we perform 1-D FFTs in x, y, and z directions in order to compute the full 3-D FFT. After each 1-D FFT, we need to 3-D transpose the grid such that the current FFT direction is contiguous in memory. For example, after 1-D FFT in the x direction, we transpose the grid to (y, z, x) order and then proceed with the 1-D FFT in the y direction. For the slab decomposition in Fig. 6(b), we perform 2-D FFT in a xy slab, and then 1-D FFT in the z direction. For the box decomposition in Fig. 6(c), we simply perform a single 3-D FFT on the entire grid. The 1-D, 2-D, and 3-D FFTs are computed using NVIDIA's cuFFT library. For the reciprocal to direct space FFT, we perform the same steps in reverse. For example in the case of PME pencil decomposition, we would first perform 1-D FFT in the z direction, then in the y direction, and finally in the x direction. The 3-D transposes are now performed in the reverse order as well.

The 3-D transpose involves communication between GPUs and can therefore become the most time consuming part of the PME reciprocal computation. For communication within a Crest node, we use the CUDA `cudaMemcpyPeerAsync()` function. We were not able to use direct GPU-to-GPU (known as peer-to-peer) communication since this is not currently enabled on the IBM POWER8 platform. We expect future NVLink GPU interconnect technology to enable direct GPU-to-GPU transfers and to make communication between GPUs on the same node much faster. For communication between nodes, we copy the GPU memory to a CPU buffer and then use the Charm++ communication layer to send the buffer to the receiving node. On the receiving node, we then copy the CPU buffer to the GPU. Future versions of Charm++ will hopefully enable direct GPU-to-GPU communication between nodes on capable hardware.

As mentioned earlier, we also changed the way the charges are spread on the grid. In the previous version of NAMD, charges on an atom patch were spread into a sub-grid and then the sub-grids were communicated among the PEs to form the PME pencils (with similar operation for PME slabs and boxes).

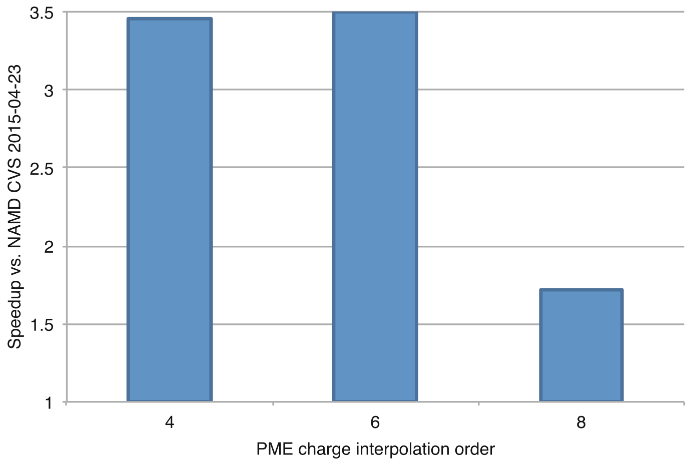


Fig. 7. Speedup for PME force computation vs. NAMD CVS version 2015-04-23 for DHFR system with 23,588 atoms and PME grid of size $64 \times 64 \times 64$. Benchmarks were run on a single node of the Titan supercomputer.

We changed the charge spreading such that instead of communicating grid points, we communicate the charges to the GPU that needs them. The GPU will then spread the charges and start working on the 3-D FFT. Note that since atoms near neighboring PME pencils boundaries spread to multiple pencils, we need to communicate a halo of atoms of the width of the PME interpolation order to obtain correct results.

Similarly to charge spreading, force gathering was changed to communicate forces instead of grid points. In detail, force gathering is now done on the GPU for all atoms that are within its PME pencil. Since atoms share neighboring PME pencils, some of the gathered forces will be partial at this stage. Therefore, after force gathering, the GPU communicates the partial forces to the CPU, and then the CPU combines the forces from multiple neighboring pencils to form the total PME reciprocal force acting on each atom.

In particular for small systems, where there is only a single GPU performing the PME reciprocal computation (i.e. box decomposition), the GPU can perform the entire PME reciprocal force computation on its own. This greatly improves PME reciprocal computation for single GPU runs, as is shown in Fig. 7.

3.3 NAMD Performance on Crest Versus Titan

In Fig. 8 we compare NAMD performance on a single node of Crest versus four nodes of Titan. Based on the difference in the underlying GPU hardware performance, K40m on Crest and K20X on Titan, one would expect about 7 % better kernel performance on Crest (for K40m without GPU Boost since users are not allowed to change the GPU settings on the Crest cluster). We see from Fig. 8 that the non-bonded force kernel performance on Crest is always somewhat less

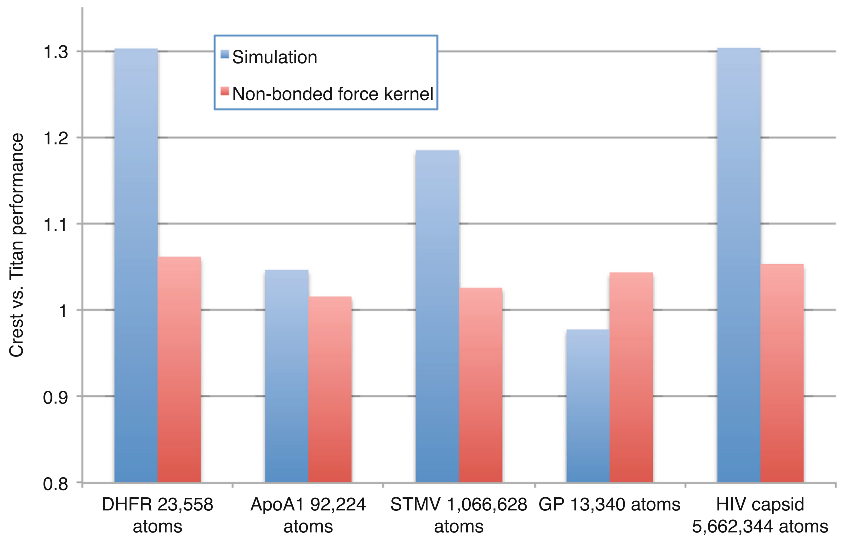


Fig. 8. Relative performance of a single Crest node vs. four nodes of Titan.

than the theoretical maximum, which is to be expected. We also see that Crest runs some of the simulations up to 30 % faster than Titan, while others (GP) actually have lower performance on Crest. Further work is necessary to see why the performance on Crest sometimes dips below the performance on Titan.

4 VMD: Simulation Preparation and Analysis

VMD [8] is a popular tool for MD simulation preparation, analysis, and visualization. In concert with a sophisticated atom selection language and a wide variety of data structures and algorithms for visualization, analysis, and structure manipulation, VMD incorporates built-in Tcl and Python scripting that can be used to perform large scale molecular modeling and analysis tasks in parallel on clouds [14], clusters, and petascale computers [9,10,15–17]. VMD provides a wide variety of tools for assembling large macromolecular complexes from constituent proteins, and combining these with solvent and ions to replicate biological conditions in vivo, and it can emit the completed system for simulation with popular GPU-accelerated MD simulation tools such as NAMD [6,7,18] and GROMACS [19,20]. The performance of VMD for a user's molecular modeling and visualization work is not well-characterized by any single kernel or time-critical loop, so we report performance for a variety of algorithms that capture key characteristics of VMD workloads.

4.1 Compiler Performance Comparisons

To evaluate the merits of GCC and XLC for POWER8 CPUs, we compared the performance of exemplary performance-critical loops from internal

Table 1. Performance of several inner loops associated with VMD analytical calculations, including atom selection array processing loops, a loop over 8-element inner products associated with multilevel summation electrostatics, and 1-D and 3-D minimum/maximum bounds loop. Each loop was compiled using GCC and XLC. We report GCC performance data for the `-mvsx` VSX vectorization option, and hand-vectorized loops written with VSX vector intrinsics, e.g. `vec_madd()`, where applicable. All XLC results were obtained with VSX instructions enabled since it did not harm performance.

Compiler	Atomsel cnt	Atomsel first	MSM-dot8	1-D min-max	3-D min-max
GCC -mvsx	0.123 s	0.061 s	0.571 s	1.163 s	0.649 s
GCC intrinsics	0.123 s			0.101 s	
XLC	0.082 s	0.038 s	0.548 s	0.094 s	0.106 s
XLC intrinsics	0.082 s			0.094 s	

VMD algorithms. Table 1 summarizes results for the compiler tests. We note that GCC exhibited surprisingly poor performance with `-mvsx` enabled for the 1-D min-max test case, but that when provided with hand-written vectorized loops using VSX intrinsics, the generated code approached the performance of XLC. We found that the code generated by XLC ran at the same speed as the hand-coded VSX loops. Our general experience during porting of VMD was that XLC provided higher overall performance on average, but particularly for cases with significant opportunity to exploit VSX instructions. As such, all subsequent VMD performance measurements were performed using XLC.

4.2 Tachyon Ray Tracing Engine

VMD incorporates the Tachyon ray tracing (RT) engine for high quality rendering on all supported platforms, including clouds, clusters, and supercomputers [9,21]. The rise of ray tracing for high-fidelity scientific visualization has led to the development of hardware-optimized RT libraries and frameworks [22,23], in combination with SPMD languages and compilers such as CUDA [24] and ISPC [25] that generate highly-optimized code for wide SIMD vector units. VMD contains a GPU-accelerated RT engine [10,15,26], however OptiX [22] is not yet available for OpenPOWER platforms, so Tachyon is currently the highest performance VMD RT engine on OpenPOWER. Figure 9 shows a visualization of the HIV-1 capsid with host cell factor cyclophilin A (CypA) [2], docked with a cryo-electron density map of the nuclear pore complex, which was used as a representative test case for Tachyon on multi-core CPUs.

Tachyon is an interesting performance test case since it is widely used for high quality VMD figure and movie renderings as a result of its speed and availability on all platforms supported by VMD. Even on platforms that support the GPU-accelerated TachyonL-OptiX [10,15,26] RT engine, it is still occasionally necessary to use the CPU-based Tachyon RT engine for scenes that greatly exceed GPU physical memory capacity. Tachyon uses a variety of linked lists

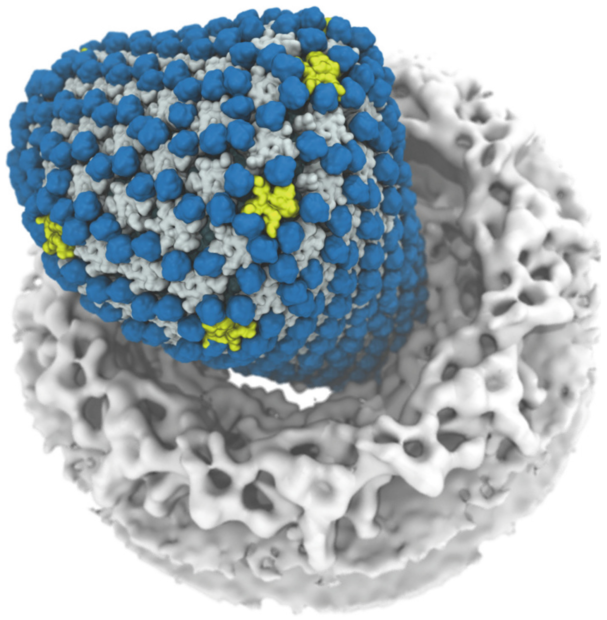
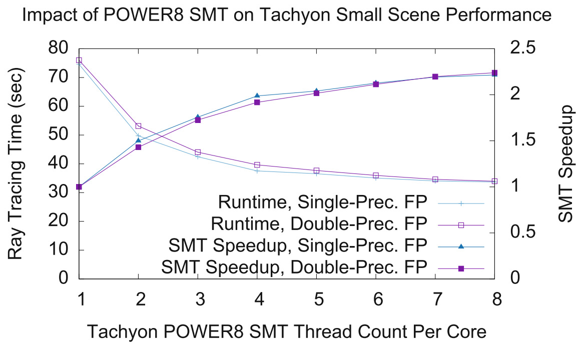


Fig. 9. Visualization of the HIV-1 capsid with pentamers highlighted in yellow, host cell factor cyclophilin A (CypA) shown in blue [2], docked with a cryo-electron density map of the nuclear pore complex shown in white. The scene, incorporating direct lighting, ambient occlusion lighting, and depth of field with large sample counts, was used to measure Tachyon ray tracing performance on the POWER8 CPU, as shown in Fig. 11. (Color figure online)

and hierarchical data structures that share some performance characteristics with molecular structure traversal algorithms in other areas of VMD. Tachyon currently does not make explicit use of POWER8 VSX vector instructions, and present-day compilers and languages are not capable of effectively autovectorizing RT algorithms, so effective use of SMT is of particular interest at present.

Tachyon CPU RT performance was measured for two very different test scenes and execution conditions, with varying counts of SMT threads per core. In the first case, shown in Fig. 10, a standalone Tachyon build was run on a simple scene containing a short DNA segment, but with very high stochastic sample counts, thereby creating a workload that emphasized floating point arithmetic and CPU cache performance. The DNA scene is small enough to expect that all performance-critical Tachyon data structures would be cached entirely on each CPU, and that NUMA-related performance effects would be minimized. Figure 10 shows that for the small DNA scene, the use of SMT is beneficial for performance all the way up to 8 threads per CPU core, or 160 threads in total, yielding a peak speedup 2.2× the performance achieved using a single thread per core.



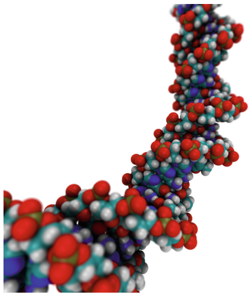


Fig. 10. Evaluation of the impact of varying per core SMT thread counts on Tachyon small scene ray tracing performance on POWER8.

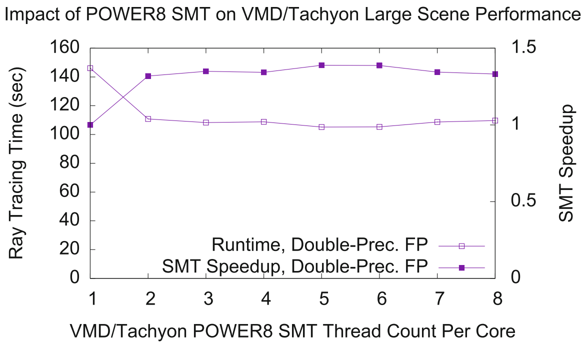


Fig. 11. Evaluation of the impact of varying per core SMT thread counts on VMD/Tachyon ray tracing performance on POWER8, for the HIV-1 scene in Fig. 9.

In the second test case, shown in Figs. 9 and 11, the Tachyon engine built into VMD was used to render a large scale geometrically complex scene of the HIV-1 capsid and host cell factor cyclophilin A docked with a cryo-electron density map of the nuclear pore complex [2]. The HIV-1 scene is representative of routine visualizations created from ongoing petascale molecular dynamics simulations. Due to the large size of the HIV-1 scene and the fact that Tachyon is linked into VMD, the impact of NUMA locality and inter-processor SMP bus bandwidth are much greater. Since Tachyon inherits the existing distribution of VMD memory allocations among NUMA nodes and the HIV-1 scene size is very large, Tachyon's runtime memory allocations are less uniformly distributed than when Tachyon is run standalone, and this also leads to increased run-to-run performance variation. As a result of these effects, the maximum SMT performance observed for the HIV-1 test case was roughly 1.3× that of using a single thread per CPU core. Table 2 summarizes performance results obtained for varying degrees of SMT threading for the large HIV-1 test case with double-precision floating point arithmetic on POWER8 and Intel Xeon E5-2660v3. The POWER8

Table 2. VMD/Tachyon HIV-1 double precision CPU ray tracing performance for varying numbers of SMT threads per core. The table shown here provides numeric values for the plot in Fig. 11, and adds performance results for the Xeon E5-2660v3.

SMT Count	POWER8			Xeon E5-2660v3		
	Threads	DNA	HIV-1	Threads	DNA	HIV-1
1	20	76.0 s	143.1 s	20	63.7 s	104.9 s
2	40	53.1 s	110.9 s	40	48.4 s	87.1 s
4	80	39.6 s	108.9 s			
8	160	33.9 s	109.8 s			

system outperformed the Xeon by 1.43× for the (cache-friendly, arithmetic bound) DNA scene rendered standalone, the Xeon outperforms POWER8 by 1.26× for the large HIV-1 scene rendered by Tachyon within VMD.

4.3 Molecular Dynamics Flexible Fitting Cross Correlation

Molecular dynamics flexible fitting combines molecular structure data from cryo-electron microscopy and X-ray crystallography with molecular dynamics simulations, to determine all-atom structures of large biomolecular complexes such as the HIV-1 capsid [1,2]. The calculation of quality-of-fit for structures obtained from hybrid fitting approaches is computationally demanding, and particularly when run on tens of thousands of MDFF trajectory frames and when multiple structural conformations need to be evaluated. To address this challenge, VMD implements fast algorithms for computing quality-of-fit between all-atom structures and experimental cryo-electron density maps using hand-coded CPU SIMD vector instructions and data-parallel CUDA kernels on NVIDIA GPUs [16].

Table 3 reports cross correlation performance for a single trajectory frame from a large rabbit hemorrhagic disease virus (RHDV) capsid test case [16,27]. The reported cross correlation performance results were obtained using density map simulation algorithms implemented using hand-coded POWER8 VSX and Intel Xeon E5-2660v3 SSE vector intrinsics. The POWER8+Tesla K40m result demonstrates performance closely approaching the Xeon E5-2687W+Quadro K6000 result previously reported [16]. The closely comparable GPU performance results are expected since the two GPUs share the same architecture but the Quadro K6000 has a core clock rate that is roughly 20 % higher.

4.4 Molecular Orbital Calculation

Calculation and display of molecular orbitals (MOs), electron densities, and molecular electrostatic potentials are helpful steps in the analysis of quantum chemistry calculations. The key challenge involved in the calculation and display of MOs and related quantities is the rapid evaluation of complex wavefunctions on a three-dimensional lattice; the resulting data can then be used for plotting

Table 3. VMD molecular dynamics flexible fitting (MDFF) quality-of-fit cross correlation analysis performance for the rabbit hemorrhagic disease virus (RHDV) capsid solved with a 6.5 Å resolution density map [16,27]. On POWER8 platforms, hand-coded VSX vector instructions are used, providing a 20 % overall performance gain relative to xlC autovectorization. The use of SMT has negligible performance benefit on POWER8. On the Intel platforms hand-coded SSE vector instructions are used, yielding a 12 % performance gain compared to the best autovectorization achieved by the Intel XE 2015 compiler. The difference between the GPU-accelerated performance results is likely attributable solely to the higher Quadro K6000 GPU clock rate.

SMT Count	POWER8 hand-coded VSX		Xeon E5-2660v3 hand-coded SSE		POWER8 + Tesla K40m		Xeon E5-2687W + Quadro K6000	
	Threads	CC time	Threads	CC time	GPUs	CC time	GPUs	CC time
1	20	1.354 s	20	0.922 s	1	0.488 s	1	0.458 s
2	40	1.345 s	40	0.905 s				
4	80	1.334 s						
8	160	1.364 s						

isocontours or isosurfaces as shown in Fig. 12, and for other analyses. VMD contains data-parallel CPU and GPU kernels for computing MOs on Intel x86 CPUs optionally paired with NVIDIA GPUs [28,29], and on a variety of ARM SoCs and CPUs paired with integrated (on-chip) or discrete (add-in board) GPUs [30].

VMD evaluates MOs on a 3-D lattice by decomposing the lattice points into 2-D planes which are computed independently by different CPU threads or different GPUs. The workload is dynamically scheduled across the pool of workers to balance load on hardware of varying capability and in the presence of external load or operating system interference. Table 4 lists performance results for the VMD CPU and GPU-accelerated molecular orbital kernels run on a C_{60} test case. While the use of POWER8 SMT was beneficial for the plain

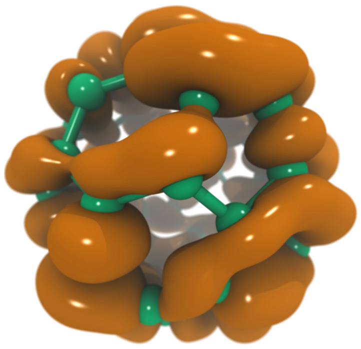


Fig. 12. VMD rendering of the atomic structure and molecular orbitals for a vibrating C_{60} simulation produced by Terachem.

Table 4. VMD C_{60} molecular orbital kernel performance for varying grid sizes, CPU thread counts and SMT utilization, vectorization approach, and GPU counts. Execution times are reported in seconds. Speedups within each platform are shown normalized to the base test case for each platform: vs. plain C++ on POWER8 CPU, vs. plain C++ on Xeon CPU, and vs. a single Tesla K40m GPU. The 4-GPU run achieved performance about 9× faster than the best POWER8 hand-coded VSX CPU results for both tests, and 5× and 8× faster than the Xeon E5-2660v3 hand-coded SSE results for medium and high resolution grid sizes, respectively. POWER8 SMT threading increases performance by up to 1.85× for the non-vectorized plain C++ implementation, but it had no measurable impact on the hand-coded VSX implementation.

C_{60} Grid Size	POWER8 160 Threads		Xeon E5-2660v3 40 Threads		POWER8 + Tesla K40m GPUs		
	C++	VSX	C++	SSE	1	2	4
172 × 173 × 169	1.09 s	0.52 s	2.42 s	0.301 s	0.190 s	0.099 s	0.058 s
	1.0×	2.10×	1.0×	8.01×	1.0×	1.91×	3.27×
516 × 519 × 507	17.57 s	8.03 s	59.18 s	7.14 s	3.49 s	1.76 s	0.91 s
	1.0×	2.18×	1.0×	8.28×	1.0×	1.98×	3.83×

C++ algorithm, the lack of impact on the hand-vectorized implementation likely indicates that it already effectively utilizes all of the POWER8 arithmetic units even with only 1 thread per core. The 4-way Tesla K40m performance result for high resolution test case closely matches performance results obtained on the same GPU hardware on Intel x86 systems. The 4-GPU result outperforms the POWER8 CPU results using hand-coded VSX instructions by a factor of 9× and it runs 5× to 8× faster than the Xeon E5-2660v3 CPU results using hand-coded SSE instructions. The hand-coded VSX and SSE MO kernels for POWER8 and Intel x86 hardware outperform the best autovectorized C++ loops in each case by 2× and 8× respectively. We note that it may be possible to further increase the performance of the hand-vectorized POWER8 VSX loop with additional tuning work.

5 Conclusions and Future Work

We have presented the adaptation of NAMD and VMD to GPU-accelerated POWER8 platforms and we have reported performance results for a variety of algorithms and test cases, and we have provided results for relevent comparison platforms. The CPU-focused test results presented here demonstrate the potential for POWER8 SMT to allow non-vectorized code to better utilize CPU functional units. Conversely, we observe that fully-vectorized code does not appear to benefit from SMT. Further developments that make use of explicit NUMA-aware memory allocations from private memory arenas should improve VMD-integrated Tachyon RT performance for large scenes such as HIV-1 by mitigating the impact of pre-existing memory fragmentation and NUMA locality. We noted that the lack of horizontal-add and other vector instructions found on competing CPUs made development of some VSX kernels more challenging than initially expected, but that substantial performance gains were still possible relative to the best compiler-generated code.

The ORNL Crest development system models key attributes of the future Summit supercomputer. The future Summit system will provide larger memory capacity, faster POWER9 CPUs and Volta GPUs, and higher performance intra-node communication paths between the CPUs and GPUs by virtue of the future NVLink interconnect. While the Crest system provides a programming environment representative of the future Summit system, not all of the desired capabilities of Summit are available yet on Crest. At the time of writing, it was not yet possible to evaluate the performance benefits associated with peer-to-peer GPU transfers as they are not yet implemented by CUDA 7.5 for OpenPOWER. Several GPU-accelerated graphics-related features of VMD were similarly unavailable for testing due to lack of OpenPOWER versions of the OpenGL or Vulkan APIs for rasterization, and the associated GLX or EGL context management APIs [14], the OptiX GPU ray tracing framework [22], or the NVENC GPU-accelerated video encoding library.

Acknowledgments. The authors acknowledge support from NIH grants 9P41GM104601 and 5R01GM098243-02, the CUDA Center of Excellence at the University of Illinois, the Blue Waters sustained-petascale computing project supported by NSF awards OCI-0725070 and ACI-1238993, the state of Illinois, "The Computational Microscope" NSF PRAC awards OCI-0832673 and ACI-1440026, and the Oak Ridge Leadership Computing Facility at Oak Ridge National Laboratory supported by the Department of Energy under Contract DE-AC05-00OR22725.

References

1. Zhao, G., Perilla, J.R., Yufenyuy, E.L., Meng, X., Chen, B., Ning, J., Ahn, J., Gronenborn, A.M., Schulten, K., Aiken, C., Zhang, P.: Mature HIV-1 capsid structure by cryo-electron microscopy and all-atom molecular dynamics. Nature **497**, 643–646 (2013)
2. Liu, C., Perilla, J.R., Ning, J., Lu, M., Hou, G., Ramalho, R., Bedwell, G., Byeon, I.J., Ahn, J., Shi, J., Gronenborn, A., Prevelige, P., Rousso, I., Aiken, C., Polenova, T., Schulten, K., Zhang, P.: Cyclophilin A stabilizes HIV-1 capsid through a novel non-canonical binding site. Nat. Commun. **7**, Article no. 10714, 10 pages (2016)
3. Sothiselvam, S., Liu, B., Han, W., Klepacki, D., Atkinson, G.C., Brauer, A., Remm, M., Tenson, T., Schulten, K., Vázquez-Laslop, N., Mankin, A.S.: Macrolide antibiotics allosterically predispose the ribosome for translation arrest. Proc. Natl. Acad. Sci. USA **111**, 9804–9809 (2014)
4. Mendes, C.L., Bode, B., Bauer, G.H., Enos, J., Beldica, C., Kramer, W.T.: Deploying a large petascale system: the Blue Waters experience. Procedia Comput. Sci. **29**, 198–209 (2014)
5. Joubert, W., Archibald, R., Berrill, M., Brown, W.M., Eisenbach, M., Grout, R., Larkin, J., Levesque, J., Messer, B., Norman, M., Philip, B., Sankaran, R., Tharrington, A., Turner, J.: Accelerated application development: the ORNL Titan experience. Comput. Electr. Eng. **46**, 123–138 (2015)

6. Phillips, J.C., Braun, R., Wang, W., Gumbart, J., Tajkhorshid, E., Villa, E., Chipot, C., Skeel, R.D., Kale, L., Schulten, K.: Scalable molecular dynamics with NAMD. J. Comp. Chem. **26**, 1781–1802 (2005)
7. Phillips, J.C., Stone, J.E., Schulten, K.: Adapting a message-driven parallel application to GPU-accelerated clusters. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC 2008, 9 pages. IEEE Press, Piscataway, NJ, USA (2008)
8. Humphrey, W., Dalke, A., Schulten, K.: VMD - visual molecular dynamics. J. Mol. Graph. **14**, 33–38 (1996)
9. Stone, J.E., Isralewitz, B., Schulten, K.: Early experiences scaling VMD molecular visualization and analysis jobs on Blue Waters. In: Extreme Scaling Workshop (XSW 2013), pp. 43–50 (2013)
10. Stone, J.E., Sener, M., Vandivort, K.L., Barragan, A., Singharoy, A., Teo, I., Ribeiro, J.V., Isralewitz, B., Liu, B., Goh, B.C., Phillips, J.C., MacGregor-Chatwin, C., Johnson, M.P., Kourkoutis, L.F., Hunter, C.N., Schulten, K.: Atomic detail visualization of photosynthetic membranes with GPU-accelerated ray tracing. Parallel Comput. **55**, 17–27 (2016)
11. Götz, A.W., Williamson, M.J., Xu, D., Poole, D., Grand, S.L., Walker, R.C.: Routine microsecond molecular dynamics simulations with AMBER on GPUs. 1. Generalized Born. J. Chem. Theory Comput. **8**, 1542–1555 (2012)
12. Essmann, U., Perera, L., Berkowitz, M.L., Darden, T., Lee, H., Pedersen, L.G.: A smooth particle mesh Ewald method. J. Chem. Phys. **103**, 8577–8593 (1995)
13. Darden, T., York, D., Pedersen, L.: Particle mesh Ewald: an N·log(N) method for Ewald sums in large systems. J. Chem. Phys. **98**, 10089–10092 (1993)
14. Stone, J.E., Messmer, P., Sisneros, R., Schulten, K.: High performance molecular visualization: In-situ and parallel rendering with EGL. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW) (2016, in Press)
15. Stone, J.E., Vandivort, K.L., Schulten, K.: GPU-accelerated molecular visualization on petascale supercomputing platforms. In: Proceedings of the 8th International Workshop on Ultrascale Visualization. UltraVis 2013, pp. 6:1–6:8. ACM, New York (2013)
16. Stone, J.E., McGreevy, R., Isralewitz, B., Schulten, K.: GPU-accelerated analysis and visualization of large structures solved by molecular dynamics flexible fitting. Faraday Discuss. **169**, 265–283 (2014)
17. Phillips, J.C., Stone, J.E., Vandivort, K.L., Armstrong, T.G., Wozniak, J.M., Wilde, M., Schulten, K.: Petascale Tcl with NAMD, VMD, and Swift/T. In: Workshop on High Performance Technical Computing in Dynamic Languages, SC 2014, pp. 6–17. IEEE Press (2014)
18. Ribeiro, J.V., Bernardi, R.C., Rudack, T., Stone, J.E., Phillips, J.C., Freddolino, P.L., Schulten, K.: QwikMD-integrative molecular dynamics toolkit for novices and experts. Sci. Rep. **6**, 26536 (2016)
19. Pronk, S., Páll, S., Schulz, R., Larsson, P., Bjelkmar, P., Apostolov, R., Shirts, M.R., Smith, J.C., Kasson, P.M., van der Spoel, D., Hess, B., Lindahl, E.: Gromacs 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. Bioinformatics **29**, 845–854 (2013)
20. Vermaas, J.V., Hardy, D.J., Stone, J.E., Tajkhorshid, E., Kohlmeyer, A.: TopoGromacs: automated topology conversion from CHARMM to GROMACS within VMD. J. Chem. Inf. Model. (2016, in Press)
21. Stone, J.E.: An efficient library for parallel ray tracing and animation. Master's thesis, Computer Science Department, University of Missouri-Rolla (1998)

22. Parker, S.G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., Stich, M.: OptiX: a general purpose ray tracing engine. In: ACM SIGGRAPH 2010 papers, SIGGRAPH 2010, pp. 66:1–66:13. ACM, New York (2010)
23. Wald, I., Woop, S., Benthin, C., Johnson, G.S., Ernst, M.: Embree: a kernel framework for efficient CPU ray tracing. ACM Trans. Graph. **33**, 143:1–143:8 (2014)
24. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with CUDA. ACM Queue **6**, 40–53 (2008)
25. Pharr, M., Mark, W.: ispc: A SPMD compiler for high-performance CPU programming. In: Innovative Parallel Computing (InPar 2012), pp. 1–13 (2012)
26. Stone, J.E., Sherman, W.R., Schulten, K.: Immersive molecular visualization with omnidirectional stereoscopic ray tracing and remote rendering. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW) (2016, in Press)
27. Wang, X., Xu, F., Liu, J., Gao, B., Liu, Y., Zhai, Y., Ma, J., Zhang, K., Baker, T.S., Schulten, K., Zheng, D., Pang, H., Sun, F.: Atomic model of rabbit hemorrhagic disease virus by cryo-electron microscopy and crystallography. PLoS Pathog. **9**, e1003132 (2013). (14 pages)
28. Stone, J.E., Saam, J., Hardy, D.J., Vandivort, K.L., Hwu, W.W., Schulten, K.: High performance computation and interactive display of molecular orbitals on GPUs and multi-core CPUs. In: Proceedings of the 2nd Workshop on General-Purpose Processing on Graphics Processing Units, ACM International Conference Proceeding Series, vol. 383, pp. 9–18. ACM, New York (2009)
29. Stone, J.E., Hardy, D.J., Saam, J., Vandivort, K.L., Schulten, K.: GPU-accelerated computation and interactive display of molecular orbitals. In: Hwu, W. (ed.) GPU Computing Gems, pp. 5–18. Morgan Kaufmann Publishers, San Francisco (2011)
30. Stone, J.E., Hallock, M.J., Phillips, J.C., Peterson, J.R., Luthey-Schulten, Z., Schulten, K.: Evaluation of emerging energy-efficient heterogeneous computing platforms for biomolecular and cellular simulation workloads. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW) (2016, in Press)