SNAP: Strong Scaling High Fidelity Molecular Dynamics Simulations on Leadership-Class Computing Platforms

Christian R. Trott, Simon D. Hammond, and Aidan P. Thompson

Center for Computing Research,
Sandia National Laboratories, Albuquerque NM 87185, USA,
crtrott@sandia.gov

Abstract. The rapidly improving compute capability of contemporary processors and accelerators is providing the opportunity for significant increases in the accuracy and fidelity of scientific calculations. In this paper we present performance studies of a new molecular dynamics (MD) potential called SNAP.

The SNAP potential has shown great promise in accurately reproducing physics and chemistry not described by simpler potentials. We have developed new algorithms to exploit high single-node concurrency provided by three different classes of machine: the Titan GPU-based system operated by Oak Ridge National Laboratory, the combined Sequoia and Vulcan BlueGene/Q machines located at Lawrence Livermore National Laboratory, and the large-scale Intel Sandy Bridge system, Chama, located at Sandia.

Our analysis focuses on strong scaling experiments with approximately 246,000 atoms over the range 1−122,880 nodes on Sequoia/Vulcan and 40−18,630 nodes on Titan. We compare these machine in terms of both simulation rate and power efficiency. We find that node performance correlates with power consumption across the range of machines, except for the case of extreme strong scaling, where more powerful compute nodes show greater efficiency.

This study is a unique assessment of a challenging, scientifically relevant calculation running on several of the world's leading contemporary production supercomputing platforms.

1 Introduction

Classical molecular dynamics simulation (MD) is a powerful approach for describing the mechanical, chemical, and thermodynamic behavior of solid and fluid materials in a rigorous manner [5]. The material is modeled as a large collection of point masses (atoms) whose motion is tracked by integrating the classical equations of motion to obtain the positions and velocities of the atoms at a large number of timesteps. The forces on the atoms are specified by an inter-atomic potential that defines the potential energy of the system as a function of the atom positions. Typical inter-atomic potentials are computationally

inexpensive and capture the basic physics of electron-mediated atomic interactions of important classes of materials, such as molecular liquids and crystalline metals. Efficient MD codes running on commodity workstations are commonly used to simulate systems with $N = 10^5 - 10^6$ atoms, the scale at which many interesting physical and chemical phenomena emerge. For a few select physics applications, much larger atom counts are required, and these applications have historically been successfully run on leadership platforms [3, 6, 7, 11]. Quantum molecular dynamics (QMD) is a much more computationally intensive method for solving a similar physics problem [9]. Instead of assuming a fixed interatomic potential, the forces on atoms are obtained by explicitly solving the quantum electronic structure of the valence electrons at each timestep. Because MD potentials are short-ranged, the computational complexity of MD generally scales as $O(N)$, whereas QMD calculations require global self-consistent convergence of the electronic structure, whose computational cost is $O(N_e^\alpha)$, where $\alpha = 2-3$ and N_e is the number of electrons. For the same reasons, MD is amenable to spatial decomposition on parallel computers, while QMD calculations allow only limited parallelism.

As a result, while high accuracy QMD simulations have supplanted MD in the range $N = 10 - 100$ atoms, QMD is still intractable for $N > 1000$, even using the largest supercomputers. Conversely, typical MD potentials often exhibit behavior that is inconsistent with QMD simulations. This has led to great interest in the development of MD potentials that match the QMD results for small systems, but can still be scaled to the interesting regime $N = 10^5 - 10^6$ atoms. These quantum-accurate potentials require many more floating point operations per atom compared to conventional potentials, but they are still short-ranged. So the computational cost remains $O(N)$, but with a larger algorithm pre-factor. This presents both opportunities and challenges in the context of Petascale computing. On the one hand, achieving good single-node performance is more difficult; on the other hand, the high compute-to-communication ratio implies enhanced strong scaling, relative to simpler potentials. In recent years Quantum Chemistry methods with $O(N)$ scaling have been developed. Similar to the quantum-accurate classical potentials, they allow the use of leadership class platforms for small to medium sized problems [4, 13]. Nevertheless, the algorithm pre-factor in these methods is much larger than that of quantum-accurate potentials, making their use for MD simulation infeasible.

In this paper, we focus our attention on a new quantum-accurate potential called SNAP. It has been used to model the shear-migration of screw dislocations in tantalum metal, the fundamental process underlying plastic deformation in body-centered cubic metals [12]. Unlike simpler potentials the SNAP potential for tantalum correctly matches QMD results for the screw dislocation core structure and minimum energy pathway for displacement of this structure. Combining the accuracy of SNAP with efficient parallel algorithms that work on Petascale computers has opened the door to truly predictive first principles simulations of materials behavior at an unprecedented scale. The contributions in this paper are: (1) an analysis of the available parallelism and optimization opportunities for

the SNAP computational kernel; (2) description of thread-scalable implementation strategies for SNAP which achieves high performance on conventional multi-core CPUs, energy-optimized highly-threaded processors, and high-performance compute-oriented GPUs; (3) demonstration of SNAP with an unprecedented series of strong scaling simulations on leadership class supercomputing platforms with machine comparisons of both simulation rate and energy efficiency.

2 Mathematical formulation of SNAP

The spectral neighbor analysis potential, or SNAP, is based on the common assumption that energy depends only the local arrangement of atoms in the material. The idea is to describe the local environment of each atom as an expansion of its neighbor density in spherical harmonic functions, and use a combination of the expansion coefficients as the energy contribution associated with that atom. Typically such an expansion uses the familiar basis of spherical harmonic functions $Y_m^l(\theta,\phi)$ multiplied with a separate radial part. Bartok *et al.* [2] instead chose to map the radial distance r to a third polar angle θ_0 and use 4D hyper-spherical harmonic functions $U^j_{m,m'}(\theta_0,\theta,\phi)$ as the basis for the expansion of the neighbor density of one atom:

$$\rho(\mathbf{r}) = \sum_{j=0,\frac{1}{2},\ldots}^{\infty} \sum_{m=-j}^{j} \sum_{m'=-j}^{j} u^j_{m,m'} U^j_{m,m'}(\theta_0,\theta,\phi) \tag{1}$$

The neighbor density of a central atom i in a particular configuration of atoms is written as a weighted sum of δ-functions. The sum is over all neighbor atoms i' within a cutoff distance R_{cut}:

$$\rho_i(\mathbf{r}) = \delta(\mathbf{0}) + \sum_{r_{ii'}<R_{cut}} f(r) w_{i'} \delta(\mathbf{r}-\mathbf{r}_{ii'}) \tag{2}$$

Here $\mathbf{r}_{ii'}$ is the 3D vector joining the position of the atoms i and i'. The $w_{i'}$ coefficients are dimensionless weights that are chosen to distinguish atoms of different chemical elements, while the central atom is arbitrarily assigned a unit weight. The function $f(r)$ ensures that the contribution of each neighbor atom goes smoothly to zero at R_{cut}. Using this formulation, each expansion coefficient can be expressed as a discrete sum over the neighbors:

$$u^j_{m,m'} = U^j_{m,m'}(0,0,0) + \sum_{r_{ii'}<R_{cut}} f(r_{ii'}) w_{i'} U^j_{m,m'}(\theta_0,\theta,\phi) \tag{3}$$

The coefficients $u^j_{m,m'}$ are complex and not invariant under rotation. Instead of using them directly in an expression for the energy, the so-called bispectrum components $B_{j_1,j_2,j}$ are used which are calculated as:

$$B_{j_1,j_2,j} = \sum_{m_1,m_1'=-j_1}^{j_1} \sum_{m_2,m_2'=-j_2}^{j_2} \sum_{m,m'=-j}^{j} (u^j_{m,m'})^* H^{j,m,m'}_{j_1,m_1,m_1',j_2,m_2,m_2'} u^{j_1}_{m_1,m_1'} u^{j_2}_{m_2,m_2'} \tag{4}$$

The constants $H^{j,m,m'}_{j_1,m_1,m'_1,j_2,m_2,m'_2}$ are known coupling coefficients, analogous to the Clebsch-Gordan coefficients for rotations on the 2-sphere. The bispectrum components $B_{j_1,j_2,j}$ are invariant under rotation and real valued. For later usage it is convenient to now also define the partial sums

$$Z^{m,m'}_{j_1,j_2,j} = \sum_{m_1,m'_1=-j_1}^{j_1} \sum_{m_2,m'_2=-j_2}^{j_2} H^{j,m,m'}_{j_1,m_1,m'_1,j_2,m_2,m'_2} u^{j_1}_{m_1,m'_1} u^{j_2}_{m_2,m'_2}. \tag{5}$$

The total energy due to the SNAP potential is composed of the N atom energies. The energy of each atom i is written as a linear sum of the K bispectrum components

$$E_{SNAP} = \sum_{i=1}^{N} E^i_{SNAP} = \beta^{\alpha_i}_0 + \sum_{k=1}^{K} \beta^{\alpha_i}_k B^i_k, \tag{6}$$

where α_i indicates the chemical element identity of atom i. The β^α_k are the SNAP linear coefficients for atoms of type α, which are obtained by weighted least-squares linear regression against a large set of quantum calculations. Hence the problem of generating the inter-atomic potential has been reduced to that of choosing the best values for the SNAP linear coefficients. The SNAP force on an atom i can be expressed as a local sum over neighbors:

$$\mathbf{F}^i_{SNAP} = -\nabla_i E_{SNAP} = -\sum_{r_{ii'}<R_{cut}} \sum_{k=1}^{K} \beta^{\alpha_{i'}}_k \frac{\partial B^{i'}_k}{\partial \mathbf{r}_i}. \tag{7}$$

3 Force algorithm and Parallelism in SNAP

The force on each atom due to the SNAP potential is formally given by Equation 7. In order to perform this calculation efficiently, we use a neighbor list, as is standard practice in the LAMMPS code [8, 10]. This list identifies all the neighbors i' of a given atom i. In order to avoid negative and half-integer indices, we have switched notation from $u^j_{m,m'}$ to $u^\eta_{\mu,\mu'}$, where $\eta = 2j$, $\mu = m + j$, and $\mu' = m' + j$. Analogous transformations are used for $H^{j,m,m'}_{j_1,m_1,m'_1,j_2,m_2,m'_2}$ and $B_{j1,j2,j}$. Also, from this point on we identify the neighbor atom index with j instead of i'. Finally, boldface symbols with omitted indices such as $\mathbf{u}_i$ are used to indicate a finite multidimensional array of the corresponding indexed variable.

Figure 1 gives the resulting force computation algorithm, where calc_U(i) calculates all expansion coefficients $u^\eta_{\mu,\mu'}$ from (3) for an atom i while calc_Z(i,$\mathbf{u}_i$) determines all $Z^{\mu,\mu'}_{\eta_1,\eta_2,\eta}$ as defined in (5). In the inner-loop first the derivatives of $\mathbf{u}_i$ with respect to the distance vector between atoms i and j are computed and then the derivatives of B^i. The most computational expansive part of the algorithm is calc_dBdR(i, j) as given in Fig. 2. For the parameter sets used in this study, it is responsible for approximately 90% of all floating point and memory operations. Thus, we concentrate our description on this function.

Compute_SNAP() {
  for i in natoms() {
    $\mathbf{u}_i$ = calc_U(i)
    $\mathbf{Z}_i$ = calc_Z(i,$\mathbf{u}_i$)
    for j in neighbors(i) {
      $\nabla_j \mathbf{u}_i$ = calc_dUdR(i, j,$\mathbf{u}_i$)
      $\nabla_j \mathbf{B}^i$ = calc_dBdR(i, j,$\mathbf{u}_i$,$\mathbf{Z}_i$,$\nabla_j \mathbf{u}_i$)
      $\mathbf{F}_{ij} = -\boldsymbol{\beta} \cdot \nabla_j \mathbf{B}^i$
      $\mathbf{F}_i$ += $-\mathbf{F}_{ij}$; $\mathbf{F}_j$ += $\mathbf{F}_{ij}$
} } }

Fig. 1. Base algorithm for the SNAP force calculation.

Function Calc_dBdR(i, j) {
  for (η, η_1, η_2) in GetBispectrumIndices() {
    $\nabla_j B_{\eta_1,\eta_2,\eta} = 0$
    for $(\mu = 0; \mu \leq \eta; \mu{+}{+}\ )$ {
      for $(\mu' = 0; \mu' \leq \eta; \mu'{+}{+})$ {
        $\nabla_j Z^{\mu,\mu'}_{\eta_1,\eta_2,\eta} = 0$
        for $(\mu_1 = \max(0, \mu + (\eta_1 - \eta_2 - \eta)/2);$
              $\mu_1 \leq \min(\eta_1, \mu + (\eta_1 + \eta_2 - \eta)/2); \mu_1{+}{+})$ {
          $\mu_2 = \mu - \mu_1$
          for $(\mu'_1 = \max(0, \mu' + (\eta_1 - \eta_2 - \eta)/2);$
                $\mu'_1 \leq \min(\eta_1, \mu' + (\eta_1 + \eta_2 - \eta)/2); \mu'_1{+}{+})$ {
        $\mu'_2 = \mu' - \mu'_1$
        $\nabla_j Z^{\mu,\mu'}_{\eta_1,\eta_2,\eta}$ += $H^{\eta,\mu,\mu'}_{\eta_1,\mu_1,\mu'_1,\eta_2,\mu_2,\mu'_2}(u^{\eta_1}_{\mu_1,\mu'_1} \nabla_j u^{\eta_2}_{\mu_2,\mu'_2} + u^{\eta_2}_{\mu_2,\mu'_2} \nabla_j u^{\eta_1}_{\mu_1,\mu'_1})$
        } }
        $\nabla_j B_{\eta_1,\eta_2,\eta}$ += $(u^{\eta}_{\mu,\mu'})^* \nabla_j Z^{\mu,\mu'}_{\eta_1,\eta_2,\eta} + Z^{\mu,\mu'}_{\eta_1,\eta_2,\eta} (\nabla_j u^{\eta}_{\mu,\mu'})^*$
} } } }

Fig. 2. Algorithm for the calculation of the bispectrum component derivatives. Typically more than 90% of all floating point operations are due to this function.

From these two algorithms it is clear that the overall structure of the force calculation is a seven-dimensional loop with the innermost loop performing several complex number calculations. Fortunately these loops provide a high degree of parallelism.

The outermost level is a loop over the atoms in a system. Part of that parallelism is used for domain decomposition on an MPI level, but each MPI rank will typically still handle multiple (and in many classic large scale simulations even millions) atoms. For the bulk of scientifically relevant simulations the number of atoms in a system lies in the range of $10^5 - 10^6$ atoms. The SNAP force calculation for each central atom i is completely independent of other atoms, assuming that current valid positions of all atoms have been distributed. In homogeneous systems, the total computational work for each atom is approximately the same.

The next loop is over the number of neighbors j of each atom i, calculating the force $\mathbf{F}_{ij}$ due to the dependence of E^i_{SNAP} on the positions of i and j. For the parameter sets of SNAP so far developed at Sandia (including the ones

Sets	System	Atom	Interaction	Workload Variation
i	1.0×10^5	N/A	N/A	Low
i,j	1.6×10^6	15.5	N/A	None
i,j,η_1,η_2,η	1.0×10^8	992	64	Large
$i,j,\eta_1,\eta_2,\eta,\mu,\mu'$	2.1×10^9	32,528	1,388	Large
$i,j,\eta_1,\eta_2,\eta,\mu,\mu',\mu_1,\mu_1'$	2.4×10^{10}	2.4×10^5	15,183	None

Table 1. Available parallelism for the SNAP potential with the parameter sets for silicon and tantalum atoms used in this study. Assumed is a system size of 100,000 atoms and the number of unique sets of loop indices corresponding to algorithms is given for the full system, for an average atom, and for one i,j interaction.

used for the later benchmark runs) the average number of neighbors per atom is 14.6. For the class of materials targeted by SNAP it is expected that typical numbers are in the range of 15-50 neighbors per atom. Calculations on different pairs of atoms i,j are again independent except for a reduction at the end to accumulate force contributions into $\mathbf{F}_i$ and $\mathbf{F}_j$. Note that interactions with the same central atom i require the same sets of $\mathbf{u}_i$ and $\mathbf{Z}_i$. The computational work for each interaction is exactly the same for the single chemical element systems considered in this paper.

Within the function Calc_dBdR(i,j) the first loop is over the number of bispectrum components. For the parameter sets considered here there are 64 coefficients. As was the case on the two outer loops, calculations of different bispectrum components are essentially independent. After Calc_dBdR is finished a reduction over the bispectrum components takes place to compute $\mathbf{F}_{ij}$. In contrast to the previous two loops the work for each bispectrum component can be vastly different due to the complicated dependence of the loop boundaries for μ, μ', μ_1, and μ_1' on the specific values of η_1, η_2, and η.

The next two loops over μ and μ' perform a reduction to calculate $\nabla_j B_{\eta_1,\eta_2,\eta}$, where for each pair μ and μ' a different $\nabla_j Z^{\mu,\mu'}_{\eta_1,\eta_2,\eta}$ is calculated through the two inner loops again as a reduction. The total number of unique sets $(\eta_1,\eta_2,\eta,\mu,\mu')$ is 1,388 for the SNAP parameter sets used in this study, while the total number of executions of the innermost loop body is 15,183 for each interaction. With the number of floating point operations in the innermost loop being 52, a single interaction requires on the order of 10^6 operations.

Table 1 provides a summary of the available parallelism. Depending on the targeted machine architecture different levels of parallelism have to be exploited. For CPU based architectures, large computer systems have on the order of 10^6 hardware threads. The largest such supercomputer is Sequoia, an IBM BG/Q system installed at the Lawrence Livermore National Laboratories in the United States, with roughly 100,000 nodes. Each node has 16 compute cores with 4 hyperthreads. Combined, this provides about 6×10^6 threads. Systems based on many-core architectures, such as the current Top500 leader Tianhe supercomputer, provide on the order of 10^7 threads. GPU based systems require even higher concurrency. With the number of threads needed per GPU on the order

System	Nodes	Cores	Threads	Power/Node	Perf./Node
Chama	1,230	19,680	39,360	368.8 W	332.8 GFLOP/s
Sequoia/Vulcan	122,880	2.0×10^6	7.9×10^6	80.26 W	204.8 GFLOP/s
Titan	18,688	5.0×10^7	5.4×10^8	439.3 W	1450.8 GFLOP/s

Table 2. Relevant hardware parameters of the platforms used for strong scaling experiments. Data was obtained from the November 2013 Top500 list [1].

of 50,000, large installations such as Titan require a total concurrency on the order of 10^9.

Comparing these numbers with the available parallelism in a typical MD simulation using SNAP, a reasonable estimate is that one needs to parallelize over interactions (i, j pairs) on CPU and many-core based systems, while GPU systems require exposure of at least one, potentially two more levels.

In the following we discuss implementation details for CPU and GPU systems. In both cases we use domain decomposition to parallelize over nodes. All interactions of a single atom are handled by its assigned node (i.e. we do not distribute work of the same atom to multiple nodes).

4 CPU Implementation

As discussed in the previous section, it is necessary to parallelize over interactions in order to use large CPU based installations efficiently for the system sizes we want to consider. Parallelizing over interactions has a large appeal, since each interaction requires exactly the same amount of computation. As a consequence, a static work partitioning of interactions to threads is sufficient. Furthermore, relatively good load balancing can be expected even if the number of interactions is very small. The worst case scenario is that there is one more interaction to be computed on a node, than there are hardware threads.

In order to parallelize over interactions the loops over the atoms i and its neighbors j have to be flattened. Doing this poses the question of how to handle the calculation of $\mathbf{u}_i$ and $\mathbf{Z}_i$. We implemented two different approaches: the *shared data* algorithm pulls the calculation of those arrays in front, and stores them for all atoms i in a globally accessible array. The *private data* algorithm handles these arrays as thread-private data. They must be computed by each thread individually each time it encounters an new central atom i. This causes duplicated calculations if multiple threads work on interactions of the same atom (i.e. $N_i \lesssim N_{threads}$). The *shared data* algorithm on the other hand requires much more memory. Together the arrays take about 300 kB per particle [1] , which means that the whole data set will not fit into the last level cache of typical

[1] The $\mathbf{Z}_i$ array is a five dimensional array whose dimension depends on the chosen order of the spherical harmonic density expansion. For the parameter sets used in this this study the dimension is 7. Since $\mathbf{Z}_i$ are complex values the total amount of data is thus $7^5 * 2 * 8$ byte $\simeq 300kB$. The $\mathbf{u}_i$ array is only three-dimensional and does significantly affect the total amount of needed memory.

shared data algorithm

```
Compute_SNAP() {
    for i in natoms() {
        u[i] = calc_U(i)
        Z[i] = calc_Z(i,u[i])
    }
    make_list_of_IJ_pairs()
    parallel_for i,j in pairs() {
        ∇_j u_i = calc_dUdR(i,j,u[i])
        ∇_j B^i = calc_dBdR(i,j,u[i],Z[i],∇_j u_i)
        F_ij = −β · ∇_j B^i
        F_i += −F_ij; F_j += F_ij
} }
```

thread-private data algorithm

```
Compute_SNAP() {
    make_list_of_IJ_pairs()
    parallel_for i,j in pairs() {
        if(i ≠ i_old) {
            u_i = calc_U(i)
            Z_i = calc_Z(i,u_i)
        }
        ∇_j u_i = calc_dUdR(i,j,u_i)
        ∇_j B^i = calc_dBdR(i,j,u_i,Z_i,∇_j u_i)
        F_ij = −β · ∇_j B^i
        F_i += −F_ij; F_j += F_ij
        i_old = i
} }
```

Fig. 3. The *shared data* and *thread-private data* algorithm of the SNAP force calculation.

CPUs if there are significantly more than 100 atoms per node. Fortunately, the *private data* algorithm performs well in that situation, since interactions of the same central atom i will usually be handled by the same thread, eliminating the duplication of calculations.

Figure 4 shows the performance of SNAP running on a dual socket Intel Sandy Bridge CPU system with varying number of atoms (blue line). 32 threads have been used (*i.e.* hyperthreading is enabled). Performance is given in GFLOP/s which is calculated from runtimes using algorithmic floating-point operations and average neighbor counts. This means any redundant implementation calculations for $\mathbf{u}_i$ are not included. For 64 or more atoms the *private data* algorithm has been used. The performance increases with atom count, going from 21 GFLOP/s at 2 atoms to a peak of 47 GFLOPS/s at 64 atoms, which corresponds to 30 interactions per thread. This curve sets the limit for what can be expected in a strong scaling experiment with multiple nodes. When one applies strong scaling until each node has only 2 atoms, parallel efficiency can not be expected to be higher than 37%.

It is worth noting that an efficiency of 37% with only two atoms on a node represents a vast improvement in scalability over commonly used MD simulations. For simple potentials such as Lennard-Jones and EAM, the efficiency at that point would be less than 1%. A parallel efficiency of 37% is also high enough that it is not unreasonable to run large scale simulations with as little as 2 atoms per node. With a simulation rate of 700 timesteps/s this would allow for typical simulation times of a few million timesteps to finish in a matter of hours.

5 GPU Implementation

For the GPU implementation using CUDA more parallelism than just the total number of interactions is needed. As a first approach we utilized the *shared-data* approach with thread blocks of 64 threads handling an interaction. Each

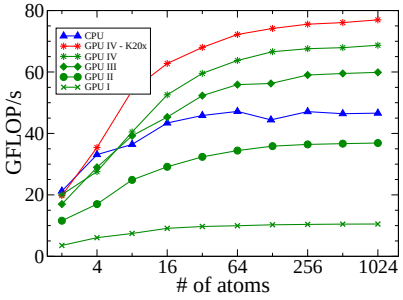

Fig. 4. Performance of SNAP with varying number of atoms on a workstation with dual Intel Sandy Bridge CPUs and NVIDIA GPUs.

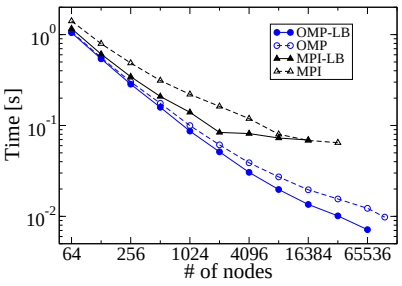

Fig. 5. Comparison of MPI-only and MPI+OpenMP runs with both micro load-balancing enabled and disabled.

thread within a thread block performs the calculation for exactly one bispectrum component. All temporary data including the per-interaction data is kept in global memory, since the amount of available shared-memory (48 kB) would only suffice for a single block with 64 threads. This data is read through the texture cache, mitigating the performance penalty of the irregular access pattern.

Figure 4 shows that the performance achieved with this initial implementation (GPU-I) is approximately four times lower than that of the previously discussed CPU implementation. This might have been expected considering that this first implementation ignores the fact that for each bispectrum component a different amount of work has to be performed. As a consequence strong load imbalances of threads within the same block occur, resulting in many cores being idle for most of the time. In order to estimate how strong this effect is we recorded how many innermost loop executions (work items) happen for each bispectrum component. The bispectrum component with the most work executes the innermost loop 1,369 times, which is 6 times higher than the average number of executions of 237 times and approximately 9% of the total number of work items of 15,183.

In order to address this load imbalance it is necessary to expose a finer level of parallelism. There are 1,388 sets of $(i, j, \eta_1, \eta_2, \eta, \mu, \mu')$ indices with the highest number of work items for a single set being 49. By grouping multiple sets together it is possible to organize the work in a way that each thread in a thread block of up to 320 threads gets approximately 50 work items assigned, resulting in good load balancing. To facilitate this the algorithm is rewritten to flatten out the loops of Calc_dBdR into a single super-loop. The complete 7 index sets are stored as an array of structs, with each thread looping over a subset of the complete list. Since each index only goes from 0 to 7 it was possible to encode the struct into a single 32 bit integer through bit masks. Coefficients representing a 63rd order expansion of the density could be encoded in a 64 bit integer. By sorting the list appropriately it is possible to make sure that the innermost reduction to compute $\nabla_j Z^{\mu,\mu'}_{\eta_1,\eta_2,\eta}$ is handled by a single thread, so that

the sum can be performed in a local variable without atomic updates. $\nabla_j B_{\eta_1,\eta_2,\eta}$ on the other hand is updated atomically.

Since with this approach more threads work on a single interaction (up to 320 instead of 64), shared memory can now be used for the temporary data without restricting the number of active threads. To further increase the number of active threads this temporary data is also stored in single precision only. Using this approach it is possible to fit three active blocks with a total of almost 1,000 threads on a single GPU multi-core, which is high enough to provide the on GPUs necessary latency hiding. Consequently, performance goes up by a factor of approximately three versus the first implementation, achieving 75% of the CPU performance.

A detailed profiling analysis shows that the biggest remaining problem are bank conflicts in shared memory. Looking at the access pattern also reveals that less than half of the entries in the 7x7x7 temporary arrays are ever accessed. Indeed given the indices k,l, and m it holds true that $l \leq k$ and $m \leq k$. Consequently only $\sum_{i=0}^{7} i * i = 140$ will ever be accessed. Using the partial sums it is possible to calculate compressed offsets into the temporary arrays, thus reducing the amount of necessary storage and making accesses more dense. The latter actually reduces the number of bank conflicts. The compression also allows the implementation to return to using full double precision for the temporary data, without reducing the number of threads per GPU multi-core. On top of this, a further reduction in bank conflicts can be achieved by padding the arrays. Since the access pattern is irregular a brute force method was employed to determine the most effective padding size, which turned out to be 159. While this size is specific to the current parameter set using a 7th order expansion, it is trivial to expand the concept and simply determine for each expansion order the most effective padding. This could even been done using autotuning. In combination these two measures improved performance by 50% over the second implementation, making the GPU faster than the CPU variant.

A further improvement is achieved by merging calc_dUdR and calc_dBdR into a single kernel allowing the first part to generate temporary data directly into shared memory instead of putting it into global memory with the calc_dBdR kernel loading it back into shared memory. One issue here is that calc_dUdR does not expose as much parallelism as calc_dBdR. Only 32 threads of a block are taking part in that calculation, and it was necessary to exploit instruction level parallelism to keep those threads busy.

We also experimented with the number of threads by making the buckets for each thread larger or smaller. Experiments showed that 320 threads per block (the maximum number while still being able to distribute roughly equal work to each thread) is not the optimal number, and we used 288 instead. At that point resources on a SM were used almost optimally. The total number of used registers reaches 62k out of 64k and 47.5 kB of the available 48 kB shared memory are utilized.

Resulting improvements, denoted as GPU-IV, added 15% to the previous implementation, reaching 1.47x of the CPU performance. Also shown in Fig. 4

are results with a NVIDIA K20x GPU which is slightly higher performing than the previously used NVIDIA K20c GPU. The latter is a workstation product with active cooling, while the former is intended for server installations and is found in most large clusters including Titan. The K20x reaches a peak performance of 77 GFLOP/s, or 168% of the peak performance of the dual Sandy Bridge CPUs. Exposing all this parallelism enables a respectable 25% efficiency with only 2 atoms on a single GPU. Similar to the CPU implementation about 64 atoms per GPU are required to achieve full performance.

Note that for both CPU and GPU performance is most likely limited by the irregular memory access in the innermost loop of calc_dBdR. There are about 2.5 floating point operations per 8 byte memory load. This means that the K20x provides an effective bandwidth of about 250 GB/s, while the CPU provides an effective bandwidth of about 150 GB/s. Those numbers are only possible for irregular memory access since the work sets are small enough that virtually all accesses are serviced by cache.

6 Scaling studies

Scaling studies were performed on three systems: Chama, Sequoia/Vulcan, and Titan all of which are listed in the November 2013 Top500 list (Chama as # 100, Sequoia as # 3, Vulcan as # 9 and Titan as # 2). The same code described with the CPU-algorithm was used on Sequoia/Vulcan and on Chama. Note that in early 2012 Sequoia and Vulcan, which are actually two partitions of one large BG/Q installation at the Lawrence Livermore National Laboratories, were available as a single system. Our scaling studies were one of the few successfully performed experiments on the entire installation. We believe that the strong scaling results we achieved on this system scaling from a single node to all 122,880 nodes are unique.

For the type of simulations targeted with SNAP, load balancing between MPI processes is a strong requirement. Even when simulating dense materials, small local density fluctuations occur. When trying to scale to single atoms per MPI rank, these small density fluctuations can lead to huge load imbalances. While LAMMPS has load balancing mechanisms, those are targeted at a much coarser level. Essentially LAMMPS decomposes a simulation box with planes in three dimensions. For load-balancing purposes LAMMPS can move those planes independently. This is an approach which works very well for gradual density changes in a system. For the type of fluctuations which pose a problem in our case, this type of load balancing does not work.

To remedy this we implemented a micro load-balancer. Instead of changing domain boundaries it reassigns responsibility of calculating the force for individual atoms. Essentially an MPI rank with more than the average number of atoms, checks all neighboring domains until it finds one with less than the average number of atoms. The load balancer then transfers responsibility for a single atom to that neighbor MPI rank. Consequently a maximum of 26 atoms can be given away, or received in a single load balancing pass. A major cost of this

approach is that generally the halo regions have to be twice as large, so that a MPI rank which receives responsibility to calculate the force of an atom is guaranteed to know about the positions of its neighbor atoms. Early experiments have shown that on GPU based systems this cost is larger than the potential time savings. On CPU based systems on the other hand a large positive effect can be observed.

Figure 5 shows the result of a strong-scaling experiment on Sequoia with 65,536 atoms comparing MPI-only and MPI+OpenMP runs with both micro load-balancing enabled and disabled. For the MPI-only runs 64 MPI ranks per node were used and two per node with 32 threads each for the MPI+OpenMP runs. Using two MPI ranks per node instead of one for the threaded runs enables better utilization of the network resources. Furthermore this means that even with a single atom per MPI rank most threads will actually have an interaction to work on.

First it is obvious that the threaded implementation achieves much better performance than the non-threaded one even for small node counts. Considering the reduction in communication this is not surprising. Also the fastest runtime achieved by the MPI-only runs is 64 ms per timestep as compared to 7 ms per timestep for the OpenMP run with load-balancing. At that point only one in 32 cores actually performs force calculations in the MPI-only runs.

Activating load-balancing improves the runtime of the MPI-only runs. Even with as little as 64 nodes performance increases by 22%. At that point an MPI process has only 8 atoms on average, so small fluctuations in density are noticeable. A detailed analysis showed that the load-balancer achieved uniform distribution of atoms over the MPI ranks. As a consequence, the performance curve is much steeper, reaching a sharp inflection point at 2,048 nodes, where the load-balancer has allocated one atom to every second MPI rank. Interestingly, the inflection point is not at 1,024 nodes, where each MPI rank has 1 atom. This is because on average only two of the four MPI ranks allocated to each processor core have an atom to work on. Each active rank therefore has more hardware resources available. As a result, the performance is improved by up to 1.93x for the MPI-only case, and by up to 1.70x for the hybrid MPI and OpenMP case.

In order to compare different architectures we run a strong scaling experiment of a system with 245,760 atoms on the three large scale clusters available to us: Chama, Sequoia/Vulcan and Titan. The particular system size was chosen because it represents a typical small to medium sized system used in many scientific studies, and it allows us to go to the limit of scaling on Sequoia/Vulcan with 2 atoms per node (1 atom per MPI rank) at full scale. We ran the same system size on the other machines to enable direct comparisons. We show both traditional measures, as well as the power normalized curves because core count or nodes can be a misleading metric to base a comparison on. A Sequoia/Vulcan node for example only uses about 80 Watts and has a theoretical peak of about 200 GFLOP/s, while a Titan node requires about 440 Watts and has a theoretical peak of almost 1500 GFLOP/s. Thus we tie the comparison to energy, one of the two biggest constrains determining super-computer design today (cost,

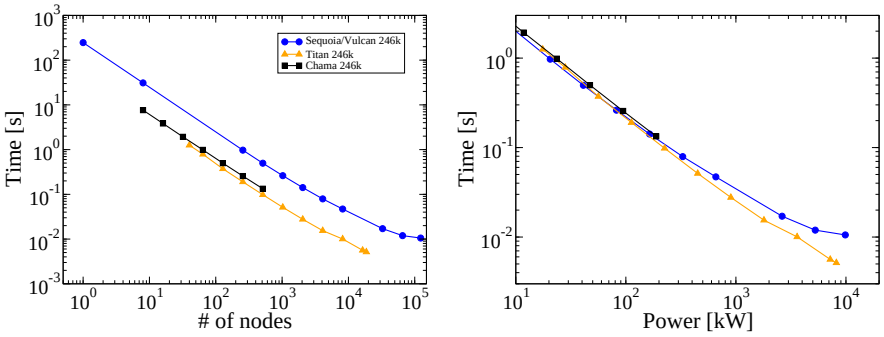

Fig. 6. Time for performing a single simulation step with 246k atoms on Sequoia/Vulcan, Titan and Chama. On the right the curves have been scaled by power per node.

the other major constraint, is hard to quantify and dependent on many soft factors such as exchange rates, rebates inflation etc.). All numbers are based on the data published in the November 2013 Top500 list [1]. Note that the power consumption reported in the Top500 list is obtained when running the HPL LINPACK benchmark, which shares characteristics of our SNAP potential i.e. floating-point dense calculations utilizing a small working set. The node power consumption as well as theoretical peak performance are calculated by taking total system values and dividing by the number of nodes. For power consumption this means that cooling and network energy costs are effectively included in the node values. For Sequoia/Vulcan we used the power per node data from Sequoia. All values are listed in Tab. 2.

Figure 6 shows the time per timestep for a strong scaling run with 245,760 atoms. In the left panel time is plotted against number of nodes, while in the right panel the curves are scaled by power consumption per node. When scaling out to full system size Titan is about two times faster than Sequoia/Vulcan. Most of this factor is due to Titan having more atoms per node at full scale ($\sim$ 13 versus 2 on Sequoia/Vulcan). Thus the surface to volume ratio from the domain decomposition is better and the GPUs are operating in a range where they can still be filled effectively. This is reflected by the parallel efficiencies which drop only to about 50% on Titan compared with 14% on Sequoia/Vulcan. Chama is not large enough to show any significant loss of parallel efficiency for a system of 246k atoms. Normalizing the performance to power consumption as shown on the right in Fig. 6 makes the relative energy efficiency much clearer. For small to medium node counts all three systems are with 20% of each other with respect to energy efficiency. In this regime Sequoia/Vulcan is actually the most effective one followed by Titan and than Chama. Only at larger node counts (i.e. less work per node) Titan is becoming more efficient than Sequoia/Vulcan. The crossover point is reached at about 200 atoms per GPU.

In Fig. 7 normalized performance in GFLOP/s per node is shown with respect to a normalized workload in atoms per node in the left panel, as well as the power normalized curves in the right panel. To get the latter the curves were scaled

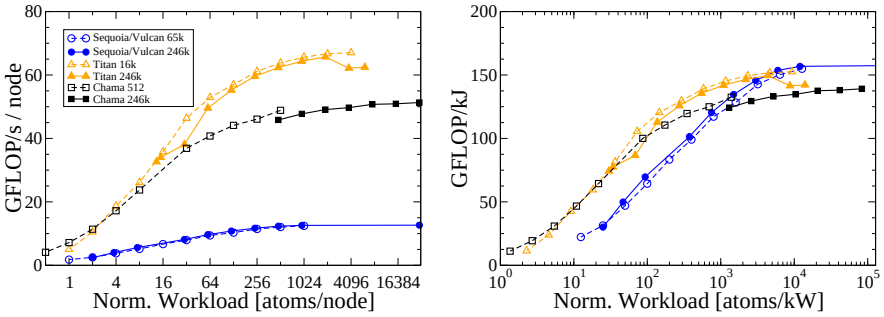

Fig. 7. Normalized performance plotted against normalized workload. In the right panel values have been scaled by power per node. Two strong scaling runs with different number of atoms are shown for each system.

by power per node on both axes. In the figure three more runs with smaller sizes chosen to reach the limit of scaling on the systems are added to the 246k atom runs. Note how the curves for each machine overlap, which means that performance per node is mainly determined by the number of atoms per node and largely independent of the total number of atoms, i.e. very good weak scaling is achieved. While on a per node basis BG/Q is dramatically slower than Chama or Titan, it is as efficient as the other systems for large workloads when normalizing by power consumption. For medium and small workloads Chama and Titan are up to three times as power efficient, since work is much less spread out in those systems, reducing the total amount of communication necessary. Note that in terms of simulation rate Titan achieves its peak performance at 2 atoms/GPU (roughly 4 atoms/kW). Chama can achieve even higher simulations rate by going to a single atom per node, or ~ 1.5 atoms/kW.

7 Conclusion

SNAP is a novel, high-fidelity approach to performing scientifically relevant atomistic simulations at the intermediate scale of 10^5 to 10^6 atoms. In this paper we have presented implementations of SNAP optimized for a range of high-performance computing hardware from contemporary multi-core processors to new energy optimized architectures such as highly threaded CPUs and compute-oriented GPUs. An efficient micro load-balancing scheme is also presented which allows strong scaling down to a single atom per MPI rank. We demonstrate that by performing micro load-balanced, strong scaled simulations we are able to utilize the entirety of the Titan and Sequoia/Vulcan supercomputers - some of the largest leadership platforms currently available. The experiences obtained in developing such an algorithm are a sentinel for many of the issues which algorithm developers may expect to face at Exascale, particularly that such systems can efficiently execute higher fidelity algorithms for contemporary physics applications where simply increasing the problem size may not be scientifically appropriate. Finally, we compare the energy efficiency of these

architectures showing a strong correlation between performance and energy except in the limit of extreme strong scaling where more powerful compute nodes, such at those on Titan, deliver higher energy efficiency.

Acknowledgement

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Furthermore we are grateful for access to compute resources at the Lawrence Livermore National Laboratory and support from Livermore Computing both of which are supported by the U.S. Department of Energy under contract DE-AC52-07NA27344.

References

1. TOP500 Supercomputer Site, `http://www.top500.org`.
2. A.P. Bartok, M. C. Payne, K. Risi, and G. Csanyi. Gaussian Approximation Potentials: the Accuracy of Quantum Mechanics, without the Electrons. *Physical Review Letters*, 104:136403, 2010.
3. J. N. Glosli, D. F. Richards, K. J. Caspersen, R. E. Rudd, J. A. Gunnels, and F. H. Streitz. Extending Stability Beyond CPU Millennium: A Micron-scale Atomistic Simulation of Kelvin-Helmholtz Instability. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, SC '07, pages 58:1–58:11, New York, NY, USA, 2007. ACM.
4. G. Goedecker. Linear Scaling Electronic Structure Methods. *Rev. Mod. Phys*, 71:1085, 1999.
5. M. Griebel, S. Knapek, and G. Zumbusch. *Numerical Simulation in Molecular Dynamics*. Springer Verlag, Heidelberg Germany, 2007.
6. T. Hamada, T. Narumi, R. Yokota, K. Yasuoka, K. Nitadori, and M. Taiji. 42 TFlops Hierarchical N-body Simulations on GPUs with Applications in Both Astrophysics and Turbulence. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 62:1–62:12, New York, NY, USA, 2009. ACM.
7. K. Kadau, T.C. Germann, and P.S. Lomdahl. Molecular Dynamics Comes of Age: 320 Billion Atom Simulation on BlueGene/L. *International Journal of Modern Physics C*, 17(12):1755–1761, 2006.
8. LAMMPS. LAMMPS molecular dynamics package WWW site: lammps.sandia.gov. Potential benchmarks: lammps.sandia.gov/bench.html.
9. A. E. Mattsson, P. A. Schultz, M. P. Desjarlais, T. R. Mattsson, and K. Leung. Designing Meaningful Density Functional Theory Calculations in Material Science–A Primer. *Modelling Simul. Mater. Sci. Eng.*, 13:R1–R31, 2005.

10. S Plimpton. Fast Parallel Algorithms For Sort-Range Molecular-Dynamics. *J. Comput. Phys*, 117(1):1–19, 1995.
11. S. Swaminarayan, T.C. Germann, K. Kadau, and G.C. Fossum. 369 Tflop/s Molecular Dynamics Simulations on the Roadrunner General-Purpose Heterogeneous Supercomputer. pages 1–10, Nov 2008.
12. A.P. Thompson, L. P. Swiler, C. R. Trott, S. M. Foiles, and G. Tucker. A new Quantum-Accurate Interatomic Potential for Tantalum. *(in preparation)*, 2014.
13. Lin-Wang Wang, Byounghak Lee, Hongzhang Shan, Zhengji Zhao, Juan Meza, Erich Strohmaier, and David H. Bailey. Linearly scaling 3d fragment method for large-scale electronic structure calculations. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 65:1–65:10, Piscataway, NJ, USA, 2008. IEEE Press.