OpenAtom: Scalable Ab-Initio Molecular Dynamics with Diverse Capabilities

Nikhil Jain[†], Eric Bohm[†], Eric Mikida[†], Subhasish Mandal[§], Minjung Kim[§], Prateek Jindal[†], Qi Li[⋆], Sohrab Ismail-Beigi[§], Glenn J. Martyna[⋆], Laxmikant V. Kale[†]

[†]Department of Computer Science, University of Illinois at Urbana-Champaign
[§]Department of Applied Physics, Yale University
[⋆]IBM TJ Watson Laboratory

Abstract. The complex interplay of tightly coupled, but disparate, computation and communication operations poses several challenges for simulating atomic scale dynamics on multi-petaflops architectures. OpenAtom addresses these challenges by exploiting overdecomposition and asynchrony in Charm++, and scales to thousands of cores for realistic scientific systems with only a few hundred atoms. At the same time, it supports several interesting ab-initio molecular dynamics simulation methods including the Car-Parrinello method, Born-Oppenheimer method, k-points, parallel tempering, and path integrals. This paper showcases the diverse functionalities as well as scalability of OpenAtom via performance case studies, with focus on the recent additions and improvements to OpenAtom. In particular, we study a metal organic framework (MOF) that consists of 424 atoms and is being explored as a candidate for a hydrogen storage material. Simulations of this system are scaled to large core counts on Cray XE6 and IBM Blue Gene/Q systems, and time per step as low as $1.7s$ is demonstrated for simulating path integrals with 32-beads of MOF on 262,144 cores of Blue Gene/Q.

1 Introduction

Modern supercomputers have become larger and more complex with each successive generation. Although new platforms present novel opportunities, best use of these platforms can be made only by overcoming the challenges that each new architecture poses to the users. Scientific methods and their requirements also change as the domain of interest and goals of research evolve over time. Hence, applications used for simulating scientific phenomena on HPC systems need to grow continually in terms of their scientific capability and parallel scalability.

OpenAtom is a scalable implementation of the Car-Parrinello Ab-initio Molecular Dynamics (CPAIMD) method [7] implemented using the Charm++ runtime system [23]. It is suitable for studying materials at the atomistic level wherein the electronic structure must be explicitly modeled in order to accurately simulate the behavior of the system being investigated. For example, Figure 1 shows the schematic of a metal-organic framework (MOF) that is the subject of materials research as a candidate for hydrogen storage [22], and is currently being simulated using OpenAtom in this regard. Typical studies at this level of detail are generally restricted to a few hundred atoms as they require numerous communication-intensive Fast Fourier Transformations (FFTs). This makes scalable parallelization of such methods challenging. In our previous work, we have shown that OpenAtom is able to make use of Charm++'s asynchrony and object-based overdecomposition approach to overcome these challenges for performing CPAIMD on IBM's Blue Gene/L and Blue Gene/P systems [6, 5].

While the computational capacity of HPC systems has been increasing steadily, the size of scientific systems of interest (such as MOF) has not grown proportionately because the time scales of interest for the study of important phenomena have not been reached (minimal 100-1000 picoseconds). This motivates a drive to achieve fastest time per step as the time step of the discrete time solver is order 0.1 femtosecond. Hence, it is critical that modeling software provide good *strong* scaling for the fixed sized problems being studied.

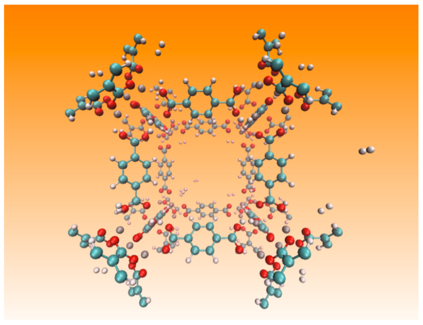


Fig. 1: Schematic representation of MOF with 43 H_2

On the other hand, scientific methods that enable faster convergence (e.g., parallel tempering [10]), or are capable of simulating more complex physical phenomena at atomic scale (e.g. quantum effects using path integrals [20]) require concurrent execution of weakly coupled atomic systems of the same size. Implementation and execution of such scenarios present productivity and performance challenges that also need to be addressed by software such as OpenAtom.

Our recent efforts in OpenAtom have been focused on finding solutions to the challenges described above so that scalable simulations can be performed on production HPC systems. This paper presents these recent additions and improvements to OpenAtom and highlights the following contributions:

- **Generalized topology-aware mapping schemes** for OpenAtom are proposed and their positive impact is demonstrated.
- **Charm-FFT**, a new scalable FFT library which uses 2D-decomposition and minimizes communication, is presented and its benefits are shown.
- **Multi-instance "Uber" method** is a novel scheme added to OpenAtom, which provides a powerful tool to seamlessly implement and execute new scientific methods/variations individually and concurrently. As a result, users can now run methods such as k-points, path integrals, and parallel tempering together in a single run of OpenAtom, if desired.
- **BOMD** [21] is presented as a new addition to OpenAtom's capability.
- **Performance results** that demonstrate the scalability of all scientific methods provided in OpenAtom are presented. A time per step of only 1.7s is shown for simulating 32-beads of MOF on 262,144 cores of Blue Gene/Q.

2 Background and Related Work

OpenAtom is an implementation of the CPAIMD method [7] in Charm++[1], and has been described in [18, 23, 6]. The CPAIMD method is an effective technique to simulate atomistic dynamics on a ground state potential surface derived from a Kohn-Sham (KS) density functional theory formulation within a local or gradient corrected approximation. It has a wide range of applications in chemistry, biology, materials science, and geophysics, etc. [8, 11]. CPAIMD computations involve many phases with complex dependencies, and as such have proven to be difficult to scale. OpenAtom utilizes Charm++'s ability to naturally compose multiple dissimilar modules and thus allows various phases of CPAIMD to overlap in both time and space.

Charm++[1] is an adaptive runtime system built upon the idea of overdecomposed migratable parallel objects that communicate asynchronously via remote method invocations. A key principle in Charm++ applications is that the programmer should not have to think in terms of nodes, cores, or some other hardware specific entity. A program is developed as a collection of parallel objects, called chares, that coordinate via messaging and are composed of both the data and the computation of the particular application. It is then the job of the runtime system to map these objects to the hardware, manage communication between these objects, and schedule them for execution as work becomes available for them. This allows the programmer to decompose the problem in a way that is natural to the algorithm itself, rather than decomposing based on the specific hardware that is being used in a given run.

2.1 Parallelization of OpenAtom in Charm++

Parallelization of the CPAIMD method follows directly from the expression of the density functional and overlap integrals between the KS electronic states [18, 23]. There are over ten different kinds of chares, each representing different phases of the computation as shown in Figure 2. Note that although the phase numbers are linearly increasing, different phases may overlap with each other based on their computation tasks as discussed next.

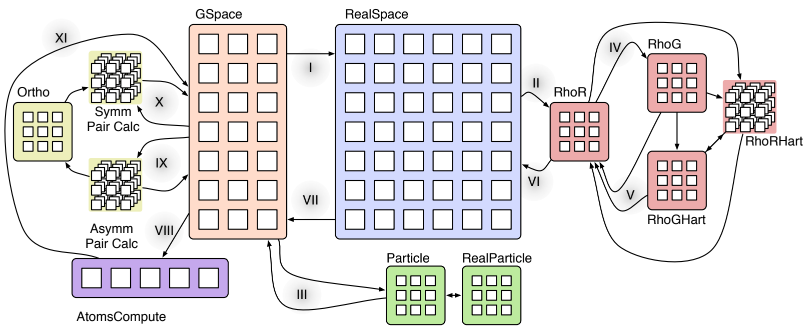


Fig. 2: Parallel structure of OpenAtom

KS Electronic States: Each state, I, has both a discrete real-space ($RS_I(x,y,z)$) and g-space ($GS_I(g_x,g_y,g_z)$) representation, where the latter is the Fourier expansion coefficients of the state in the plane wave basis. The representations are interconverted via concurrent 3D-FFTs (phases I and VII in Figure 2). The real-space representation for each state is decomposed in 1D along planes of the 3D grid. However, application of a spherical cut-off to the g-space representation results in an imbalanced decomposition if 1D decomposition along planes is used. This imbalance is corrected by aggregating small "planes" into larger chunks of data as described in [18].

Electronic Density: The electronic density, ρ, is expressed using both a discrete real-space ($RhoR$) and g-space ($RhoG$) representation, where $RhoR(x,y,z) = \sum_I |RS_I(x,y,z)|^2$. As for the states, $RhoR$ is decomposed along planes, while imbalances in $RhoG$ due to spherical cut-off are corrected by aggregation of smaller planes into larger chunks. The application of the Euler-Exponential Spline method [19] to the computation of local electron-nuclear interaction creates another grid with real-space and g-space components ($RhoRHart$ and $RhoGHart$). In addition to being decomposed like $RhoR$ and $RhoG$, these grids are also decomposed along the number of atom types, $N_{atm-type}$. All these computations are overlapped with each other and contribute to the "Kohn-Sham potential" (phase V), which is communicated to RS (phase VI in Figure 2).

Nonlocal Pseudopotential: The non-local pseudopotential energy accounts for the fact that the CPAIMD method mathematically eliminates "core" electrons and considers only the "valence" electrons. The interaction and the kinetic energy of non-interacting electrons is computed independently of the density-related terms by particle planes, and thus leads to adaptive overlap of phase III with phases II, IV, V, and VI.

Pair Calculators: Corrections are necessary to handle first order variations that cause deviations from orthogonality in $GS_I(g_x,g_y,g_z)$. To do so, the Λ matrix is computed and the forces, $F_{GS_I}(g_x,g_y,g_z)$, are modified:

$$\begin{aligned} \Lambda(I,K) &= \textstyle\sum_g F_{GS_I}(g_x,g_y,g_z)\ GS_K(g_x,g_y,g_z) \\ F_{GS_I}(g_x,g_y,g_z) &-= \ \textstyle\sum_K \Lambda(I,K)\ GS_K(g_x,g_y,g_z) \end{aligned}$$

This task is performed by the PC_{asymm} chares, which concurrently compute matrix multiples for pairs of states (phase IX in Figure 2). Further corrections in second order variations to orthogonality must be applied to the newly evolved states. The PC_{symm} chares perform this task by computing the overlap matrix and are assisted by the *Ortho* chares for computing the inverse square root of the overlap matrix. (phase X in Figure 2).

Atoms: As the atom position data is needed by multiple phases, it is replicated throughout the platform as a *group*, i.e., a chare array with one chare on every processor. Given the small number of atoms, this does not add a significant memory overhead. Integration of the forces to adjust the position of the particles is parallelized up to N_{atms} and is trivial compared to the other operations. This computation is kicked off in phase VIII and is completed in phase XI.

2.2 Related Work

CPMD is an MPI-based implementation of ab-initio molecular dynamics developed through collaboration between IBM and the Max-Planck Institute, Stuttgart [9]. It has a large feature list which includes path integrals and support for excited states. However, in [3], it shows weak scaling up to 256 nodes only. QBox is another MPI-based implementation of first-principle molecular dynamics developed at UC Davis that demonstrated scaling up to 64K nodes of Blue Gene/L for very large atomic systems [13]. Its feature list for performing CPAIMD and BOMD is similar to that of OpenAtom. However, to the best of our knowledge, it does not have native support for multi-instance methods that enables execution of ensemble methods such as k-points, path integrals, parallel tempering, etc. without code input from the users.

In HPC, several application and runtime system developers have studied techniques for mapping [2, 5, 12, 14] to three-dimensional torus topologies with the emergence of supercomputers like the IBM Blue Gene and Cray XT/XE series. Bhatele et al. [4] explore use of information about application's communication patterns and network's topology to create automated tools for generating better mappings. Hoefler et al. [15] discuss generic mapping algorithms to minimize contention and demonstrate their applicability to torus, PERCS, and fat-tree networks. In addition to being custom designed for OpenAtom, mapping techniques presented in this paper differ from related approaches in two ways. First, mapping of a large number of objects of distinct types to processes is performed. Second, instead of being platform dependent, higher level mapping rules are defined to optimize performance on multiple platforms.

3 New Capabilities

The CPAIMD method has been commonly used to simulate nuclear motion on a ground state potential surface. Recently, researchers have extended the basic CPAIMD method in many ways to expand the scope of problems which can be effectively handled. Several of these extensions share a commonality in that they each consist of a set of slightly different, but mostly independent, instances of the standard CPAIMD computation. These include k-points sampling, quantum path integrals molecular dynamics, spin density functionals, and parallel tempering simulations. Depending on the extension, the instances interact in different ways among themselves, but those differences lead to relatively small changes in the overall flow of control. We refer to all these extensions as multi-instance methods.

3.1 Uber Scheme

To support different multi-instance methods, we have implemented an overarching Uber indexing infrastructure. This scheme allows multiple instances to reuse all the objects that implement CPAIMD in OpenAtom by creating distinct copies of the objects that are required by the instances. Objects that belong to a given instance are maintained as a distinct set of chare arrays and thus form an *Uber* comprising one simulation instance. Objects that are shared among different instances are referenced using shallow copies. Furthermore, Ubers are *composable* across different methods, i.e., multiple types of multi-instance methods can be used in any given simulation.

When multi-instance methods are executed, the first step taken by the Uber scheme is the division of compute resources among the instances. Given that the work performed by most of the Ubers is of similar load, a balanced division of compute resources is performed. Section 4.3 presents the schemes that can be used to select specific cores that are assigned to each of the Ubers. Next, objects required for performing simulation within each Uber are created. On any given process and from any of these objects, the variable *thisInstance* can be accessed to find more information about the Uber a given object belongs to. Currently, an Uber is identified by four indices, each of which refers to a type of multi-instance method supported in OpenAtom (discussed next). After the initial set up, all Ubers simulate the configurations assigned to them. Information exchange and synchronization among Ubers is efficiently performed using basic Charm++ constructs such as remote method invocation and collective operations.

In OpenAtom, an Uber is identified by four indices, which are the instance's offsets in four different types of multi-instance methods:

- *Path Integrals*: used to study nuclear quantum effects.
- *Parallel Tempering*: used for sampling to treat rough energy landscapes.
- *k-points*: enables sampling of the Brillion Zone (BZ) to study metals and/or small systems.
- *Spin Density Functional*: treats magnetic systems.

We now briefly describe each of these methods that have been added recently to OpenAtom, except Spin which is currently being implemented.

Path Integrals: In order to explore nuclear quantum effects, Feynman's Imaginary Time Path Integral method (CPAIMD_PI) [20] has been implemented. In this method, each classical nucleus is replaced by a ring polymer of P beads connected by harmonic links to their nearest neighbors. The method's computational complexity increases linearly with P as the inter-bead interactions are imaginary time ordered and each bead group forms a classical subsystem.

CPAIMD_PI has been integrated into OpenAtom such that each Uber has an independent electronic computation ($RS, GS, RhoRS$, etc.) associated with that bead's set of nuclei. Therefore, the entirety of the standard CPAIMD method shown in Figure 2 is local to each Uber. The additional work required to evaluate and integrate the intrapolymer forces to evolve the ensemble is order P. It is implemented by force and position exchanges between each representation of the N nuclear particles from all the beads. This communication extends the standard CPAIMD nuclear force integration phase (phase XI in Figure 2) such that the simulation cannot proceed until the bead forces are computed. Thus, it forces a synchronization across all beads in every time step.

Parallel Tempering: One widely used method to sample rough energy landscapes in statistical physics is Parallel Tempering (PT) [10]. In this method, a set of complete CPAIMD parallel simulations are initiated with different temperatures. The lower temperatures in the set explore low lying minima while the higher temperatures traverse the energy landscape. After every time step, the Ubers that are nearest neighbors in temperature space exchange temperatures via a rigorous Monte-Carlo acceptance rule. The computational complexity of this method also increases linearly with the number of temperatures being explored. However, a global synchronization is not needed at the end of each time step since the temperature exchange only happens among nearest neighbors.

k-points sampling of the Brillion zone: In a previous work, we studied large, insulating systems where computation at only the Γ-point of the Brillion zone (BZ) [7] was sufficient [23]. In small, metallic or semiconducting systems, more points are required, and that is the functionality k-points sampling provides. Away from the Γ-point, at finite $\mathbf{k}$, the states are complex and a set of n_k k-points with weights w_k are used to sample the BZ. Different k-points interact in the formation of the density - there is only 1 density summed over all k-points taking into account the weights. Hence different Ubers get their own copy of state chares, but all of them point to the same density chares ($RhoRS$, $RhoGHart$, etc.) and atoms. The parallel scalability of this method is typically bounded by the time spent in the density phase.

3.2 Born-Oppenheimer Method

Other than CPAIMD, the Born-Oppenheimer method [21] (BOMD) is the other common method used to generate the dynamics of nuclei on the ground state energy surface provided by Kohn-Sham density functional theory. Unlike the CPAIMD method which introduces a fictitious dynamics for the expansion coefficients of the KS-states, under BOMD, the density functional (and hence the expansion coefficients of the KS-states) is minimized and then the atoms are evolved using a straightforward symplectic integrator. This leads to a secular growth in the energy. We have added the capability of using BOMD as an alternative for performing simulations in OpenAtom. Use of BOMD impacts the flow diagram in Figure 2 in the following way: instead of performing phase VIII in every time step, the system is first minimized and then phase VIII is performed.

Method comparison: Both CPAIMD and BOMD methods have been known to be stable and can be used to simulate important scientific phenomena. At any time, an improvement in one method can leap-frog the other as the preferred way to go. The advantage of the BOMD is its simplicity. The disadvantage is that the minimization procedure is truncated at a finite tolerance in practice, which can lead to higher aggregated error.

4 Parallel Optimizations

Parallel implementation of phases described in Section 2.1 leads to several communication intensive operations in OpenAtom. In any given phase, several

FFTs, section-reductions, and multicasts are performed concurrently. Multi-instance methods exacerbate the situation by increasing the number of occurrences of these operations and by adding communication of their own. As a result, it is important that communication is well orchestrated and task-mapping is performed to maximize the network utilization and reduce the overheads.

4.1 Distance-aware mapping

Significant work had been done on mapping OpenAtom to compact the 3D-grid network topology used in systems such as IBM Blue Gene/P [16]. Since the 3D-nature of simulated space (e.g. 3D state grid) matched the 3D-grid of Blue Gene/P's torus, high performing precise mapping schemes were developed to obtain improved performance on those systems. However, the mappings from the past no longer lead to optimal performance because of (1) changes in dimensionality of the networks, e.g. Blue Gene/Q has a 5D-torus, and (2) irregular allocations, e.g. on Cray XE6/XK7 systems, typical allocations are not restricted to an isolated high bisection bandwidth cuboid. Hence, we have developed new schemes that improve upon the old schemes in two ways: portability to a larger set of current supercomputers and less time to compute the mapping.

Separation of concerns: The main principle underlying the mapping improvements is the separation of logic that decides mapping from assumptions regarding the interconnect topology. For example, the new mapping schemes take decisions based on relative closeness of pairs of processes, but how the closeness is defined and computed is left to the topology manager. This separation enables us to define generic rules of thumb on relative placements of objects of various types with respect to other objects.

Boilerplate mapping algorithm: A typical mapping routine for a given chare type consists of three steps: find the available list, reorder/process the list based on the object type, and make assignments. The first step simply queries the topology manager to provide a distance-aware list of available cores/processors. Thereafter, to find suitable candidates among the available cores for the given objects, the available list is either divided among smaller sets or sorted in a particular order using the topology manager. Finally, suitable cores are assigned objects while accounting for load balance and exclusion among various cores. Throughout the process, a highly efficient exclusion list is maintained to down-select cores for mapping remaining objects of the same type or other types.

Distance-aware order of processes: To obtain a list of available cores for an object type, we start with a list of all cores available to the current job. Thereafter, any exclusions defined by the previous mappings of other types of objects are applied. The exclusions are typically useful in assigning to different cores objects of different types that are expected to be active concurrently. In addition, they are used for excluding cores with special tasks, e.g. rank 0 is responsible for control flow management tasks, and is thus given fewer objects. We provide the option to override exclusions either by the user as a configuration parameter or due to lack of sufficient number of cores in the current job.

In the past, for mapping on Blue Gene/P, the ordering of the list was closely tied to the number of chares that host a state in the simulated system. By forcing the number of such chares to be a factor of the number of cores, the mapping was able to divide the available set of cores evenly. Given the isolated cuboidal allocations of Blue Gene/P, the mapping was also able to divide the available set of cores among smaller cuboids and assign them to the states [5], leading to high efficiency communication patterns.

In the new mapping scheme, all the above restrictions have been removed, while preserving the performance. The topology manager orders a given set of cores by making a pass through the set of processors in a topology-aware manner. For making the pass, the available cores are divided among small topologically-close units and ordered accordingly. For example, on Cray's XE6, the traversal is performed along the longest axis using small cubes of size $4 \times 4 \times 4$. The main advantage of such a traversal is the guaranteed topological proximity of the cores that are close in the list. At the same time, communication among a pair of cores, P1, that is reasonably distant from another pair of cores, P2, is less likely to interfere with the communication of the pair P2.

Mapping the states: The two types of state objects, RS and GS, play a central role in the control flow of OpenAtom. Forward and backward FFTs are performed between RS and GS in every iteration. After the forward FFT to RS, a plane-wise reduction on RS is performed to the density objects, which return the result via a multicast to RS. Following the backward FFT to GS, multicast and reductions are performed between GS and pair calculators. Given the plane-based nature of both these operations and the bisection bandwidth requirement of $O(\#states)$ FFTs between RS and GS, it is better to spread RS and GS on the given cores such that communication to/from the planes does not interfere, while the planes use as much bisection bandwidth as possible.

Hence, the mapping code divides the distance-aware ordered list of cores it obtains from the topology manager evenly among the planes of RS/GS using a block-mapping scheme. This mapping does not add any of the cores to the global exclusion list since every core in the system has at least one RS/GS object.

Mapping the density: Contributions from all the RS objects are combined to create the density for $RhoR$ by a reduction operation. Given the plane-based division of RS, the aggregation is also performed along the planes. Hence, to improve the performance of the reduction, the density planes of $RhoR$ are placed near the cores that host the corresponding RS planes. These cores are added to the exclusion list for mapping the remaining density objects. Other density objects, $RhoG$, $RhoRHart$, and $RhoGHart$, are then evenly spread on cores sorted by their distance from the centroid of the cores that host $RhoR$.

Particle planes and pair calculators: Both types of particle planes, RPP and GPP, are closely tied to the states. The GPP objects are co-located with GS objects since they work on a large amount of common data. The RPP objects are spread across the set of cores that host GS/GPP objects for corresponding plane. This helps improve the performance of FFTs between RPP and GPP.

Finally, to map the pair calculators, a new distance-aware list of cores is obtained from the topology manager and the pair calculators are mapped in that order while maintaining load balance. This scheme works well because it places the pair calculators for a given plane close to where its *GS* objects are mapped.

4.2 Overdecomposed FFTs with cutoffs

The existing code for performing parallel FFTs in OpenAtom is based on 1D-decomposition of data. Hence, the amount of parallelism available for state and density FFTs are $O(\#states * \#planes)$ and $O(\#planes)$, respectively. In a typical OpenAtom simulation, the number of states is at the most 1000. Each of the states is represented using grids that contain at the most $300 \times 300 \times 300$ points. This implies that the maximum parallelism available in state FFTs is $O(300,000)$, but is only $O(300)$ for density FFTs. Thus 1D-decomposition based density FFTs severely limits the scalability of OpenAtom, especially on large machines with many more compute nodes.

Charm-FFT overview: To eliminate the scaling bottleneck due to density FFTs, we have developed a fully asynchronous Charm++ based FFT library, Charm-FFT. This library allows users to create multiple instances of the library and perform concurrent FFTs using them. Each of the FFT runs in the background as other parts of user code execute, and a callback is invoked when the FFT is complete. The key features of this library are:

1. *2D-decomposition*: Users can define fine-grained 2D-decomposition that increases the amount of available parallelism and improves network utilization.
2. *Cutoff-based smaller grid*: The data grid typically has a cutoff in g-space, e.g. density has a g-space spherical cutoff. Charm-FFT improves performance by avoiding communication and computation of the data beyond the cutoff.
3. *User-defined mapping of library objects*: The placement of objects that constitute the library instance can be defined by the user based on the application's other concurrent communication and placement of other objects.
4. *Overlap with other computational work*: Given the callback-based interface and Charm++'s asynchrony, the FFTs are performed in the background while other application work can be done in parallel.

Charm-FFT details: The creation of an instance of the library is performed by calling `Charm_createFFT` from any process. The user is required to specify the size of the FFT grid and the desired decomposition. A cutoff and mapping of the FFT objects can also be specified. Optionally, a callback can be specified which is invoked when the distributed creation of the library instance is completed. Internally, three types of Charm++ objects are created: `D1`, `D2`, and `D3`. Each of these objects owns a thin bar (a pencil) of the FFT-grid in one of the dimensions, e.g. in Figure 3(a), `D1` objects own pencils along Z axis. The decomposition of the FFT-grid among these objects is decided based on the user input.

Typically, `D1` objects are associated with the grid in the real-space, while `D3` objects are used for the grid in the g-space. The `D2` objects are not visible to the user as they are used for the intermediate transpose only. Before executing

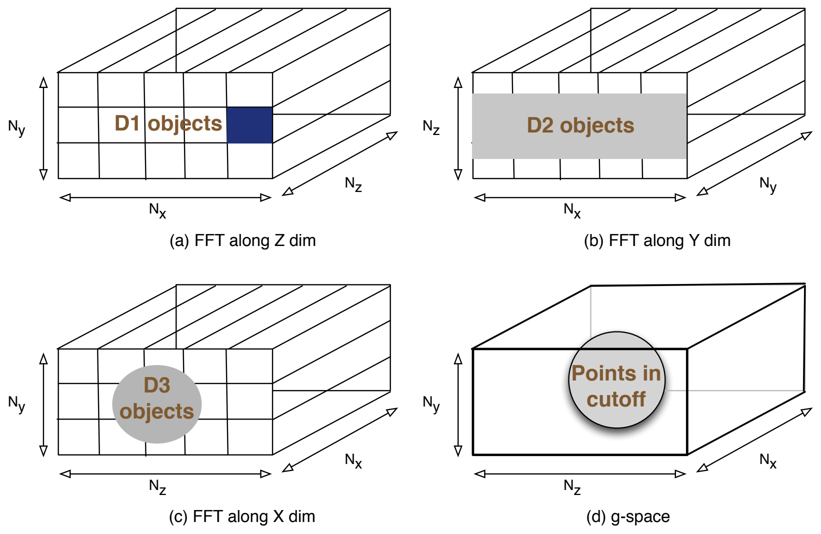


Fig. 3: Charm-FFT: concurrent cutoff-based FFTs with 2D-decomposition.

FFTs, the user is required to inform `D1` and `D3` objects about the memory where the grid resides by making local API calls in a distributed manner.

When the setup is complete, an FFT can be started on an instance by calling `Charm_doForwardFFT` or `Charm_doBackwardFFT`. These calls return immediately without actually performing the FFT, but after registering it with the library. If `Charm_doForwardFFT` is invoked, the FFTs are performed locally along Z dimension by `D1` objects. Following this FFT, any data along Z axis that is beyond the cutoff is ignored, and *only the data within the cutoff is communicated* to `D2` (Figure 3(b)). On `D2` objects, FFT along Y dimension is performed which further reduces the FFT-grid to a cylinder of thin bars as shown in Figure 3(c). This data is communicated to `D3` objects, where FFT along X dimension reduces the cylinder to a sphere (Figure 3(d)). At this point, the user specified callback is invoked informing the application of FFT's completion. The distribution of pencils, which have grid points within the sphere, to `D3` objects is performed such that the total number of grid points that are in the sphere are load balanced across `D3` objects. For `Charm_doBackwardFFT` call, these steps are performed in reverse order. Note that if *FFTs are started on multiple instances one after the other, all of them are performed concurrently.*

Adapting OpenAtom to use Charm-FFT: In order to use Charm-FFT with OpenAtom, a significant fraction of the density object implementation has been rewritten. This is because the decomposition of density objects is tied to the decomposition of the FFT-grid. The integration has provided three benefits:

1) The decomposition of the density objects is no longer restricted to be a 1D-decomposition. Users can choose a decomposition that suits their system.

2) The RS to density reduction is now divided among finer chunks and is targeted to objects that are distributed among more cores. This is likely to improve the performance due to better utilization of the network.
3) The significant lines of code (SLOC) count for the control flow of density has been reduced by more than 50% from $4,198$ to $1,831$.

Mapping of FFT objects: To make the best use of the 2D-decomposition of density, a new mapping scheme has been developed for the density and Charm-FFT objects. Since the *RhoR* objects are no longer tied to only one plane of RS, they are evenly spread among the available cores. However, while spreading them uniformly, we attempt to keep *RhoR* objects close to the RS planes with which they communicate. Other density objects are similarly spread while maintaining their proximity to the *RhoR* objects with which they interact. The Charm-FFT objects, `D1` and `D3`, are colocated with the real-space and g-space objects. The `D2` objects are assigned in close proximity of the `D1` objects they interact with.

4.3 Scaling multiple instances

When multiple instances, i.e., Ubers described in Section 3.1, are executed concurrently, two additional concerns arise: 1) How should the objects that belong to different Ubers be mapped? 2) What impact does presence of multiple instances have on the performance of OpenAtom? In this section, we explore these issues and discuss how they are addressed in OpenAtom.

In Section 3.1, we have described the implementation of different multi-instance methods. From that description, it is easy to see that inter-Uber communication is infrequent and low volume. Hence, it is preferable to map objects of different Ubers on different cores, so that they do not interfere with each other. We have experimented with two types of mappings based on this idea:

1) Disjoint partitions (DPS): In this scheme, the ordered distance-aware list of cores created by the topology manager is divided evenly among the instances using a block-mapping scheme. Given the topologically sorted property of the list, this reduces interference among the intra-Uber communication of different Ubers. This also reduces the number of hops for communication within a Uber.

2) Interleaved partitions (IPS): This scheme divides the topologically sorted list of cores among various instances in a round-robin manner. Here, while the intra-Uber communication of different Ubers may interfere, the increased bisection bandwidth may improve the performance of the 3D-FFTs. This scheme may also benefit from overlap of computation time of one Uber with communication of other Ubers since cores connected to a router are assigned to different Ubers.

Performance comparison: To compare the two schemes, DPS and IPS, we execute the MOF system (Section 5) with 2, 4, and 8 Ubers, where each Uber is allocated 2,048 cores of Blue Waters. When only two Ubers are executed, both schemes provide similar performance. However, as the number of Ubers is increased to four and eight, DPS reduces the time per step by up to 31% and 40%, respectively in comparison to IPS. From these results, we conclude that avoiding interference among intra-Uber communication of different Ubers is better, and thus DPS is used for all the remaining results in this paper.

Effect of Ubers on performance

Figure 4b presents the time line view of Projections [17] obtained when OpenAtom is executed with four Ubers on a Blue Gene/Q system for one time step. Each horizontal bar in this figure shows the computation executed on a core (or process) colored using the legend shown in Figure 4a. It can be observed that a significant fraction of the timeline is colored white, which implies high idle time.

High idle time is observed because the execution of different Ubers is not as synchronized as it should be given their similar workload. As seen in Figure 4b, this is because the start of the time step in some Ubers is delayed (highlighted in the figure), which in turn is caused by these Ubers waiting on information computed by the multi-instance methods. Longer waits are observed for some Ubers since transmission of such information is blocked by forward progress made by other Ubers that have already received the information. To avoid these delays, we force all Ubers to wait at the end of each time step till all instances have received the data needed to perform the next time step.

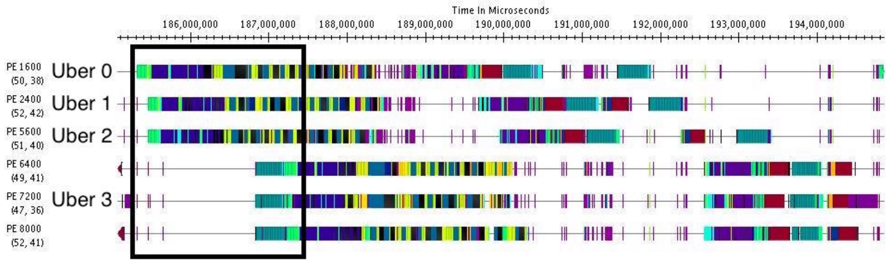
Figure 4c shows another type of delay caused by inter-Uber interference. This delay is because, in Charm++, global operations such as broadcasts and reductions are implemented using optimized tree-based construction that spans all cores (as is done in most parallel languages). However, when multiple instances are executed, the global broadcasts and reductions are meant for only a subset of objects on some cores. Such operations may get delayed if intermediate cores, which are not part of the source Uber, are busy performing other work. This inefficiency is removed by replacing global broadcasts and reductions by Charm++'s sections-based operations. Use of these constructs ensures that only participating cores are used for forwarding data during global operations, and thus minimizes aforementioned delays.

After eliminating idle time due to inter-Uber interference, we observe that the small amount of additional work done for the multi-instance methods unexpectedly takes a very long time as highlighted in Figure 4d. We find two reasons for this: 1) Excessive fine-grained division of work required by multi-instance methods leads to a large number of small-sized broadcasts and reductions, 2) Core 0 is overloaded since it is assigned work both as a member of an Uber and as the multi-instance method coordinator. These issues are solved first by increasing the granularity of the chare array that performs the multi-instance method; this reduces the number of broadcasts and reductions. Second, core 0 is excluded from being assigned work for any Uber.

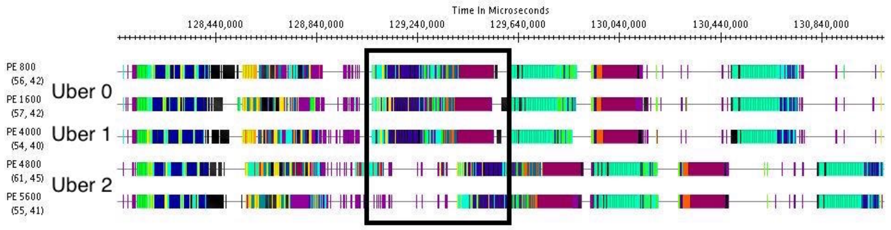
Figure 4e presents the last performance issue we observe: as core 0 is offloaded, one of the cores in one of the Ubers (which gets one less core than others) gets overloaded with the pair calculator work resulting in some performance loss. To remove this inefficiency, we allow the affected Uber to place only pair calculators on core 0. This works fine because no other computation overlaps with the computation done by pair calculators.

Idle	⬜	Non-local init	🟩
RTS overhead	⬛	Z-matrix computation	🟦
Orthonormalization backward path	🟦	State FFT (forward)	🟩
Non-local FFT	🟦	Particle plane receive	🟨
State FFT (backward)	🟪	Compute atom forces	🟧
Force FFT	🟪	SDAG overhead	🟦
Orthonormalization forward path	🟥	State force computation	🟩

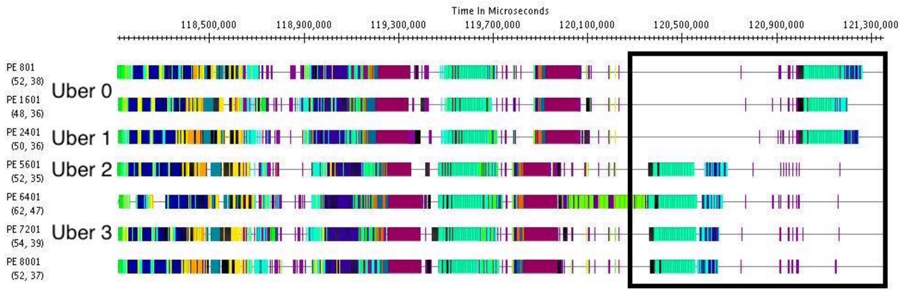
(a) Legend for the performance analysis graphs



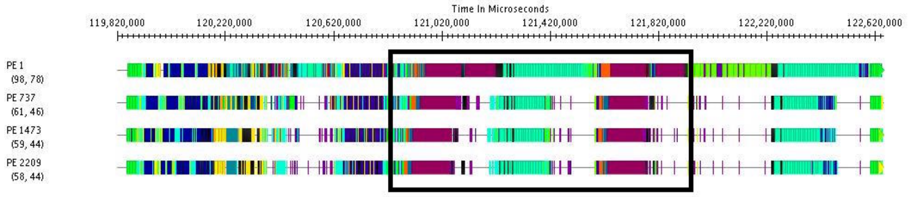
(b) Out of sync Ubers cause idle time.



(c) Global reductions lead to interference.



(d) Idle time after CPAIMD.



(e) Load imbalance on one of the cores.

Fig. 4: Multi-instance performance optimization.

5 Scaling Results

In this section, we present scaling results obtained by integrating capabilities and optimizations described in Sections 3 & 4 into OpenAtom. All the experiments have been performed on Blue Waters, a Cray XE6/XK7 system, and Vulcan and Mira, IBM Blue Gene/Q systems. Most experiments were repeated five times to account for runtime variabilities, but only up to 1% deviation was observed on both types of systems.

We use two systems of scientific interest in these studies: Liquid water and Metal-organic Framework (MOF). Water is a simple system which contains a box of water with 32 molecules. MOF is a more complex larger system used to study suitability of metal-organic frameworks (Figure 1) for H_2 storage [22]. The MOF system used in this paper is MOF-5 which comprises $Zn_4O(BDC)_3$ (BDC 1,4 benzenedicarboxylate). It contains 424 atoms, 1552 electrons, and 776 KS states. We have found it to be stable at the cutoff of 50 Rydberg, which is used in our simulations. Each state is represented by a $220 \times 220 \times 220$ size grid.

5.1 Performance of Charm-FFT

The first set of results shows the performance of Charm-FFT as a FFT library. To understand the impact of decomposition on the time taken to compute a 3D-FFT, we perform a FFT of a $300 \times 300 \times 300$ size grid on 512 nodes of Blue Gene/Q using different decompositions. For these experiments, the baseline execution time is 76 ms, which is obtained when 1D-decomposition of the grid is performed, i.e., 300 objects are used. In Figure 5, it can be seen that as we perform finer decomposition of the grid along two dimensions, the time to compute 3D-FFT reduces significantly.

#Objects	Decomposition	Time (ms)
100	10×10	80
300	300×1	76
300	75×4	69
300	20×15	45
400	20×20	35
900	30×30	24
1600	40×40	24
2500	50×50	22
3600	60×60	23

Fig. 5: FFT on a $300 \times 300 \times 300$ grid.

The best performance is obtained when the grid is divided among 2,500 objects that are arranged as a 2D grid of size 50×50. In this case, the time to perform FFT is reduced by 70% in comparison to the baseline. Further decreasing the decomposition granularity leads to excess communication overhead.

Figure 6(left) demonstrates that the choice of cutoff can have a significant impact on the time to perform FFT. For a grid of size $300 \times 300 \times 300$, up to 3x reduction in execution time can be seen on 512 nodes of Blue Gene/Q. While cutoffs as low as 100 are unrealistic from a scientific perspective, G^2 values that eliminate as many as half the grid points are common. In Figure 6(left), the x-axis value of 6,400 represents this common scenario, where 41% reduction in execution time is observed.

Finally, in Figure 6(right), we present the impact of using Charm-FFT in OpenAtom for the 32-molecule Water system scaled to core counts that are

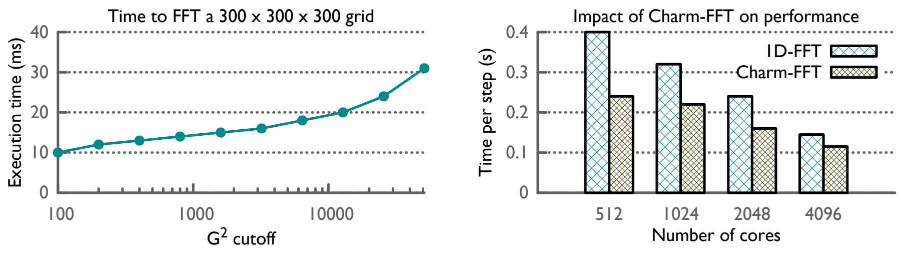


Fig. 6: (left) As the G^2 cutoff decreases, time to FFT reduces. (right) Charm-FFT improves the time per step of OpenAtom by up to 40%.

at the parallelization limits of the Water system. For most of the core counts, Charm-FFT is able to increase the available parallelism and reduces the time per step by 30-40%. For core counts less than 400, we find that the performance of the default version of OpenAtom matches closely with the version that uses Charm-FFT. This is expected since for small core counts, the default 1D-decomposition is able to utilize most of the network bandwidth.

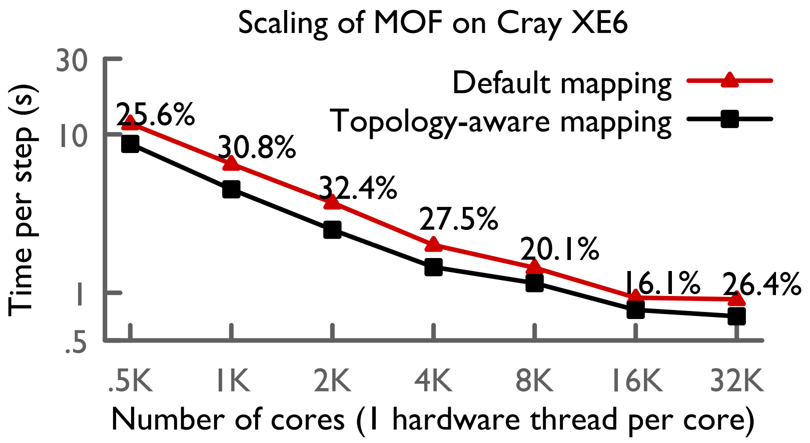
5.2 Single instance execution

In this section, we present performance results for simulating a single instance of MOF with OpenAtom. Figure 7 shows strong scaling results when the core count is increased from 512 to 32,768 on Blue Waters. It can be seen that the time per step decreases significantly from 11.7 seconds to less than a second as more cores are used. Our topology-aware mapping scheme consistently provides a performance boost of $16 - 32\%$ on all system sizes. Similar improvements are obtained on Mira where three hardware threads are utilized on every core. Best execution time of $0.67s$ per time step is obtained on 32,768 cores of Blue Waters for the MOF system which has only 776 electronic states. Note that topology aware mappings are computed once at the beginning of long running simulations. Hence, overhead due to such computations is minimal. For example, for a typical science run of several hours on 1,024 nodes, computing the mapping takes less than 3.2 seconds.

In Figure 8, good scalability is shown for BOMD computation as we scale from 4,096 cores to 16,384 cores on Blue Gene/Q. Use of topology-aware mapping outperforms the default mapping by up to 32% in these cases. In fact, with topology-aware mapping, 74% reduction in time per step is obtained when the number of cores is increased by four times, i.e., perfect scaling is observed. These results strongly indicate that OpenAtom is able to provide scalable support to different simulation methods by exploiting their common characteristics.

5.3 Scalability of Ubers

Now, we present performance results for simulating multiple instances of MOF on Blue Gene/Q. As a representative of multi-instance methods, we use the Path



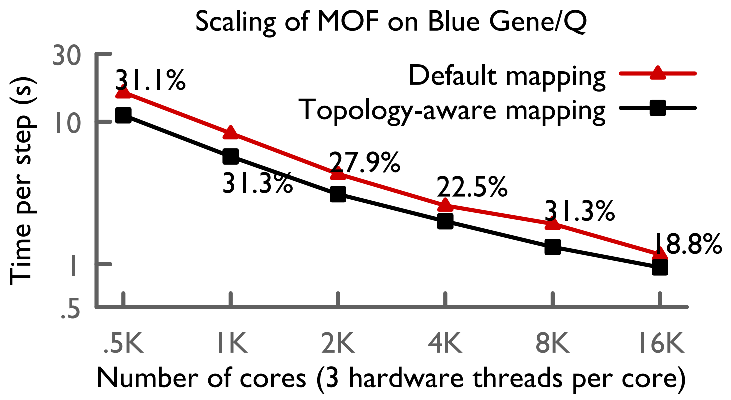


Fig. 7: OpenAtom shows good scaling on both Cray XE6 and Blue Gene/Q. Benefit of topology aware mapping is significant as shown by the % values.

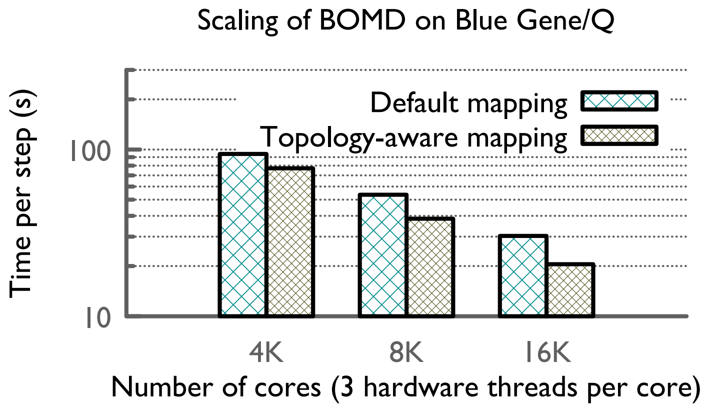


Fig. 8: Perfect scaling and positive impact of topology-aware mapping is demonstrated for BOMD computation.

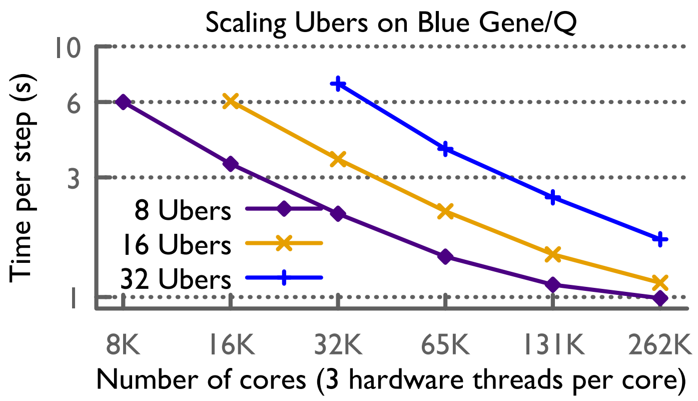


Fig. 9: By exploiting Uber infrastructure and topology-aware mapping, OpenAtom enables strong scaling of Path Integrals up to a quarter million cores on Blue Gene/Q. These are representative results that should extend to other multi-instance methods such as the k-points, Parallel Tempering, and Spin.

Integral method (CPAIMD_PI) in these experiments. Three configurations with 8, 16, and 32 Ubers are strong scaled from 8,192 cores to 262,144 cores, where three hardware threads are used on each core. For each of these configurations, Figure 9 shows that an efficiency of 52% is obtained when the core count is increased by 8x from the smallest configuration executed. For example, the 32 beads simulations observe a 4.2$\times$ speed up when core count is increased from 32,768 to 262,144. Time per step as low as 1.7s is obtained for executing 32 beads on 262K cores of Blue Gene/Q. For other configurations, time per step close to one second is obtained by making use of multiple hardware threads available on the system. In fact, due to the communication intensive nature of these simulations, one out of every three threads is dedicated to advancing communication asynchronously, while the other two threads perform computation.

6 Conclusion

In this paper, we have presented the capabilities and scalability of OpenAtom. New science capabilities, viz. multi-instance methods and BOMD, have been added to OpenAtom recently and are described in this paper. Positive impact of optimization techniques, namely distance-aware mapping, Uber indexing, and overdecomposed 3D-FFTs with spherical cutoff, has also been shown on two production HPC platforms, IBM Blue Gene/Q and Cray XE6. By leveraging these techniques, we have demonstrated that OpenAtom provides efficient strong scaling up to 32,768 cores for MOF, an important science system with only a few hundred atoms. Finally, a time per step of 1.7s and strong scaling up to 262,144 cores have been shown for multi-instance scientific simulations. These results strongly suggest that OpenAtom is a highly scalable simulation code with diverse capabilities.

Acknowledgments

This research is partly funded by the NSF SI2-SSI grant titled Collaborative Research: Scalable, Extensible, and Open Framework for Ground and Excited State Properties of Complex Systems with ID ACI 13-39715. This research is also part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (award number OCI 07-25070) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. This research also used computer time on Livermore Computing's high performance computing resources, provided under the M&IC Program.

References

1. Acun, B., Gupta, A., Jain, N., Langer, A., Menon, H., Mikida, E., Ni, X., Robson, M., Sun, Y., Totoni, E., Wesolowski, L., Kale, L.: Parallel Programming with Migratable Objects: Charm++ in Practice. SC (2014)
2. Agarwal, T., Sharma, A., Kalé, L.V.: Topology-aware task mapping for reducing communication contention on large parallel machines. In: Proceedings of IEEE International Parallel and Distributed Processing Symposium 2006 (April 2006)
3. Alam, S., Bekas, C., Boettiger, H., Curioni, A., Fourestey, G., Homberg, W., Knobloch, M., Laino, T., Maurer, T., Mohr, B., Pleiter, D., Schiller, A., Schulthess, T., Weber, V.: Early experiences with scientific applications on the ibm blue gene/q supercomputer. IBM Journal of Research and Development 57(1/2), 14:1–14:9 (2013), http://dx.doi.org/10.1147/JRD.2012.2234331
4. Bhatele, A.: Automating Topology Aware Mapping for Supercomputers. Ph.D. thesis, Dept. of Computer Science, University of Illinois (August 2010), http://hdl.handle.net/2142/16578
5. Bhatele, A., Bohm, E., Kale, L.V.: Optimizing communication for charm++ applications by reducing network contention. Concurrency and Computation: Practice and Experience 23(2), 211–222 (2011)
6. Bohm, E., Bhatele, A., Kale, L.V., Tuckerman, M.E., Kumar, S., Gunnels, J.A., Martyna, G.J.: Fine Grained Parallelization of the Car-Parrinello ab initio MD Method on Blue Gene/L. IBM Journal of Research and Development: Applications of Massively Parallel Systems 52(1/2), 159–174 (2008)
7. Car, R., Parrinello, M.: Unified approach for molecular dynamics and density functional theory. Phys. Rev. Lett. 55, 2471 ((1985))
8. Carloni, P., Bloechl, P., Parrinello, M.: Electronic Structure of the Cu,Zn Superoxide dimutase active site and its interactions with the substrate. J. Phys. Chem. 99, 1338–1348 ((1995))
9. cpmd.org, http://www.cpmd.org/
10. Earl, D.J., Deem, M.: Parallel tempering: Theory, applications, and new perspectives. Phys. Chem. Chem. Phys. 7, 3910–3916 ((2005))
11. F, B., M, B., M, P.: Ab initio simulation of rotational dynamics of solvated ammonium ion in water. J Am. Chem. Soc. 121, 10883 ((1999))

12. Fitch, B.G., Rayshubskiy, A., Eleftheriou, M., Ward, T.J.C., Giampapa, M., Pitman, M.C.: Blue Matter: Approaching the Limits of Concurrency for Classical Molecular Dynamics. In: SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing. ACM Press, New York, NY, USA (2006)
13. Gygi, F., Draeger, E.W., Schulz, M., de Supinski, B.R., Gunnels, J.A., Austel, V., Sexton, J.C., Franchetti, F., Kral, S., Ueberhuber, C.W., Lorenz, J.: Large-scale electronic structure calculations of high-z metals on the bluegene/l platform. In: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing. SC '06, ACM, New York, NY, USA (2006), `http://doi.acm.org/10.1145/1188455.1188502`
14. Gygi, F., Draeger, E.W., Schulz, M., Supinski, B.R.D., Gunnels, J.A., Austel, V., Sexton, J.C., Franchetti, F., Kral, S., Ueberhuber, C., Lorenz, J.: Large-Scale Electronic Structure Calculations of High-Z Metals on the Blue Gene/L Platform. In: Proceedings of the International Conference in Supercomputing. ACM Press (2006)
15. Hoefler, T., Snir, M.: Generic topology mapping strategies for large-scale parallel architectures. In: Proceedings of the international conference on Supercomputing. pp. 75–84. ICS '11, ACM, New York, NY, USA (2011)
16. IBM Blue Gene Team: Overview of the IBM Blue Gene/P project. IBM Journal of Research and Development 52(1/2) (2008)
17. Kale, L.V., Zheng, G., Lee, C.W., Kumar, S.: Scaling applications to massively parallel machines using projections performance analysis tool. In: Future Generation Computer Systems Special Issue on: Large-Scale System Performance Modeling and Analysis. vol. 22, pp. 347–358 (February 2006)
18. Kumar, S., Shi, Y., Bohm, E., Kale, L.V.: Scalable, fine grain, parallelization of the car-parrinello ab initio molecular dynamics method. Tech. rep., UIUC, Dept. of Computer Science (2005)
19. Lee, H.S., Tuckerman, M., Martyna, G.: Efficient evaluation of nonlocal pseudopotentials via euler exponential spline interpolation. Chem. Phys. Chem. 6, 18271835 (2005)
20. Marx, D., Parrinello, M.: Ab initio path integral molecular dynamics. Z. Phys. B 95, 143 ((1994))
21. Payne, M.C., Teter, M.P., Allan, D.C., Arias, T.A., Joannopoulos, J.D.: Iterative minimization techniques for ab initio total-energy calculations: molecular dynamics and conjugate gradients. Rev. Mod. Phys. 64, 1045 (1992)
22. Rosi, N.L., Eckert, J., Eddaoudi, M., Vodak, D.T., Kim, J., O'Keeffe, M., Yaghi, O.M.: Hydrogen storage in microporous metal-organic frameworks. Science 300(5622), 1127–1129 (2003), `http://www.sciencemag.org/content/300/5622/1127.abstract`
23. Vadali, R.V., Shi, Y., Kumar, S., Kale, L.V., Tuckerman, M.E., Martyna, G.J.: Scalable fine-grained parallelization of plane-wave-based ab initio molecular dynamics for large supercomputers. Journal of Comptational Chemistry 25(16), 2006–2022 (Oct 2004)