Towards Massively Concurrent Adaptive Sampling
for Molecular Dynamics

Jesús A. Izaguirre and Douglas Thain, University of Notre Dame, USA
Eric Darve, Stanford University, USA
Corresponding author: izaguir@nd.edu

October 8, 2015

1. Concurrent Adaptive Sampling

Traditional molecular dynamics (MD) codes are devised for single, long simulations. A complementary approach to study dynamics of bio-molecules is to perform ensembles of shorter simulations, and to create statistical models of the dynamics. Examples of these approaches include Markov State Models (MSM) and the Weighted Ensemble (WE). MSM is however limited by the inherent statistical bias that results from the Markovian assumption. In addition, currently, there is no technique to efficiently build the Markov states a priori. This is one of the main limiting factors of this class of methods.

We are pursuing the implementation of massively concurrent algorithms in elastic, distributed environments. Concurrent Adaptive Sampling (CAS) is a highly concurrent ensemble algorithm for computing dynamics of proteins and other bio-molecules. This approach allows making long time-scale predictions (slow rates and slow conformational changes leading to new metastable states) using a large ensemble of short trajectories. A central aim in this work is the creation of advanced statistical and adaptive space partitioning algorithms.

Combined with a powerful new parallel computing paradigm, this has the potential to enable truly long time scale simulations, much beyond what can be accomplished today. The main features of this new method, called Concurrent Adaptive Sampling (CAS), are as follows:

1. The sampling is always at equilibrium. This means that the observed trajectories are physical and correspond to real processes in the proper statistical ensemble (e.g., canonical ensemble).

2. The method scales even to large number of collective variables (CVs). Contrary to many other enhanced sampling methods, we are not limited by the number of CVs being used, as the computational cost does not increase dramatically with the number of CVs. In many other accelerated MD methods, the cost grows exponentially with the number of CVs.

3. The method adaptively chooses the macro-states to use. Such states correspond to chemically significant transition processes between metastable states.

4. The method is ideally suited to massively parallel computing on a grid or in the cloud because it relies on running a large number of short trajectories, rather than a few very long trajectories as is often done in MD.
2. Preliminary results of folding the FIP35 WW Domain

The co-authors have developed an earlier version of CAS called Accelerated Weighted Ensemble — Work Queue or AWE-WQ. AWE-WQ was shown to scale to 3,000 nodes of CPU and GPU clusters in an application to the folding of the 34 residue FIP35 WW Domain. Unlike CAS, AWE works with a static partitioning. AWE-WQ is built with Work Queue, a master worker framework and it is elastic: it allows pooling of heterogeneous resources, scalability to over 3000 workers, and ability to cope with failures of both worker and master processes.

This can be seen in Figure 2b. Over 1.5 ms of simulation was collected over 8 months with peak aggregate performance of 1000 ns/h. Comparison was done with a 200 µs brute force simulation collected on GPUs over a similar timespan. The folding and unfolded rates were of comparable accuracy: 1.5±0.3 µs⁻¹ for AWE-WQ and 1.8 ± 1.0 µs⁻¹ for brute force. Figure 2a shows folding pathways which correspond to experimentally observed ones.
3. Scaling Up with the Work Queue Framework

To implement the massive concurrency needed by the CAS algorithms, we build upon the Work Queue (WQ) distributed application framework. WQ allows us to achieve scale by harnessing multiple cluster, cloud, and grid systems simultaneously, while making the system transparently robust to common failure modes.

From the application developer’s perspective, Work Queue is merely a library with a straightforward submit-wait task interface. The main application is known as a master which submits tasks to a queue, where they are asynchronously dispatched to remote workers. Each task is a command line to be executed, annotated with the files and other dependencies that it requires. As such, each task can be an invocation of any standard MD code such as GROMACS, NAMD, or OpenMM. As tasks are completed, they are returned to the master application which may consume the results and submit more tasks. The system is natively implemented in C but has bindings for Perl, Python, and other high level languages which are commonly used in scientific computing. Figure 3 shows the core API and basic architecture of Work Queue.

```
queue = wq_create ( details );
task = wq_task_create ( details );
taskid = wq_submit ( queue, task );
task = wq_wait ( queue, timeout );
wq_remove ( queue, taskid );
wq_delete ( queue );
```

**Figure 3: Work Queue API and Architecture**

While the API is simple, the implementation is quite sophisticated. Generalized workers can be distributed to any computational resource to which the user has access: local clusters, national infrastructure, public clouds, etc. The workers join together into a tree rooted at the application master, which is responsible for dispatching tasks and collecting the results. Concurrency is exploited at every level: tasks may be multi-threaded, workers may run multiple tasks, and workers join together into a tree of arbitrary depth. Storage available at each node is used to build up a cache of commonly-used data items, and to construct the environment expected by each task. In such an environment, failures are a normal occurrence, and so the framework handles failed tasks, crashed workers, and network disconnections. In this way, WQ is more robust than traditional HPC techniques such as MPI, making it suitable for long-running applications that span administrative boundaries.

Work Queue has been used in production for several years and has enabled the construction of a wide variety of novel applications. For example, Work Queue version 1 (WQ1) enabled biometrics image comparison\(^5\) and scalable genome assembly on O(1K) cores\(^6\). WQ2 added multi-level hierarchy and enabled the Accelerated Weighted Ensemble (described above) on O(3K) cores\(^7\) and the Lobster HEP data analysis system running on O(10K) cores\(^8\). As an open source product\(^9\), it has attracted a community of users and developers and has been used as a vehicle for teaching parallel programming at Notre Dame, the University of Wisconsin, and the University of Arizona.
4. Conclusion

We present an unbiased approach to accelerating the computation of kinetic quantities using molecular dynamics simulations. This approach is superior to Markov State Models, which are biased, but it shares with them the ability to achieve massive concurrency. To deliver the necessary degree of concurrency, we make use of the Work Queue application framework. By combining algorithmic and computational capabilities, we aim to scale over the state-of-the-art by four to six orders of magnitude.

References


