Facilitating the Scalability of ParSplice for Exascale Testbeds

Vinay B. Ramakrishnaiah[1], Jonas L. Landsgesell[2], Ying Zhou[3], Iris Linck[4], Mouad Ramil[5], Joshua Bevan[6], Danny Perez[7], Louis J. Vernon[7], Thomas D. Swinburne[7], Robert S. Pavel[7], and Christoph Junghans[7]

[1]UWYO, Laramie, WY, USA, [2]Univ. of Stuttgart, Stuttgart, Germany ,[3]Loughborough University, Leicestershire, UK, [4]CU Denver, Denver, CO, USA, [5]École des ponts ParisTech, Champs-sur-Marne, France, [6]UIUC, Champaign, IL, USA, [7]LANL, Los Alamos, NM, USA.

I. Introduction

Parallel trajectory splicing, or ParSplice [1], is an attempt to solve the enduring challenge of simulating the evolution of complex atomistic systems over long time scales. Conventional molecular dynamics (MD) suffer from time scale limitations; therefore, typical simulations can only be performed for durations on the order of nanoseconds. This hinders the physical insights that can be provided by the simulations for slow processes because the system often remains trapped in the same region of configuration space for extended periods of time. The problem can be alleviated using accelerated-MD (AMD) methods [2]. These methods enable MD-quality simulations that can exceed microseconds. A recent generalization of parallel replica dynamics, parallel trajectory splicing (ParSplice), aims at improving the performance of AMD method for systems with heterogeneous distributions of barriers. ParSplice addresses this problem by parallelizing the generation of long trajectories in a time-parallel fashion and by employing a speculative execution strategy. We present the preliminary results of our attempts to enhance the scalability of ParSplice.

II. Parallel Trajectory Splicing

Within the ParSplice framework, one can show that a sequential MD trajectory can, to an excellent and controllable approximation, be decomposed into a number of independent segments. ParSplice makes use of this property to predictively and concurrently generate segments in many states using multiple MD instances. In a subsequent step, these segments are spliced together to form the trajectory.

The current ParSplice implementation consists of N producers that complete requests for segments that begin in states specified by a predictor process. The predictor keeps track of the states that were previously visited and constructs a Markov chain to predict the states that are likely to be visited soon, and hence the optimal set of segments that should be generated. The implementation also consists of a Splicer that manages the database, and generates the trajectory by splicing the segments from the database. The workers that generate the MD segments are grouped under work managers. Since most of the functions in ParSplice can be performed asynchronously, the splicer, the databases, and the work manager are executed in parallel on different MPI processes.

The performance of ParSplice relies on two main factors: i) the generation of a large number of segments, and ii) the availability of good predictions of the set of segments that are most likely to be incorporated into the trajectory. Therefore, we take a two-pronged approach of exploiting massive parallelism using heterogeneous architectures for large-scale segment generation, and improving the accuracy of the predictor such that the generated segments are more likely to be used shortly after their generation.

III. Improving Predictor Performance

The predictor builds a Markov chain model of the dynamics based on the previously generated segments and performs Kinetic Monte Carlo simulations (KMC) to predict the next probable states along the trajectory. While the previous implementation used only segments to parameterize the Markov chain, transitions from state to state are typically strongly accelerated at higher temperature. Therefore, we propose to also use statistics produced by a few workers that are performing runs at one or more elevated temperatures. While these high-temperature runs cannot be spliced into the generated trajectory, these typically discover new states faster so that these can be added to the predictor model even before they are visited by the segment-generating simulations. In fact, the expected number of transitions between two states i and j during a unit time at low temperature is $N_{i,j}^{L} = N_{i,j}^{H} e^{(\beta_H - \beta_L)E_{i,j}}$, where $\beta = \frac{1}{k_B T}$ is proportional to the inverse temperature and E is the transition barrier between the two states. This estimate for the number of times a state is visited is incorporated in the KMC predictor so that it becomes aware of states that were not yet discovered at lower temperatures. This approach allows for more speculation which is key for a scalable predictor. It avoids the generation of a large number of excess segments since workers already can be dispatched to gather low temperature statistics on new states.

Proper choice of temperatures and computation of the energy barriers should result in improved probabilities of the

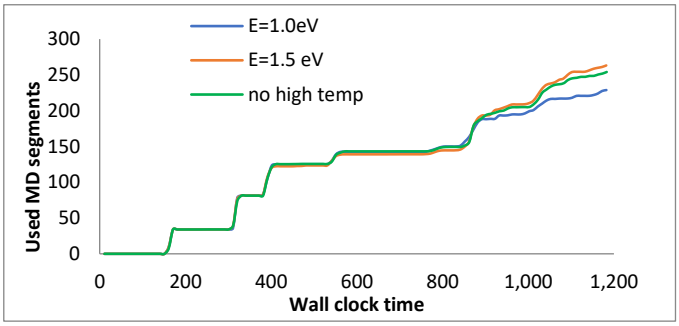

Fig. 1: Average number of MD segments used vs Wall clock time for different assumed energy barriers compared to the run which does not use high temperature statistics.

LA-UR-17-26401

segments spliced into the trajectory in the near future of their generation. Under the assumption that the probability of finding a certain number of transitions of type i is Poisson-distributed along with the fact that transitions at different temperatures are independent, we obtain that the most likely value of E solves for $\frac{\sum_{\{\beta_i\}} N_i \beta_i}{\sum_{\{\beta_i\}} N_i} - \frac{\sum_{\{\beta_i\}} \beta_i t e^{-\beta_i E_i}}{\sum_{\{\beta_i\}} t\, e^{-\beta_i E_i}} = 0$. Of course, the barrier can often be computed directly with advanced simulation methods, but this is expensive and adds to the complexity of the code.

As a proof of concept that incorporating high temperature statistics into the KMC predictor model will improve performance we present the results for incorporating high temperature runs assuming a constant energy barrier $E_{i,j} = E$ (i.e., this does not yet use the expression derived above to infer $E_{i,j}$). We demonstrate in Fig. 1 that in principle there is a speedup possible using additional high temperature results in the KMC predictor model. The actual deployment of this strategy in production will require the development of strategies for the optimal assignment of high temperature simulations. This work is ongoing.

For large numbers of workers, the execution time of the predictor may prove to be a bottleneck. Therefore, the predictor was factored out from the main ParSplice process into a dedicated MPI process. This eliminates computations that have to be performed by the splicer, and therefore might make the predictor scalable. Along with this, a hybrid MPI+MT model was employed to receive the statistics and update the Markov model in parallel.

IV. Exploiting Latent Parallelism

The ParSplice code is written in C++ and is built on the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) [3] framework. The hardware used to test the ParSplice algorithm consisted of Intel Haswell nodes. Intel's VTune Amplifier was used to analyze the code, which showed that upwards of 90% of the execution time is spent on generating the MD segments using LAMMPS function calls. As a first step in improving the overall performance of ParSplice by tailoring the code to the hardware, compiler optimizations were used to generate auto-vectorized AVX2 instructions. The KMC predictor in ParSplice was improved using the hybrid approach of message-passing + multi-threading (MP+MT) and the results are shown in Fig. 2.

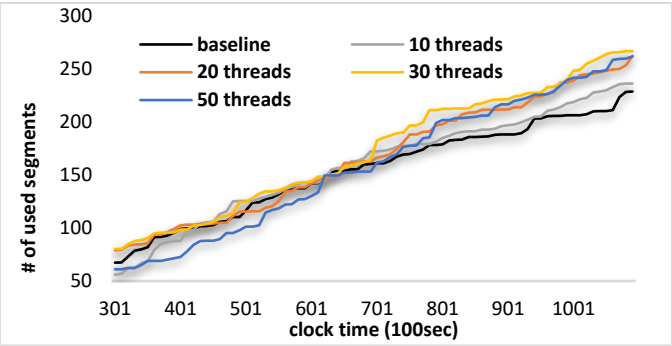

Fig. 2: Performance gain by using the MP+MT for the KMC predictor.

As most of the time is spent in computations using the LAMMPS framework, we explored the use of GPUs for accelerating the underlying MD simulations. The GPU package for LAMMPS [3] is capable of utilizing one or more GPUs coupled with one or more multi-core CPUs. NVIDIA K40m GPUs were used along with the Compute Unified Device Architecture (CUDA) application development platform. This led to the predictively spawned MPI processes to make use of GPUs and CPUs in each node to compute pair-wise interactions in molecules efficiently in a load balanced manner.

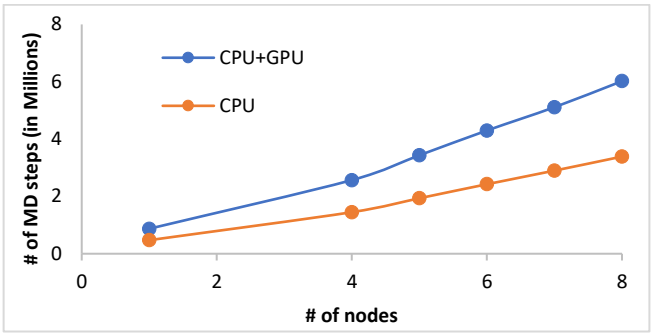

Fig. 2: Comparison of the number of MD segments generated using CPUs and CPUs+GPUs in a time duration of 20 min.

Fig. 3 shows the scaling plot of the total number of MD segments generated by CPU+GPU nodes versus only the CPU nodes. It is clear from the plot that CPU+GPUs give a performance improvement of ~2x while scaling over large number of nodes.

V. Conclusion and Future Work

A two-pronged approach of using heterogeneous architectures, and improving the efficiency of the predictor in ParSplice was explored. The use of different many core architectures like Intel Knights Landing (KNL) is currently being investigated. Optimized dynamic load balancing between different architectures, and the issue of inherent uncertainty in the prediction model is being addressed in order to improve the performance, as the current predictor only takes into account the previous observations to formulate the problem.

VI. References

[1] D. Perez, E. D. Cubuk, A. Waterland, E. Kaxiras and A. F. Voter, "Long-time dynamics through parallel trajectory splicing.," *Journal of chemical theory and computation*, vol. 12, no. 1, pp. 18-28, 2015.

[2] B. P. Uberuaga and A. F. Voter, "Accelerated molecular dynamics methods.," in *Radiation Effects in Solids*, Netherlands, Springer, 2007, pp. 25-43.

[3] "LAMMPS," Sandia National Laboratory, [Online]. Available: http://lammps.sandia.gov.