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.

Introduction

Parallel trajectory splicing, or ParSplice, 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.
  - Typical simulations can only be performed for durations on the order of nanoseconds.
  - Hinders physical insights.
- Alleviated using accelerated-MD (AMD) methods.
- ParSplice aims at improving the performance of AMD methods for systems with heterogeneous distributions of barriers.
- Parallelize the generation of long trajectories in time-parallel fashion.
- Employ speculative execution strategy.

The preliminary results of our attempts to enhance the scalability of ParSplice are presented in this poster.

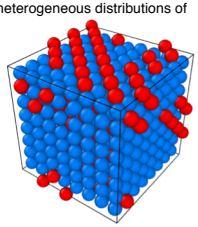

Fig. 1: Time evolution of UO_2 crystal simulated using ParSplice.

Parallel Trajectory Splicing

- Conventional trajectory can be decomposed into segments
- ParSplice uses this property to concurrently generate segments
- The segments are spliced together to form a trajectory
- Current implementation:
  - Producers complete requests for segments
  - Splicer uses Markov chain based predictor to preemptively schedule production of segments
  - Segments are stored in a database

MPI parallelization

- Most functions are asynchronous
- The splicer, databases, and work manager are executed in parallel processes
- Work manager spawns parallel workers
- Good weak scaling, but relatively poor strong scaling

Motivation

Large number of atoms can be simulated, but the temporal reach of MD is limited. The performance of parsplice can be further improved by

- Efficient prediction of the MD trajectory
- Large scale MD segment generation

Two-pronged approach to solve the problem

- Improve efficiency of predictor to effectively utilize the generated segments in the near future of their generation
- Exploit massive parallelism using heterogeneous architectures for large scale segment generation

Improving Predictor Efficiency

The predictor builds a Markov chain based on the previously visited states by the MD segments and performs a Kinetic Monte Carlo (KMC) analysis to predict the next probable state of the trajectory.

- Physics can be accelerated using elevated temperature
- Assign a fraction of workers to perform ParSplice runs at an elevated temperature
- Update KMC predictor using elevated temperature runs
- Statsitics are weighted according to $N_L = N_H e^{(B_H - B_L)E}$
- Proper choice of temperatures and energy barriers resulted in improved probabilities of the segments spliced into the trajectory as shown in Fig. 2

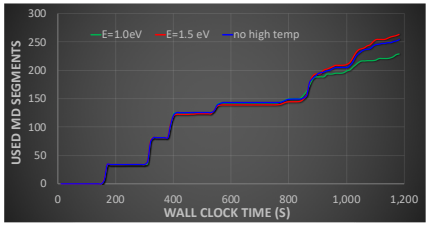

Fig. 2: Used MD segments vs Wall clock time for different assumed energy barriers compared to run which does not use high temperature statistics.

- Incorporate Bayesian estimator that takes the inherent model uncertainty into account (Fig. 3)

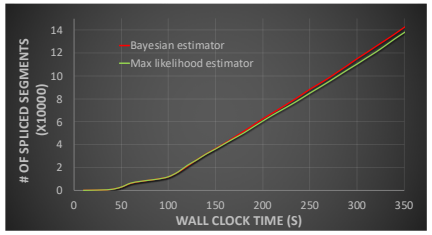

Fig. 3: Comparison of the number of spliced segments using Bayesian estimator vs. Maximum likelihood estimator.

Exploiting Latent Parallelism

The ParSplice code is written in C++ and is built on the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) software framework.

- Hardware platform used for testing: Intel Haswell (E5-2660_v3) nodes on Darwin cluster at the Los Alamos National Laboratory
- Intel Vtune Amplifier showed that more than 90% of the execution time is spent in LAMMPS function calls

The following optimizations were performed to tailor the code to the hardware:

- Compiler optimizations to generate auto-vectorized AVX-2 instructions (Fig. 4)
- GPUs to accelerate the underlying MD simulations (Fig. 5)
  - Hardware: Intel Haswell (E5-2660_v3) + NVIDIA K40m nodes

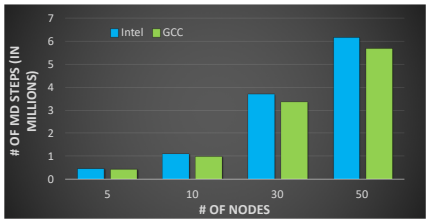

Fig. 4: Improvement by auto-vectorization using the Intel compiler compared to GNU C compiler.

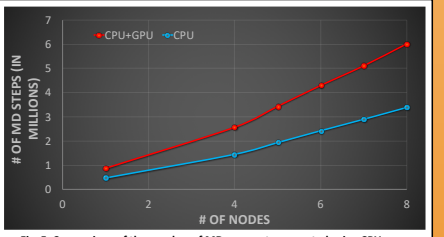

Fig. 5: Comparison of the number of MD segments generated using CPUs and CPUs+GPUs in a time duration of 20 min.

- Improved KMC predictor using message-passing + multi-threading (Fig. 6)

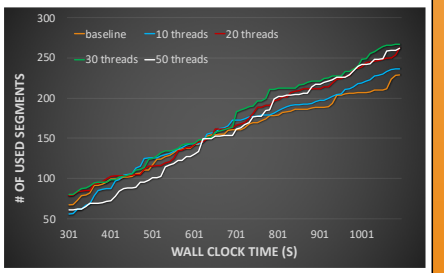

Fig. 6: Performance gain by using the hybrid approach of message-passing+multi-threading the KMC predictor.

Conclusions

A two-pronged approach of using heterogeneous architectures, and improving the efficiency of the predictor in ParSplice was explored and currently we are investigating:

- Use of different many-core architectures like Intel Knights Landing (KNL).
- Optimized dynamic load balancing between different architectures.
- Issue of inherent uncertainty in the prediction model, as the current predictor only takes into account the previous observations to formulate the problem.

*vramakr1@uwyo.edu

LA-UR-17-26402