Abstract

Prediction of nuclear phenomena is significant to the fields of astrophysics, nuclear medicine, and nuclear engineering. These simulations rely on computational models and depend heavily on high-performance computing (HPC) to calculate nuclear properties that cannot be measured experimentally. Quantum Monte Carlo (QMC) methods, specifically Auxiliary Field Diffusion Monte Carlo (AFDMC), have recently proven successful in computing ground-state properties of light-mass and medium-mass nuclei and are projected to scale up to heavy nuclei.

We investigate the application of OpenMP to AFDMC. We present a detailed description of how a portable hybrid parallelization model may be implemented for many-thread architectures, and describe the dangers of improperly scoped variables to OpenMP.

Physics Background

In low-energy nuclear structure models, nuclei are described as a collection of interacting protons and neutrons. The properties of that system, such as radioactivity and resonances, are determined in part by the mass and the interaction model.

Depending on the size of the nucleus, different methods are used to model these properties. Among these, Quantum Monte Carlo (QMC) has proven successful and robust with respect to problem size.

A version of the Auxiliary Field Diffusion Monte Carlo (AFDMC). Using AFDMC, we are able to solve for a ground state wavefunction, which can be used to compute observables such as radius, density, decay rates, and more.

AFDMC was designed for large-scale distributed terascale HPC systems. It currently uses MPI to distribute the work across each rank and periodically synchronizes for load-balancing. However, AFDMC using MPI only is less efficient on current many-core petascale systems with increased on-node parallelism through increased cores and threads. OpenMP can use shared memory which can be much more efficient for accelerated architectures.

A hybrid model (MPI/OpenMP) is possible in AFDMC: implementing fine-grain parallelization with OpenMP will allow the code to efficiently use the on-node parallelism of the Intel Xeon Phi 'Knights Landing' (KNL) processor and GPU processors like the NVIDIA P100.

Using new directives available in OpenMP 4.5, specifically the OMP Target, AFDMC can achieve performance portability across these two architectures with a single code.

Motivation

The MPI-only AFDMC version cannot take full advantage of the 272 hardware threads on a KNL.

Runtimes (LEFT) of constant problem-size on 272 ranks distributed over N ranks.

- The code benefits from vectorization, which scales well
- X1 rank per core causes inefficient use of thread architecture

Conclusions

We present a code design concept for the development of the AFDMC and similar codes in two steps: scaling structure and portability in OpenMP.

1. Scoping of data structures is complex but crucial to efficient (thread-safe) OpenMP

Fortran/C++ "Objects" must be redefined in AFDMC such that the entire Object can be either private or shared. Currently, the improper data scoring causes race conditions between threads trying to update large Object data structures.

Example: AFDMC uses a Fortran module called PSICALC. Within PSICALC, a large Fortran Object structure CORPSI is defined. Importing of PSICALC at high levels forces CORPSI to be shared between threads. It is difficult to avoid race conditions for components that need to be private. The solution will require redefining the Object in both structure and scope.

2. To achieve performance portability, we propose the use of a OpenMP 4.5 Target directive.

OMP Target defines a data environment (variables and instructions) that is contained, packaged at compile time, and can be delivered to the device on node.

OMP Target will port to KNL and GPU similarly. The Target will consist of the entire data environment of a stack, so each MPI rank will carry one Target. At each step, the Target will be delivered to the device; propagation will be done on device, and the resulting walkers are returned to host.

We anticipate improved efficiency in AFDMC through a restructuring of object data structures and the use of a OMP Target to propagate stacks of walkers.

References

Acknowledgments

We present an efficient implementation of the AFDMC code using MPI and OpenMP. The code was ported to Knights Landing (KNL): 2nd Generation Intel® Xeon Phi™ Processor.