Proposal for the Demonstration of a Shared Memory Implementation of MPI for Parallel Monte Carlo Calculations

Brian Cornille

March 15, 2015

In the MPI-3.0 Standard important improvements and extensions have been included to allow for shared memory parallel programming alongside the existing model. Utilizing this programming model will have several benefits over models that only provides support for shared memory parallel computing or distributed parallel computing. With a shared memory only programming model, such as OpenMP, computations are limited to shared memory hardware. For extremely large Monte Carlo calculations, this does not provide sufficient computational resources. In distributed memory programming models, the computational mesh or domain is usually distributed across or repeated for each process. In the case of Monte Carlo calculations, each process must be aware of the complete geometry and also keep track of tallies on this geometry. For these reasons, it is best to repeat this data on each process for these types of calculations. However, for Monte Carlo simulations that attempt to attain extremely large resolution for tallies, the memory requirements for this approach can be prohibitive. Advanced codes may take advantage of both approaches in order to leverage the maximum performance from HPC clusters. Previously, this required the use of multiple programming models (MPI+OpenMP for example), which could be somewhat difficult to implement. I plan to demonstrate shared memory usage in a Monte Carlo simulation using an MPI+MPI approach.

When multiple processes have access to the same memory, there is increased risk for programming error. The main concern in this scenario is that each process has no way of knowing when or in what manner other processes are accessing the same memory locations. If safety precautions are not taken to ensure that data is always up-to-date when a process wants to execute a save or load at some memory location, then what is called a *race condition* may occur. If a race condition exists in a program, then that program is very likely to produce incorrect results. Furthermore, the results can vary from run to run due to minor inconsistencies in the order with which processes may access the memory associated with the race condition. As one would expect, the MPI-3.0 shared memory model does not automatically provide safety against race conditions using basic load/save operations for the shared memory. I have been able to produce an example of this occurring using MPI. For generating tallies in Monte Carlo simulations many save (i.e. write) operations must be done on whatever data structure is used to store said tally. In order to utilize a shared memory data structure for tallies, a safe mechanism must be chosen to write calculated data. The MPI standard provides this functionality in a single function, MPI_ACCUMULATE. I have also produced an example program utilizing this function to demonstrate that it works as expected. This function will be core to creating a shared memory tally for Monte Carlo calculations.

I will write a simple Monte Carlo program that utilizes the shared memory capabilities of MPI. This program will be based on the principles of Monte Carlo methods that were taught in class. For the random number generator I plan to use the SPRNG library. This library produces reproducible, independent random number streams for each process in MPI. Since I am not interested in trying to reproduce particular physics with this code, I will try to stick to sampling analytically invertible PDFs that could represent some toy model. However, with good modular programming practices, the program should easily be able to be extended to arbitrary random sampling techniques for various possible processes. The program will be designed to handle 2^{nd} order analytic surfaces, although inputs may be limited to specific types based on what is needed for my toy model calculations. I have already written several proof-of-concept programs, which can be found in my git repository for this project: `https://github.com/bcornille/mpi-shared-ne506`. My progress and final program will also be accessible through this link. The major result for this project will be to demonstrate good parallel scaling of computations with a constant memory footprint for a given geometry/tally. In the best case scenario, this project could be used as a template for development of more advanced Monte Carlo tools that would like to take advantage of shared memory parallel programming.