MCB - A Monte Carlo Benchmark

Nick Gentile and Brian Miller

1 November 2011

1 Purpose of the Benchmark

Confirm correct hybrid MPI + OpenMP performance, single CPU performance, parallel scaling.

2 Characteristics of the Benchmark

The Monte Carlo Benchmark (MCB) is intended to model the computational performance of Monte Carlo algorithms on parallel architectures. It models the solution of a simple heuristic transport equation using a Monte Carlo technique. The MCB employs typical features of Monte Carlo algorithms such as particle creation, particle tracking, tallying particle information, and particle destruction. Particles are also traded among processors using MPI calls.

The heuristic transport equation models the behavior of particles that are born, travel with a constant velocity, scatter, and are absorbed. The solution of this heuristic transport equation is well approximated by a diffusion equation, which can be solved by a Fourier series. The MCB includes a function that returns this approximate solution so that the accuracy of the Monte Carlo simulation can be evaluated.

The particles in the MCB simulation do not interact with material by depositing energy, do not use physical cross sections, or allow the modeling of real transport effects such as frequency dependent properties or material motion corrections. The MCB is implemented on a simple orthogonal grid. Because of these limitations, the MCB is solely intended to serve as a benchmark and not intended to model real physics.

3 Building the Benchmark

MCB uses portions of the Boost class library `http://www.boost.org`. The variable **BOOST_PREFIX** in the Makefile must be defined to be the installed location of Boost.

There are also several compilation flags that must be defined in the Makefile to build the benchmark. Included in the Makefile.sample are several examples of compilation flags that are used to build the benchmark on LLNL machines. The list of compilation flags required to build the benchmark is: **CXX**, **CXXFLAGS**, **OPENMPFLAG**, **MPI_INCLUDE**, **LDFLAGS**, **LIBPATH**, **LIBS**. In particular, any include paths, library paths, library names and compiler flags needed to compile an MPI plus OpenMP code need be defined in this section of the makefile.

The code has been built using the XL C++ compiler on IBM Blue Gene and Power series computers as well as X86_64 Linux clusters using the GNU C++ compiler.

4 Running the Benchmark

The executable **MCBenchmark.exe** takes several command line arguments but only a few are necessary.

4.1 Required Arguments

--nMpiTasksX=px, **--nMpiTasksY=py**: where **px** and **py** are integers set the number of MPI tasks to use in each dimension of the domain decomposed 2D rectangular grid. The product of these task counts must equal the total number of MPI tasks requested at run time.

4.2 Optional Arguments

--weakScaling: takes no argument and if used will set each MPI task's portion of the domain decomposed mesh to have constant size independent of the number of tasks used to execute the benchmark, by default 100x100. If this command line argument is not used, a strong scaling study is assumed and the global mesh size will be by default 100x100. These default sizes can be overridden by **--nZonesX** and **--nZonesY**.

--nZonesX=nx, **--nZonesY=ny**: with **nx** and **ny** integers control the size of the 2D rectangular grid. In the weak scaling case, these values are used for the local

grid size for each MPI task. For the strong scaling case these values are the global grid size which is then decomposed into smaller chunks for each task.

--xDim=xMax, **--yDim=yMax**: with **xMax** and **yMax** float values control the physical size of the problem domain. For the weak scaling case, the 2D rectangular mesh covers the region $[0, \text{px} * \text{xDim}]\text{x}[0, \text{py} * \text{yDim}]$. For the strong scaling case, the 2D rectangular mesh covers the region $[0, \text{xDim}]\text{x}[0, \text{yDim}]$. The default values are **xDim**=2.0, **yDim**=2.0.

--sigmaA=sigA, **--sigmaS=sigS**: with **sigA** and **sigS** float values control the absorption and scatting constants, respectively. The default values are **sigA**=1.0, **sigS**=20.0.

--xSource=xr, **--ySource=yr**: with **xr** and **yr** float values determines the coordinates of the energy source. For the weak scaling case these arguments are ignored and the source is centered at $(\text{px} * \text{xMax}, \text{py} * \text{yMax})$. The default values are **xr**=1.0, and **yr**=1.0.

--numParticles=np: with **np** an integer value sets the number of particles initially created and solved in the benchmark. The default value of **np**=1000000.

--numCycles=nc: with **nc** an integer value sets the number of timesteps to take over the course of the run. The mode of operation specifying a fixed number of timesteps is for debugging purposes only. The default value of **nc**=1000.

4.3 Typical Execution Command

To run the code the user must set the number of OpenMP threads to use, determine the number of MPI tasks desired and choose a domain decomposition in x and y. For example using srun on an LLNL linux machine with 12 cores per node:

```
setenv OMP_NUM_THREADS 12

srun --nodes=4 --ntasks=4 --ntasks-per-node=1 --cpus-per-task=12 ./MCBenchmark.exe --nMpiTasksX=2 --nMpiTasksY=2 --weakScaling
```