MCB

Summary Version
1.0

Purpose of Benchmark

Study MPI+OpenMP parallel scaling efficiency, including the performance trade-off for switching between MPI processes and OpenMP threads. Verify C++ compiler support for OpenMP and templates. Evaluate performance for integer computations.

Characteristics of Benchmark

MCB is a Monte Carlo particle transport benchmark. As with any Monte Carlo benchmark, the number of branch mis-predictions will be higher than in a pure number crunching application. Some of the computation in MCB is performed using integer arithmetic to avoid issues associated with round-off of floating point numbers. Compilers are unlikely to find SIMDizable loops in the current version of MCB.

This benchmark uses both MPI and OpenMP to deliver high levels of parallelism. ASC applications often simulate many kinds of physics in a single run. This leads to large memory usage per process. The amount of memory per core is dropping, below the size of a process, so it is crucial that ASC can effectively use multiple threads within a single MPI process. On a system that is well suited to ASC workloads, MCB will exhibit little fall-off in performance between a pure MPI run and a run with 8 or more threads per process.

MCB requires a minimum of 1GB of main memory per MPI task.

Mechanics of Building MCB

The `mcb` directory contains a `README` file giving fairly detailed information on building and running MCB. This section gives a brief explanation on how to get MCB up and running on a new platform.

Create a build script and modify a `Makefile` to reflect the characteristics of your system. Samples named `build-linux-x86_64.sh` and `Makefile-linux-x86_64` for systems running ASC's variant of Red Hat Enterprise Linux 6 and `build-bgq.sh` and `Makefile-bgq` for IBM Blue Gene/Q systems have been provided in the `mcb` directory. The flags that most often change for a new system or compiler are `CXX, CXXFLAGS, OPENMPFLAG`, and `MPI_INCLUDE`. The meanings of these flags should be self-explanatory.

It should be easy to build MCB for an x86_64 cluster or a Blue Gene system. Building MCB for an Intel Xeon Phi system should also be easy if it runs in "native mode" on the accelerator board.

Code modifications will be required on systems using Nvidia boards or other hardware that does not support OpenMP. The changes will be sufficiently extensive that they cannot be described in this document.

Mechanics of Running MCB

MCB is a proxy app for ASC multi-physics simulation codes that require at least 1 GB of memory per MPI process. MCB runs used in CORAL bids should limit the number of MPI processes to no more than the number of GB of memory in the node, even if the MCB benchmark doesn't require that much memory per MPI process.

The `mcb/run-decks` directory contains scripts to run a variety of problems. The scripts with names of the form `M_mcb_coral_...csh` are intended for use with the Moab batch system (or the SLURM `sbatch` command) and the `srun` job launcher from SLURM. You will need to modify them if your system does not use Moab and `srun`.

To verify that MCB gives correct answers, run the `M_mcb_coral_BGQ_val.csh` script (modified for your system). The value printed at the end of the run for "MC max error" is 0.0025 for 32 nodes of a BGQ system with a total of 10240000 particles. The error should scale roughly like 1/sqrt(numParticles) when run with 2-50 million particles.

```
sbatch -p pdebug --nodes=32 --ntasks=128 --time=30:00
./M_mcb_coral_BGQ_val.csh
```

The scripts assume 16 cores per node and run 1, 2, 4, 8, and 16 processes per node to evaluate the trade-off between MPI processes and OpenMP threads. Evaluating the performance changes that occur when switching between MPI processes and OpenMP threads is one of the key goals of the MCB runs. If a node has 57 cores, it will be hard to trade threads for processes. In this case it would be best to only use 56 cores because 56 has enough small prime factors that several different process-by-thread combinations can be tried.

Once MCB runs correctly, you can run a weak scaling study to check for scaling issues. `M_mcb_coral_BGQ_4ht_lo.csh` and `M_mcb_coral_x86_1ht_lo.csh` can be run with any number of nodes and will automatically change the number of particles and the spatial decomposition to make implement weak scaling. The BGQ script uses 4 hardware threads per core while the x86 script uses 1 hardware thread per core. The weak scaling study assigns a fixed amount of work per core, not per hardware thread. The scripts assume that `PROCS_PER_NODE` evenly divides the number of cores per node. It will be necessary to modify `CORES_PER_NODE` and `THREADS_PER_CORE` in the batch script for systems other than a BGQ. `NUM_NODES` can be set manually if the system does not use SLURM.

The MCB problems that will be run to generate a CORAL response use several command line arguments to "nail down" the problem size. Changing the number of cores does NOT change the amount of work. The CORAL baseline Figure of Merit problem can be run by typing:

```
sbatch -p pbatch --nodes=4096 --ntasks=65536 --time=30:00
./M_mcb_seq_base.csh
```

The MCB run requested in CORAL responses (CORAL class problem) uses twice as many zones and twice as many particles as the baseline problem. This CORAL class problem can be run by typing:

```
sbatch -p pbatch --nodes=4096 --ntasks=65536 --time=30:00
./M_mcb_coral_2xseq.csh
```

Reporting Results

MCB prints a FOM near the end of each run. The FOM is proportional to the total number of particles tracked per second. In the absence of parallel overhead, the FOM will scale linearly with the number of cores used. The FOM, the number of nodes, the total number of MPI processes, the number of OpenMP threads per process, and the total number of particles (the numParticles value printed near the start of the run) should be reported.

MCB includes a validation test whose answer can be compared to an analytic diffusion problem. The value for "MC max error" from the validation run should be reported along with the number of nodes, total number of MPI processes, the number of OpenMP threads per process, and the total number of particles.

The README file provides more details on building and running MCB.