Quicksilver

Summary Version

1.0 (Commit 07b62cb in the Quicksilver public repo)

Purpose of Benchmark

Test performance of Monte Carlo Transport methods.

Characteristics of Benchmark

Quicksilver is a proxy app that represents some elements of the Mercury workload by solving a simplified dynamic Monte Carlo particle transport problem. Quicksilver attempts to replicate the memory access patterns, communication patterns, and the branching or divergence of Mercury for problems using multigroup cross sections. OpenMP and MPI are used for parallelization. A GPU version is available.

Performance of Quicksilver is likely to be dominated by latency bound table look-ups, a highly branchy/divergent code path, and poor vectorization potential.

A paper showing performance on modern hardware, discussing the representativeness of Quicksilver to its parent code Mercury, and describing the changes needed to port the original version of Quicksilver to GPUs can be found here.

Building the Benchmark

Instructions to build Quicksilver can be found in the Makefile. Quicksilver is a relatively easy to build code with no external dependencies (except MPI and OpenMP). You should be able to build Quicksilver on nearly any system by customizing the values of only four variables in the Makefile:

- CXX The name of the C++ compiler (with path if necessary) Quicksilver uses C++11 features, so a C++11 compliant compiler should be used.
- CXXFLAGS Command line switches to pass to the C++ compiler when compiling objects *and* when linking the executable.
- CPPFLAGS Command line switches to pass to the compiler *only* when compiling objects
- LDFLAGS Command line switches to pass to the compiler *only* when linking the executable

Sample definitions for a number of common systems are provided.

Quicksilver recognizes a number of pre-processor macros that enable or disable various code features such as MPI, OpenMP, etc. These are described in the Makefile. In particular, the Quicksilver benchmark can be built for GPUs with either CUDA or OpenMP 4.5. Instructions to build these versions are given in the Makefile. Unified memory is assumed. Note that OpenMP is not needed for the CTS benchmark problem.

The code is of late beta quality with no current known bugs, but no promises that none exist. Performance of the code is likely sub-optimal as little work has been done to tune the code for GPUs.

Running the Benchmark

Quicksilver's behavior is controlled by a combination of command line options and an input file. All of the parameters that can be set on the command line can also be set in the input file. The input file values will override the command line. Run `$ qs -h` to see documentation on the available command line switches.

The Examples/CTS2_Benchmark directory contains the benchmark problem. In that directory you will find:

- CTS2_1.inp is a 1-rank problem
- CTS2_36.inp is a 36-rank problem
- CTS2.inp is a scale independent problem that contains only the description of the problem physics. ***When using this file, parameters related to the size of the problem must be specified on the command line***.
- CTS2_scaling.sh shows a weak scaling study from 1 to 36 ranks.

For example, a 16-rank version of the problem with a 4x2x2 domain decomposition is executed with the command line:

```
$ qs -i CTS2.inp -X64 -Y32 -Z32 -x64 -y32 -z32 -I4 -J2 -K2 -n655360
```

Outputs

Quicksilver writes its main diagnostic output to stdout. This output includes a complete record of the parameters that were used for the run. If you capture this output in a file, it will be a valid Quicksilver input file that will repeat the run that produced the output. Lines in the output that are unrelated to setting problem parameters are automatically ignored by the input parser.

After the problem parameters and a few lines that report the progress of problem initialization, Quicksilver prints one line of output per time step. These lines contain values of various tallies as well as some performance information. The meanings of the columns in the output are explained below (columns in bold contribute directly to the FOM).

cycle:	The time step number
start:	The number of particles present at the start of the time step
source:	The number of particles created by sources for the time step
rr:	The number of particles destroyed by a "Russian Roulette" algorithm when there are more particles than the target number
split:	The number particles created by a splitting algorithm when there are fewer particles than the target number
absorb:	The number of segments that resulted in an absorb reaction

scatter: The number of segments that resulted in a scatter reaction
fission: The number of segments that resulted in a fission reaction
produce: The number of particles created by fission reactions
collision: The sum of absorb, scatter, and fission
escape: The number of particles that escape the problem domain
census: The number of particles that make it to census (the end of the time step)
num_seg: The number of segments completed during the time step
scalar_flux: The value of the scalar flux tally
cycleInit: The time spent in cycleInit (in seconds)
cycleTracking: The time spent in cycleTracking (in seconds)
cycleFinalize: The time spent in cycleFinalize (in seconds)

After all time steps are complete, the values of various timers are reported along with the Figure of Merit, and the results of the verification tests are printed.

Requirements for Sizing and Scaling Problems

The benchmark problem is a homogenous domain with reflective boundary conditions, so it can be easily scaled to an arbitrary size.

- You must use 16x16x16 = 4096 mesh elements per rank.
- You must use 40960 particles per rank.
- You must keep a constant mesh element size by increasing the size of the simulation domain (-X, -Y, and -Z) proportionally to the number of mesh elements (-x, -y, and -z). For the CTS2 benchmark problem the mesh element size is one unit. This means the size of the simulation domain must be equal to the number of mesh elements in each direction.
- When weak scaling, increase both the number of mesh elements and the number of particles.
- You should specify an explicit domain decomposition with the -I, -J, and -K command line options (or the xDom, yDom, and zDom parameters). Quicksilver can create a decomposition without these flags, but the load balance will be very poor.
- To keep the problem load balanced, choose the number of mesh elements in any direction such that it is evenly divisible by the number of MPI ranks in that direction. (I.e., the argument of –x should be divisible by the argument of –I.)

Figure of Merit (FOM)

The figure of merit for Quicksilver is printed automatically at the end of each run. It is calculated by dividing the total number of Monte Carlo segments executed on all threads on all ranks by the total (wall clock) runtime in the cycle_tracking function. A Monte Carlo segment is counted when a particle crosses a mesh facet, undegoes a reaction (scattering, absorption, or fission), or reaches the end of the time step (census). There is no normalization for the number of cores, ranks, etc. so more computing resources will produce a larger figure of merit.

Only cycle_tracking time is included in the FOM so all parts of the code outside of the cycle_tracking loop do not contribute to the FOM.

Benchmark Verification

Because Quicksilver is a Monte Carlo code it is difficult to rigorously verify. However, for this benchmark we have created some statistical tests that will catch many gross errors. These tests are computed automatically at the end of each run and PASS/FAIL results are printed. Successful runs should PASS all of these tests.

Figure of Merit Data

The figure of merit data shown here was calculated with between 1 and 36 ranks on a single node of the LLNL Quartz cluster. A Quartz node has 2 sockets of 18-core Intel Xeon E5-2695 v4 (Broadwell) per node.

Ranks	FOM
1	9.668e+05
2	1.900e+06
4	3.659e+06
8	6.425e+07
16	1.063e+07
32	1.982e+07
36	2.131e+07

Table 1: Figure of merit data on LLNL Quartz.

Change Log

- 30-Mar-19 Initial version

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.