Quicksilver

Summary Version

1.02 (Commit 65091af 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.

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. Documentation of the input file parameters is in preparation.

The Examples/CORAL2_Benchmark directory contains two benchmark problems. Only Problem 1 is required. In the Problem1 directory you will find:

- Coral2_P1_1.inp is 1-node problem, sized for BG/Q
- Coral2_P1_4096 is a 4096-node problem, sized for BG/Q
- Coral2_P1.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***.
- P1_64t.sh shows a weak scaling study from 1 to 24576 nodes with 64 threads per node.
- P1_16t.sh shows a weak scaling study from 1 to 24576 nodes with 16 threads per node.
- P1_04t.sh shows a weak scaling study from 1 to 24576 nodes with 4 threads per node.

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 columns in the output are:

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.

Recommendations 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.

- cycle_tracking should take between 0.5 and 5 seconds per call. Choose the number of particles to obtain a time in this range.
- Problems should have at least 1000 mesh elements 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 Problem 1 the mesh element size is one unit. This means the size of the simulation domain must be equal to the number of mesh elements.
- 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.)
- At the start of each time step, Quicksilver will source a number of new particles equal to 10% of the target number of particles. For best results, chose the number of particles per mesh element such that it is divisible by 10. Quicksilver rounds down the number of source particle per elements. Hence, if the number of particles is set below 10 particles per mesh element Quicksilver will source no particles.

Batch Size Parameters

Quicksilver divides the total number of particles to be simulated on each rank into some number of batches. MPI communication of particles that cross into the spatial domains of other ranks is performed between batches. This communication is non-blocking to allow the next batch to proceed while the communication progresses. On GPUs, each batch is a single kernel launch. By default, particles are divided into 10 batches. Increasing the number of batches can provide more opportunities to overlap communication and computation, however it can also increase overhead due to kernel launches and data packing and unpacking. To help optimize Quicksilver's performance in different regimes we have made the number of batches a tunable parameter. We have found that 10 batches may be too few as the number of particles per rank increases.

There are two methods to set the number of batches. You can either set the number of batches directly using the -b command line switch or, you can specify the number of particles per batch using the -g switch. The latter option may provide more consistent performance when comparing runs with different number of particles. Both the number of batches and the number of particles per batch can be set in the input using the nBatches and batchSize parameters. If you attempt to set both the number of batches and the number of particles in a batch the number of particles per batch will take precedence.

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 on BG/Q

The figure of merit data shown here was calculated with either 1, 4, or 16 MPI ranks per node (and 64, 16, or 4 OpenMP threads per MPI rank, respectively) using the commands in the scripts provided in the Examples/CORAL2_Benchmark/Problem1 directory. The total number of threads per node is always 64. For example, the 32-node case with 64 threads per rank is

```
srun -N32 -n32 qs -i Coral2_P1.inp -X64 -Y64 -Z32 -x64 -y64 -z32 -I4 -J4 -
K2 --nParticles 5242880
```

Each run uses 4,096 mesh elements per rank and 40 particles per mesh element (163,840 particles per node). Note that the runs with 4 threads per rank (16 ranks per node) have only 256 mesh elements per rank and therefore violate the minimum of 1000 mesh elements per rank.

	1 Rank per Node 64 Threads per Rank		4 Ranks per Node 16 Threads per Rank		16 Ranks per Node 4 Threads per Rank	
Nodes	Ranks	FOM	Ranks	FOM	Ranks	FOM
1	1	1.292e+06	4	1.266e+06	16	1.215e+06
2	2	2.508e+06	8	2.465e+06	32	2.351e+06
4	4	4.832e+06	16	4.683e+06	64	4.531e+06
8	8	9.207e+06	32	8.933e+06	128	8.878e+06
16	16	1.764e+07	64	1.725e+07	256	1.751e+07
32	32	3.374e+07	128	3.398e+07	512	3.475e+07
64	64	6.386e+07	256	6.671e+07	1024	6.863e+07
128	128	1.260e+08	512	1.325e+08	2048	1.367e+08
256	256	2.492e+08	1024	2.630e+08	4192	2.711e+08
512	512	4.932e+08	2048	5.218e+08	8096	5.380e+08
1024	1024	9.737e+08	4096	1.032e+09	16384	1.072e+09
2048	2048	1.936e+09	8192	2.055e+09	32768	2.131e+09
4096	4096	3.852e+09	16384	4.088e+09	65536	4.228e+09
8192	8192	7.574e+09	32768	8.083e+09	131072	8.386e+09
16384	16384	1.507e+10	65536	1.619e+10	262144	1.678e+10
24576	24576	2.285e+10	98404	2.440e+10	393216	2.511e+10

Table 1: Figure of merit data for Problem 1 on Vulcan for 1, 4, and 16 ranks per node (weak scaling). Note that all cases use 64 threads per node since Vulcan has 16 cores per node with 4 hardware threads per core.

Change Log

- 26-Feb-18 Added FOM data for 4 ranks per node and reformatted FOM data into a single table. (1 rank per node data was previously Table 1 and 16 ranks per node was Table 2.)
- 26-Feb-18 Added section to explain batch size parameters (-g and -b switches).
- 26-Feb-18 Increased tolerance for fluence test from 5% to 6%. We received multiple reports of correctly working simulations with fluence variations between 5% and 6%.
- 26-Feb-18 Removed nBatches and batchSize parameters from all sample job files. In every case, the values set in the .inp files were the same as the default values. Removing the values from the .inp files allows the values to be set on the command line using the -g and -b command line switches.
- 18-Jan-18 Fixed incorrect units in FOM printout. The correct value was printed, only the units were wrong
- 18-Jan-18 Added timers to separately report time spent in the tracking kernel and time spent in activities related to MPI communication (including MPI calls and packing/unpacking of data).
- 18-Jan-18 This document updated with a description of Quicksilver output.
- 22-Jan-18 This document updated with references to published materials.