Testing the Random Number Generator in ‘cudaBayesreg’

Adelino Ferreira da Silva

April 16, 2010

This report presents a summary of the empirical results used to assess the quality of the random number generator (RNG) in ‘*cudaBayesreg’*. Two RNG test suites were used: DierHarder [1], [2], and TestU01 [3]. Four batteries of tests included in TestU01 were used: Rabbit, Alphabit, FIPS-140-2, and SmallCrush.

The code listed in Appendix C reproduces typical conditions of utilization of the RNG in ‘cudaBayesreg’. Specifically, the binary file used in all the tests but the SmallCrush suite was generated with the following parameters: (i) Number of threads: 50000; (ii) Number of iterations: 3000; (iii) Number of uniform random variables per thread: 20; (iv) grid of 391 blocks, and 128 threads per block. For the SmallCrush suite, a similar file of floating point random numbers was generated. The SmallCrush tests were performed in single-precision, since they were conducted on a notebook equipped with a NVIDIA ”GeForce 8400M GS” card having Compute Capability 1.1, which does not support double-precision.

The tests, summarized in Appendices A and B, produced the following results:

1. All DieHarder tests, and all the tests in ‘Rabbit’, ‘Alphabit’, and ‘FIPS-140-2’ were passed.
2. Three SmallCrush tests failed, namely 1-BirthdaySpacings, 6-MaxOft, and 7-WeightDistrib. The remaining 12 were passed.

1 References

References

[1] Robert G. Brown. *DieHarder: A Gnu Public Licensed Random Number Tester*. Durham, NC 27708-0305, Oct. 2009.

[2] Dirk Eddelbuettel and Robert G. Brown. *RDieHarder: An R interface to the DieHarder suite of Random Number Generator Tests.* Debian, 2007.

[3] PIERRE L’ECUYER and RICHARD SIMARD. TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, 33(4), 2007.

2 Appendix A

```
#=============================================================================#
#          dieharder version 3.29.4beta Copyright 2003 Robert G. Brown        #
#=============================================================================#
   rng_name    |           filename             |rands/second|
 file_input_raw|                       test1.bin|  6.36e+06  |
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|     100|0.81799801|  PASSED
      diehard_operm5|   5|   1000000|     100|0.22084207|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.62803018|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.18075902|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.22506789|  PASSED
        diehard_opso|   0|   2097152|     100|0.13278313|  PASSED
        diehard_oqso|   0|   2097152|     100|0.50763405|  PASSED
         diehard_dna|   0|   2097152|     100|0.15163072|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.72275236|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.32986639|  PASSED
 diehard_parking_lot|   0|     12000|     100|0.65319410|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.35325419|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.33776928|  PASSED
     diehard_squeeze|   0|    100000|     100|0.96632833|  PASSED
        diehard_sums|   0|       100|     100|0.00801640|  PASSED
        diehard_runs|   0|    100000|     100|0.81739071|  PASSED
        diehard_runs|   0|    100000|     100|0.76468475|  PASSED
       diehard_craps|   0|    200000|     100|0.05776067|  PASSED
       diehard_craps|   0|    200000|     100|0.53516474|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.43531443|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.87527353|  PASSED
         sts_monobit|   1|    100000|     100|0.77910996|  PASSED
            sts_runs|   2|    100000|     100|0.98074929|  PASSED
          sts_serial|   1|    100000|     100|0.40221928|  PASSED
          sts_serial|   2|    100000|     100|0.35277980|  PASSED
          sts_serial|   3|    100000|     100|0.69237310|  PASSED
          sts_serial|   3|    100000|     100|0.70701942|  PASSED
          sts_serial|   4|    100000|     100|0.97579139|  PASSED
          sts_serial|   4|    100000|     100|0.97830135|  PASSED
          sts_serial|   5|    100000|     100|0.05902037|  PASSED
          sts_serial|   5|    100000|     100|0.07379595|  PASSED
          sts_serial|   6|    100000|     100|0.17672985|  PASSED
          sts_serial|   6|    100000|     100|0.35557316|  PASSED
          sts_serial|   7|    100000|     100|0.83404102|  PASSED
          sts_serial|   7|    100000|     100|0.30742566|  PASSED
          sts_serial|   8|    100000|     100|0.68722937|  PASSED
          sts_serial|   8|    100000|     100|0.95814708|  PASSED
          sts_serial|   9|    100000|     100|0.42576930|  PASSED
          sts_serial|   9|    100000|     100|0.04868476|  PASSED
          sts_serial|  10|    100000|     100|0.48415397|  PASSED
          sts_serial|  10|    100000|     100|0.77526395|  PASSED
          sts_serial|  11|    100000|     100|0.31238275|  PASSED
          sts_serial|  11|    100000|     100|0.13217194|  PASSED
          sts_serial|  12|    100000|     100|0.93584421|  PASSED
          sts_serial|  12|    100000|     100|0.36189413|  PASSED
          sts_serial|  13|    100000|     100|0.83815401|  PASSED
          sts_serial|  13|    100000|     100|0.84725205|  PASSED
          sts_serial|  14|    100000|     100|0.13447819|  PASSED
          sts_serial|  14|    100000|     100|0.43578161|  PASSED
          sts_serial|  15|    100000|     100|0.09574604|  PASSED
          sts_serial|  15|    100000|     100|0.91983887|  PASSED
          sts_serial|  16|    100000|     100|0.38801373|  PASSED
          sts_serial|  16|    100000|     100|0.85853150|  PASSED
         rgb_bitdist|   5|    100000|     100|0.70368991|  PASSED
rgb_minimum_distance|   5|     10000|    1000|0.89821861|  PASSED
    rgb_permutations|   5|    100000|     100|0.12397983|  PASSED
      rgb_lagged_sum|   5|   1000000|     100|0.50363370|  PASSED
     rgb_kstest_test|   5|     10000|    1000|0.03669471|  PASSED
```

3 Appendix B

```
========= Summary results of Rabbit =========

 Version:          TestU01 1.2.3
 File:             test1.bin
 Number of bits:   33554432
 Number of statistics:  39
 Total CPU time:   00:00:29.51

 All tests were passed




========= Summary results of Alphabit =========

 Version:          TestU01 1.2.3
 File:             test1.bin
 Number of bits:   33554432
 Number of statistics:  17
 Total CPU time:   00:00:01.47

 All tests were passed




============== Summary results of FIPS-140-2 ==============

 File:             test1.bin
 Number of bits:   20000

      Test          s-value        p-value    FIPS Decision
 --------------------------------------------------------
 Monobit              9895           0.93       Pass
 Poker               11.22           0.74       Pass

 0 Runs, length 1:     2473                     Pass
 0 Runs, length 2:     1236                     Pass
 0 Runs, length 3:      606                     Pass
 0 Runs, length 4:      337                     Pass
 0 Runs, length 5:      163                     Pass
 0 Runs, length 6+:     163                     Pass

 1 Runs, length 1:     2511                     Pass
 1 Runs, length 2:     1230                     Pass
 1 Runs, length 3:      616                     Pass
 1 Runs, length 4:      313                     Pass
 1 Runs, length 5:      170                     Pass
 1 Runs, length 6+:     137                     Pass

 Longest run of 0:       17          0.07       Pass
 Longest run of 1:       19          0.02       Pass
 ----------------------------------------------------------
 All values are within the required intervals of FIPS-140-2




========= Summary results of SmallCrush =========

 Version:          TestU01 1.2.3
 File:             test1f.txt
 Number of statistics:  15
 Total CPU time:   00:01:28.45
 The following tests gave p-values outside [0.001, 0.9990]:
 (eps  means a value < 1.0e-300):
 (eps1 means a value < 1.0e-15):

       Test                          p-value
 ----------------------------------------------
  1  BirthdaySpacings                 eps
  6  MaxOft                           eps
  7  WeightDistrib                    eps
 ----------------------------------------------
 All other tests were passed
```

4 Appendix C

```
/*
 * Program "testrunifBinSeed" used to generate the binary file used in:
 *   Dieharder
 *   TestU01: bbattery_Rabbit ; bbattery_AlphabitFile ;  bbattery_FIPS_140_2File
 *
 * time testrunifBinSeed
 *
 * n. of required threads = 50000
 * dGrid = 391      dBlock = 128
 * real    6m1.567s
 * user    0m41.964s
 * sys     0m36.601s
 *
 * Size of "test1.bin": 11.2 GB
 *
 * time dieharder -a -n 5 -g 201 -f test1.bin  > log.dieharderSeed  &
 * real    12m40.746s
 * user    11m21.343s
 * sys     0m14.152s
 *
*/

#include <climits>
#include <cstdio>
#include <ctime>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
// includes, project
#include <cutil_inline.h>
#include "d_rngNR.cu"
```

```
#define BLOCK 128 // specify CUDA block size

__global__ void
runifK(Uint *ru, Uint p, int m)
{
int ti = blockIdx.x * blockDim.x + threadIdx.x;
Ran myran(p + ti);
Uint n=UINT32_MAX; // a random integer between 1 and n (inclusive)
int ix;
for(int i=0; i < m; i++) {
ix = ti*m+i;
ru[ix] = 1 + myran.int64() % (n-1);
}
}

int main(int argc, char** argv)
{
  Uint seed;
srand(rseed());
// srand(getpid());
int Nrep = 3000;
int k = 50000; // threads
int mK = 20; // in kernel
int* pnreg = &k;

char f0[] = "test1.bin";
FILE *fp;
if( (fp = fopen(f0, "wb")) == NULL ) {
printf("ERROR\n"); exit(1); }

if( cutCheckCmdLineFlag(argc, (const char**)argv, "device") )
cutilDeviceInit(argc, argv);
else
cudaSetDevice( cutGetMaxGflopsDeviceId() );

int size_C = (*pnreg)*mK;
int mem_size_C = sizeof(Uint) * size_C;
Uint* d_C;
cutilSafeCall(cudaMalloc((void**) &d_C, mem_size_C));
  // allocate host memory for the result
  Uint* h_C = (Uint*) malloc(mem_size_C);
//-------------------------------------------------------------------
// setup execution parameters
int nthreads, nblocks;
div_t d = div((*pnreg), BLOCK);
if(*pnreg <= BLOCK) {
nblocks = 1;
nthreads = (*pnreg);
} else {
  // not necessarily a multiple of block size
nblocks = int(ceil(float(*pnreg)/BLOCK));
nthreads = BLOCK;
}
dim3 dGrid = nblocks;
dim3 dBlock = nthreads;
printf("n. of required threads = %d\n",(*pnreg));
printf("dGrid = %d \t dBlock = %d \n", nblocks, nthreads);
//-------------------------------------------------------------------
  for(int rep=0; rep < Nrep; rep++) {
seed = rand();
// execute the kernel
runifK<<< dGrid, dBlock >>>(d_C, seed, mK);
// check if kernel execution generated and error
cutilCheckMsg("Kernel execution failed");
// copy result from device to host
cutilSafeCall(cudaMemcpy(h_C, d_C, mem_size_C, cudaMemcpyDeviceToHost) );
fwrite(h_C,1, mem_size_C, fp); // binary write
}
//-------------------------------------------------------------------
fclose(fp);
// clean up memory
  free(h_C);
cutilSafeCall(cudaFree(d_C));
cudaThreadExit();
return 0;
}
```