Very large scale tests (VLST) for random numbers

We have developed a new type of large tests to complement the available standard tests for analyzing the quality of random numbers in a mathematical sense and to verify their usefulness in Monte Carlo calculations.

Considering that today we are able to run computationally intensive tasks on a modern HPC installation, possibly involving 16’000 cores for a day or even longer, it is quite obvious that we need to adapt our testing strategy to the dimension of these applications. For this reason, we have used a small HPC system (about 30 TFLOPS) exclusively for testing the AHS-RNG, both deterministic and true random, and for testing commonly accepted PRNGs, such as the MT19937 and the XOshiro256**. The HPC system, named TRALLES in memory of Johann Georg Tralles (1763-1822), a German mathematician and geodesist, consists of 16 Supermicro servers with a total of 1024 cores (AMD 7702P), a total of 8000 GB of RAM, and 750 TB of Raid 5 disk capacity.

1. Test U01 "BigCrush”

The most comprehensive standard test for random numbers is currently the U01 test. In the large "Bigcrush" variant, 106 different statistical tests with 160 different calculations yield 254 p-values as a result. The total number of bits used for the different tests is about 11'427 billion bits. The computing time is about 2:30 to 4 hours.

The first VLST is the test U01, not to neglect the classical mathematical statistics. We take the Bigcrush version of test U01, but we run this test not only once, but 10’000 tests with MT19937 and XOshiro256** and always 50’000 tests with AHS-RNG in deterministic mode, AHS-RNG true random mode with 50’000 times the same starting values and 50’000 times AHS-RNG true random with different starting parameters. We can provide the results of the p-values in an excel file or a bc file for further investigation. A detailed analysis of the results is planned for the near future.

A first analysis of the results of MT19937, XOshiro256** and our AHS-RNG in both deterministic and non-deterministic modes confirmed the correctness of the AHS-RNG. As for the MT19937, we were not surprised to find the known inadequacy of the MT19937 with respect to the "linear correlation test" (p values nos. 177 and 179). All other tests are as expected, i.e., well distributed over the space 0 to 1. Working with bigcrush, we found minor deficiencies in the mathematical basis and minor errors in the programming. Upon request, we are happy to support an announced revision.

2. Birthday Paradox

The second VLST is the birthday paradox. We do not use the 365-day version, but decided to use a "year" of 512 days. In this way, we consider all 9 bits from the output stream of the random number generators. By using the 9 bits continuously and without spaces, we get cyclic data from 9 different positions within the 64-bit random numbers.

For the 365-day version and 23 people in one room, there are 1255 different variants: from zero identical birthdays, once two identical birthdays, to 23 identical birthdays. For the 512-day version and 27 people in a room, there are 3010 different variants. We run 5000 jobs with 1000 billion room fills, testing the random numbers for generating the different days. This gives a total of 5 × 10^{15} test cases, each consisting of generating 27 "birthdays" using the random number generator under test and finding the appropriate variant number from the 3010 possible.
The first analysis is, of course, the number of different cases per variant that we encounter, both in the total number and in the 5000 subtotals per run. We know the exact expectation for each of the 3010 variants and can therefore compare the expectations with the detected cases and calculate the basic statistics.
The second analysis is to compare the number of "at least two identical" cases per 100 million cases, the results we obtained with the expected values. Since we have 10’000 values of subtotals per run, we can statistically analyze a total of 50 million values.
The third analysis consists in checking the days (0 to 511) that we encountered during the tests. We have available to us the subtotals from each run, i.e., 5000 records of 512 days. In total, we have 1.35 × 10^{17} random "day values" for our statistics.
The fourth analysis consists of checking the distribution of "ones" and "zeros" using the 1.35 × 10^{17} days. We get 9 results per day, so we can analyze a total of 1.215 × 10^{18} bits.
A fifth analysis concerns Poisson statistics for five rare events. For the cases 1x4, 2x3, and 2x2 identical (expectation of 38.6683... per 1000 billion) / 1x5 and 1x4 (43.9652...) / 1x6 and 3x2 (36.3850...) / 1x7 (32.5476...) /1x7 and 1x2 (12.5615...) we store the type and cycle of occurrence so that we can check the result of the RNGs against some Poisson statistics.
With this second VLST, we obtain results on the one-bit distribution, the 9-bit distribution, and the combinatorial correctness over the range of 243 bit strings.

<u>3. 1'000'032'000 keys with 256 bits, each compared with all the others</u>

With the third VLST we test the correctness of the RNGs when generating 256-bit keys, as they are often used in cryptography. If we generate more than 2^{256} keys, we must of course encounter identical keys. However, since this is not possible at the moment and probably never will be, we can rely on the binomial results of the differences between two independent 256-bit strings. As the number of comparisons increases, the spread of the expected number of differences also increases.
For this test, we compare 1’000’032’000 keys, each with all the other 1’000’031’999. Since we do not need to compare B with A if we compared A with B, we get 500’032’000’011’984’000 results. For this number of cases, the expected differences in the binomial distribution table start at the minimum of 60 bit differences (0.87 expected cases) and go up to 196 bit differences for the maximum of expected differences (0.87 expected cases), since the values are symmetrically distributed on either side of the highest expectation at 128 bit differences. Considering the very large number of comparisons, we do not store the individual results, but only calculate the total value per number of different bits. Since we perform the calculation in parallel on 990 cores, we also have the values of 990 partial sums.

Using these results, we can analyze the relationship between the expected number of cases per number of differences and the number of cases counted, calculate the standard deviation, and other useful statistics.

4. A Repetition Test for PRNGs and RNGs

The fourth VLST is based on a test proposed by Gil, Gonnet and Petersen (A Repetition Test for Pseudo-Random Number Generators, Monte Carlo Methods and Appl. Math. Vol. 12, No 5-6, pp. 385-393 / 2006). In this test, we generate 32-bit random numbers until we get an identical (the first repetition) to one of the previously generated (and stored) numbers from this cycle. But as an extension of the described test (run 100 cycles only three times and calculate the average), we run 100 billion cycles and store the number of results in run length per cycle. In practice, the run length of a cycle can vary from 2 to about 460’000. Per cycle, we store the run length of the cycle, the position of the first occurrence of the identical to the repetition, and the value of the random number that generates the first repetition. The number of the cycle is the position in the table. Since we have these details from all 100 billion cycles, we can produce many different statistics. The most important, of course, is to compare the number of cases at a given run length with the expected value and calculate the standard deviation.
It was pure coincidence that we discovered a small problem with artifacts in the MT19937 PRNG in this statistic, no need to worry because this artifact occurs on average only every 5 billion cycles.
Since we have to produce 100 billion times about 82'137 random numbers of 32 bits, i.e., about 8.2137×10^{15} 32-bit numbers, we also perform a classification of these numbers. This allows us to compute statistics on the normal distribution of these numbers over the 4'294'967'296 (2^{32}) distinct values.
Appendix A explains that there is a possible case where the last number (the first repetition) of a case is identical to the first random number of the next case. To check for these very rare exceptions, we also store the occurrence of these exceptional cases (expected value of 23.283... cases per 100 billion).

5. Pairs or triplets of the new random number with the last 10000 generated

The fifth test has its origin in the search for artifacts in PRNGs. Experience with this test has shown that it is useful for testing randomness in general. The basic idea is simple: we start by generating unsigned 32-bit random numbers and store them in a table with 10000 elements.
When the table is filled with 10’000 values, we continue with the normal mode. In the main program we now generate a new random number and test if this value is present in the last 10’000 generated values. Be careful, the new generated value may appear several times in the table, exceptionally, but possible. After this test, we need to replace the oldest random number with the newest one to always have the last 10’000 generated random numbers in the table. Each time we found in the table a random number with the same value as the new one, we write this information in a binary file: Number of the cycle of the RNG, the value itself and the distance of the second value, the new number, from the first value in

the table. The number of the cycle indicates the n-th generated 32-bit random number. Considering the dimension of the table, the distance can vary between 1 and 10’000; 1 is the case in which the same value is generated twice without another one in between, and 10’000 is the case in which the new random number forms a pair with the last one, the oldest number that is replaced in this cycle.
In practice, we run on 1000 cores in parallel for 20’000 billion cycles each. In total, we collect the information about 2 * 10^{16} generated random numbers. The probability of finding a pair or a double by comparing the new value with the last 10’000 generated before is only one in 429’496.7296. This is easy to understand since the probability of forming a pair between two 32-bit random numbers is one in 2^{32} . Since this possibility exists 10’000 times in our case, the expected value is 10’000 times larger. Per job, the expectation is 46’566’128.73.... pairs of 20’000 billion random numbers. So for the total of 1000 partial calculations, the expectation is 46’566’128’730 pairs. Given this large number, we decided to calculate the normal distribution of occurrence for each bit pattern, a total of 4'294'967'296.
The first interest is of course the matrix of two pairs in a row. We have defined a matrix of 10’000 first pairs * 10’000 second pairs. This allows us to detect all anomalies related to two pairs in a row. This test confirmed our detection of the very rare artifacts in MT19937 (on average one artifact per 5 billion random numbers generated) and we obtained evidence that no artifacts exist in MT19937 other than these artifacts. Besides this analysis, there are other statistics to study, such as the geometric distribution of cases with identical numbers in the last 10’000, or the number of triples.

6. "Measuring the random numbers"

Last but not least, we come to our sixth VLST, the run-length test of identical random numbers, the masterpiece of the new RNG tests. This is computationally the most challenging test, as we perform "measuring the random numbers", random numbers expressed as 32-bit unsigned integers. First, we calculate the exact length of the distances between two identical 32-bit random numbers, and also some additional metrics. The length of the distance between identical random numbers is geometrically distributed, the distance can be up to, and even over 140 billion cycles, in the case of very large extensive test sets, or by rare extreme values in smaller test sets. We run 5000 billion cycles per core. Since we need about 120 GB of ram per job (the way we did the program), we can only run 4 jobs per server. Running the test four times gives 1’280’000 billion cycles (4*16*4*5000 billion). We separately record the length to the first occurrence of each of the 4’294’967’296 distinct values, then the lengths between subsequent occurrences (including one occurrence), and the number of instances for each distinct random number pattern. At the end of the 5,000 billion cycles, we consider the first part of each case to be the end of the last unfinished string and include it in the global string length files. In this way, we get a total of 5'000 billion cases per job, for a total of 1'280'000 billion cycles.
During this "ring closure" we came across an interesting "paradox". From the beginning to the first appearance of a certain pattern we have for the length an average of 2^{32} cycles.

However, if we go back from the end (cycle number 5’000 billion), we also have an average of 2^{32} cycles to the last occurrence of the particular pattern. Since still no repetition is contained, the exact expectation must be 2^{32} - 1. In the first moment this fact looks strange, because the global average length must be only 2^{32} . If we think about these different values, we must conclude that this fact is not related to the "ring closure", but must also occur in every cycle. To prove this fact and to get additional metrics, we inserted a "checkpoint" at each multiple of 1 billion; per job we get 5'000 checkpoints.
At each checkpoint, we count the number of strings that arrive at the checkpoint and, with one exception, continue. This number must always be 4'294'967'296 strings.
At each control point, we calculate the total length forward and backward (including the cycle in which the pattern appears) for all strings arriving at the control point.
A separate metric is the number and average length of all strings that start after the -1 checkpoint and end before the actual checkpoint, i.e., do not touch any checkpoint. We have calculated the expected number of these cases to be 107’882’641.039220241.... for the interval of 1 billion. The average expected length of these cases is 34’587’472.070454.... per case.
Separately, we get on one side the number of cases per distance, and on the other side the number how often each random number occurred. With the first information, we can now calculate, for example, how many cases we expect between 20’000’000 and 20’100’000, and give the result for the RNG on test.
For the control, we calculate the numbers per distance multiplied by the distance, and the sum must be 1’280’000 billion × 2 .^{32}

Further tests we are considering for the future

This is a first overview of our toolbox for testing RNGs. There are a few others that we plan to implement in the future. For 64-bit integers, we could do a large sort of 1000 billion values and see if the distances between values or identical values match expectations. Or we could calculate the run length of zeros and ones on a large scale like 10exp15 64-bit integers in a stream. We also want to perform NIST's tests for cryptographic security.

Weidingen, on August 16th 2023

Alain Schumacher / SICAP R&D / vers. 1.1