Computer Science and Engineering 2015, 5(2): 25-29
DOI: 10.5923/j.computer.20150502.01

Testing Randomness: The Original Poker Approach Acceleration Using Parallel MATLAB with OpenMP

Wael M. F. Abdel-Rehim[1,*], Ismail A. Ismail[2], Ehab Morsy[3]

[1]Department of Mathematics and Computer Science, Faculty of Science, Suez University, Suez, Egypt
[2]College of Computers and Informatics, Misr International University, Cairo, Egypt
[3]Department of mathematics, Suez Canal University, Ismailia, Egypt

Abstract In this paper, motivated by certain practical applications such as Monte Carlo simulation and cryptography. On the other hand, Pseudo-random numbers are often required for simulations performed on parallel computers. We implement the classical Poker test using parallel MATLAB with OpenMP (Open Multi-Processing) using MEX-file. After that, we compares the performance of the implementation with that implemented the Poker test in parallel with MATLAB using MEX-file with multithreading. We show that the running time of implementing the Poker method using parallel MATLAB with OpenMP is significantly less than that using MEX-file (MEX stands for MATLAB Executable) with multithreading, especially when the number of random numbers is sufficiently large.

Keywords Poker test, Randomness, Cryptography, Parallel random numbers test, MATLAB Multithreading, OpenMP

1. Introduction

Measuring the quality of randomness of a given sequence is a crucial problem that significantly affects the quality of many practical applications such as distributed algorithms, cryptography [1, 2], statistical sampling and computer simulation, as example pseudo-random numbers used in Monte Carlo calculations [3]. In other words, the quality of such applications depends on generating unpredictable (random) sequence of quantities. From the practical point of view, such sequence must be of sufficiently large size in the sense that the probability of any particular value being selected must be sufficiently small in order to prevent an adversary from optimizing a search scheme based on such probability.

There are many techniques described in the literature for generating random, pseudorandom bits, parallel pseudorandom bits and numbers. A random bit generator is a device or an algorithm which outputs a sequence of independent and unbiased binary digits. A random bit generator can be used to generate uniformly distributed random numbers. However, generating of random bits is an inefficient procedure in most practical environments (storing and transmitting a large number of random bits are impractical if these are required in applications). We can overcome this difficulty by substituting a random bit generator with a Pseudorandom Bit Generator (PRBG).

* Corresponding author:
wael_fawaz@hotmail.com (Wael M. F. Abdel-Rehim)
Published online at http://journal.sapub.org/computer

In order to make sure that such generators are secure enough, they should be subjected to a variety of statistical tests designed to detect the specific characteristics expected of random sequences. We now review a number of empirical tests described in the literatures; for further details [4-7].

Runs Test tests the runs up and down or the runs above and below the mean by comparing the actual values to expected values. The statistic for comparison is the chi-square.

Frequency Test develops frequency distribution of individual samples, uses the chi-square test to compare the distribution of the set of numbers generated to a uniform distribution.

Poker Test (to be explained later in details) treats numbers grouped together as a poker's hand. Then the hands obtained are compared to what is expected using the chi-square test [8, 9].

These sequential tests often check for correlations within a stream (on one processor), or the combined stream from all processors [10], while parallel tests check for correlations between different streams (on different processes) [11].

MATLAB is a wonderful high-level programming language for scientific research. It is an interactive environment that provides high-performance computational routines [12]. Since threads are a common software solution for parallel programming, on multi-core systems [13]. We found that MATLAB supports kinds of parallelism one of them is multithreaded parallelism [14].

MATLAB is an interpreted language, M-files execute slower than compiled programs written in other languages, such as C, C++, and Fortran. Therefore, we use MEX-file (MEX stands for MATLAB Executable), because when we integrate MATLAB with C++ code we can combine the

advantages of MATLAB with the advantage of multithreading that increase speed. Moreover, the execution time of MEX files is significantly less than M-files [10, 15].

OpenMP (Open Multi-Processing) is an Application Program Interface (API) that supports multi-platform shared memory multiprocessing programming like C++. Moreover, OpenMP is a nice way to get a parallel program from a sequential program and it is standard for shared memory programming for scientific applications.

This paper is organized as follows. In Section 2, we discuss Poker test. In particular, two versions of Poker test are presented in the literatures, the classical Poker test and the modified Poker test. In Section 3, we run some experiments for Poker test in parallel with MATLAP followed by a discussion of the results. Finally, Section 4 we provide some final conclusions.

2. Poker Test

In this section we present in details the two versions of Poker test, the classical Poker test and the approximated Poker test.

2.1. Classical Poker Test

The classical poker test consists of using all possible categories obtained from poker that uses five hands, i.e. AAAAA (five of a kind), AAAAB (four of a kind), AAABB (full house), AAABC (three of a kind), AABBC (two pairs), AABCD (one pair), and ABCDE (bust). In general, the poker test using five hands considers n groups of five successive integers denoted by $(X_{5i}, X_{5i+1},\ldots, X_{5i+4})$, $0 \le i \le n$, and then observes which of the seven possible patterns is matched by each quintuple. The following table summarizes such patterns and their corresponding probabilities.

Name	Pattern	Probability
All different	ABCDE	0.3024
One Pair	AABCD	0.5040
Two pairs	AABBC	0.1080
Three of a kind	AAABC	0.0720
Full house	AAABB	0.0090
Four of a kind	AAAAB	0.0045
Five of a kind	AAAAA	0.0001

It is well known that the number of hands a poker test can apply with is not restricted to five [5]. In particular, Poker test that uses four hands is more convenient to be applied to certain applications such as simulation(see [16]), and cryptography (see [2], [17]) in which we need to generate random integers or a random sequence of bits. For example, in cryptography, secret keys (used for encryption of messages or other purposes) are generated using random number generators (RNGs) (see [17]). Thus we applied Poker test to bit streams (typically represented by a 32-bit or 64-bit unsigned integer) rather than floating point numbers, and since 64 bits is not evenly divisible by five we use the closest number that divides 64: four. That is, the generated sequence of random numbers is divided into segments of four bits (see [18]).

Given a sequence of n random numbers to be tested, it is shown that there is a limit based on n as to how large the value of k can be [19]. On the other hand, most practical applications apply poker test with different values of k in order to ensure that the underlying sequence is truly random [20].

Some cryptographic algorithms using block cipher take blocks, or keys with different sizes 128, 192, or 256 bits. Therefore, if the block or the key size is 192, we find that they are not evenly divisible by five or four, however divisible by two. Therefore, series of pseudorandom numbers generated is divided into parts, each consisting of two bits. This encourages us to apply Poker test with hands of two numbers instead of hands of three, four or five numbers, especially in applications involving testing the randomness of a sequences of bit such as cryptography.

Motivated by increase speed of the test we also implemented the Poker test in parallel with MATLAB using MEX-file with one, two, three and four threads, reducing the execution time of the test (see [21]).

A Chi-square test is based on the number of quintuple in each category. We count the number of occurrences in each k-tuples, and then use a chi-square analysis against the theoretical probabilities to determine whether the stack represents a fair poker deck. We computed the theoretical probabilities of some categories two (k=2), three (k=3), five (k=4) and seven (k=5) categories (see [18, 22, 23, 24]).

2.2. Approximated Poker Test

At the time the classical Poker test is designed, checking the occurrences of these subsequences of length five using a computer program creates difficulties for the programmers as they have no one systematic similarity. This motivates constructing a simpler version of the classical test to overcome the programming difficulties involved.

A good compromise would simply be to count the number of distinct values in the set of five (see [6, 16]). Namely, corresponding to the classical Poker test that uses five hands we get five categories, 1different, 2different, 3 different, 4 different and 5 different. Thus, a finite time algorithms have been designed to implement such modified Poker test; see [4, 24].

This breakdown is easier to determine systematically, and the test is nearly as good. In general, we consider n groups of k successive numbers, and then count the number of k-tuples with r different values. A chi-square test is then made using the following probability of the existence of r different.

$$\Pr = \frac{d(d-1)\ldots(d-r+1)}{d^{k}} \left\{ {k \atop r} \right\}$$

Where $\left\{ {k \atop r} \right\}$ denote the Stirling number of the second kind (see [25]) (the number of ways to partition a set of k

elements into exactly r parts). The Stirling number can be computed using a well known formula.

Then different hands obtained can be compared to what is expected using the chi-square test to see how far the data has strayed from the theoretical distribution.

3. Experimental Results

In this section we implement the classical Poker test in parallel with MATLAB using MEX-file with multithreading and the corresponding modified version using parallel MATLAB with OpenMP. We evaluate both versions of the test by implementing programs using C++ code that create random numbers. After that, we use MATLAB using MEX-file to count the occurrence of these differences or count number of occurrences, then classified each to possible type of poker hand. Finally, it determines the chi-square. We have compared the classical Poker test in parallel with one, and two threads and compare the performance.

The experimental results are reported on Laptop Core i5 2.50 GHz CPU, 4GB of RAM, 3M of Cache and, MATLAB 8.1 (R2013a).

We analyze the running time of the classical Poker test described in Figure 2, using parallel MATLAB with OpenMP. We determine the running time of executing the algorithm (time is in milliseconds). The resulting running time is shown in the following table 1.

Table 1. Summary of execution time in ms, for the original Poker test using threads and Using OpenMp

Random No	1 Thread	2 Thread	Using OpenMp
1000	27	15	14
5000	29	16	15
10000	46	32	31
50000	62	47	43
100000	78	67	62
500000	94	79	71
1000000	156	140	125
5000000	188	172	140
10000000	265	234	171
50000000	421	356	218
100000000	642	507	296

The results of Table 1 (shown in Figure 1) imply there is a significant improvement in term of the running time in the case of applying the classical Poker test with one, two threads and using parallel MATLAB with OpenMP, especially when the number of random numbers is sufficiently large.

Finally, we compare the performance of the classical Poker test described in Figure 2, with one, and two threads with that using parallel MATLAB with OpenMP. We are interested in the speedup of execution time for implement of the classical Poker method in parallel. Speedup Formula:

$$\text{Speedup } (S_n) = T_1/T_n$$

Where S_n is the speedup experienced on n processors, T_1 is the execution time for the sequential implementation, and T_n is the execution time on n processors. We calculated the speedup using the last equation. The tests were performed on one, and two threads as shown in figure 1. The resulting speedup is shown in table 2.

Table 2. Speedup results for two threads and Using OpenMp for each random numbers using Poker test

Random No	Speedup for 2 thread	Speedup for Using OpenMp
1000	1.80	1.93
5000	1.81	1.93
10000	1.44	1.48
50000	1.32	1.44
100000	1.16	1.26
500000	1.19	1.32
1000000	1.11	1.25
5000000	1.09	1.34
10000000	1.13	1.55
50000000	1.18	1.93
100000000	1.27	2.17

From Table 2 and figure 1, we show that the speedup of the implement of the classical Poker test in parallel with MATLAB using two threads is greater than one thread. Furthermore, Poker approach that using parallel MATLAB with OpenMP is significantly greater than that using one and two threads.

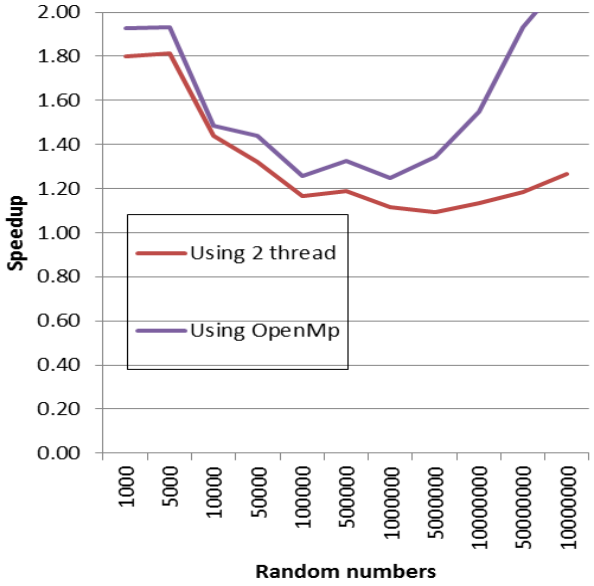

Figure 1. Speedup gained by implementing the classical Poker test on two threads and Using OpenMp using parallel MATLAB

```
1. Read number of hands to deal
2. Open File to read random numbers
3. do
4. Loads the hands into an array // a sequence of 5 numbers
5. Determine which kind of combination this group of 5
   contains
6. Count the number of similar values; breaks at 5
7. Increment the appropriate counter //(7 counters: all
   different, one pair, two pairs, three of a kind, full house,
   four of a kind and five of a kind)
8. While (loads the hands < number of hands)
9. Calculate the percentage of the n total repetitions
   corresponding to each counter
10. Computes the expected theoretical values
11. Compute chi square using the expected probabilities
12. Measure execution time in the program
13. Print "The program execution rime"
14. Print "Chi square"
```

Figure 2. The classical Poker test pseudo-code algorithm

4. Conclusions

We have been studied Poker method for testing randomness using parallel MATLAB with OpenMP. In particular, we have been discussed the Poker using parallel MATLAB with OpenMP using MEX-file and compared the performance with parallel with MATLAB using MEX-file with one and two threads. From the computations point of view, we have been compared the performance of implemented the Poker test in parallel with MATLAB using MEX-file with multithreading.

We show that the speedups of implementing the Poker test using parallel MATLAB with OpenMP is significantly greater than that using MEX-file using multithreading with one and two threads, especially when the number of random numbers is sufficiently large.

REFERENCES

[1] Fan, Zheng, Xiao-jian, Tian, Jing-yi, Song, Xue-yan, Li, 2008. Pseudo-random sequence generator based on the generalized Henon map. The Journal of China Universities of Posts and Telecommunications, 15(3), pp. 64–68.

[2] Menezes, A. J., Oorschot, Paul C van, Vanstone, Scott A, 1997. Handbook of applied cryptography. CRC Press. ISBN: 0849385237.

[3] Brent, R. P., 1998. Random number generation and simulation on vector and parallel computers. Proc. Fourth International Euro-Par Conference (Southampton, UK, 1-4 Sept 1998), D. Pritchard and J. Reeve (editors), Lecture Notes in Computer Science, Vol. 1470, Springer-Verlag, Berlin, 1998, 1-20.

[4] Hamilton, John A., Nash, David A., 1997. Distributed Simulation. CRC Press, ISBN: 0849325900.

[5] Kendall, M G, Smith, B B, 1938. Randomness and random sampling numbers. Journal of the Royal Statistical Society 101, 147–166.

[6] Knuth, D. E., 1997. The Art of Computer Programming: Seminumerical Algorithms. Volume 2 (3rd Ed.), Addison-Wesley Longman Publishing Co., Inc, ISBN: 0201896842.

[7] Sheskin, David J., 1997. Handbook of: Parametric and Nonparametric Statistical Procedures. CRC Press, ISBN: 0849331196

[8] Rutti, Mario, 2004. A Random Number Generator Test Suite for the C++ Standard. Diploma Thesis, Institute for Theoretical Physics, ETH Zurich.

[9] Stewart, William J., 2009. Probability, Markov chains, queues, and simulation: the mathematical basis. Princeton University Press, ISBN: 0691140626.

[10] Coddington, Paul D. and Ko, Sung-Hoon, 1998. Techniques for Empirical Testing of Parallel Random Number Generators. Northeast Parallel Architecture Center. Proc. International Conference on Supercomputing (ICS'98).

[11] Srinivasan, A., Ceperley, D., and Mascagni, M., January 2003. Testing Parallel Random Number Generators. Parallel Computing, Volume 29, Issue 1, Pages 69–94, ISSN: 0167-8191.

[12] Mathworks Inc. (2013) MATLAB user's guide. [Online]. http://www.mathworks.com/products/matlab/.

[13] Luszczek, Piotr. Enhancing Multi-Core System Performance Using Parallel Computing with MATLAB, MathWorks. http://www.mathworks.com/company/newsletters/articles/enhancing-multi-core-system-performance-using-parallel-computing-with-matlab.html.

[14] Moler, Cleve. Parallel MATLAB: Multiple processors and multiple cores. The MathWorks Newsletters, 2013. http://www.mathworks.com/company/newsletters/articles/parallel-matlab-multiple-processors-and-multiple-cores.html.

[15] Bachnak, Rafic and Lee, Roger, Winter 2003. Converting M-Files to Stand-Alone Applications Technology Interface (Electronic Journal of Engineering Technology), Vol.5, No.1, ISSN 1523-9926.

[16] Karian, Zaven A., Dudewicz, Edward J., 1998. Modern statistical systems and GPSS simulation. Second Edition, CRC Press, ISBN: 0849339227.

[17] Brands, Stefan, Gill, Richard, 1995. Cryptography, statistics and pseudo randomness I. Probability and mathematical statistics, Vol. 15, pp. 101–114.

[18] Abdel-Rehim, Wael M. F., Ismail, Ismail A., Morsy, Ehab, 2012. Testing randomness: Implementing poker approaches with hands of four numbers. International Journal of Computer Science Issues, Vol. 9, Issue 4.

[19] Supaan, Suriyati Binti, 2008. Analysis for A5/1 and A5/2 algorithm (stream ciphers). Faculty of Electrical Engineering, Senior Thesis, Faculty of Electrical Engineering, Universiti Teknologi Malaysia.

[20] Talamba, Sorin, 2001. A Theoretical and Empirical Study of Uniform Pseudo-Random Number Generators. Senior Thesis, Department of Computer Science, Middlebury College.

[21] Abdel-Rehim, Wael M. F., Ismail, Ismail A., Morsy, Ehab, 2015." Testing Randomness: The Original Poker Approach Acceleration Using Parallel MATLAB". Journal of Computer

Science and Applications (CSA).Vol. 2, No. 2. ISSN: 2333-9071.

[22] Abdel-Rehim, Wael M. F., Ismail, Ismail A., Morsy, Ehab, 2012. Implementing the classical poker approach for Testing Randomness, (Submitted).

[23] Abdel-Rehim, Wael M. F., Ismail, Ismail A., Morsy, Ehab, 2012. Testing Randomness: Poker Test with Hands of Three Numbers. Journal of Computer Science 8 (8): 1353-1357.

[24] Abdel-Rehim, Wael M. F., 2012. Testing randomness: Poker test with hands of two numbers. Proc. 8th International Computing Conference in Arabic: ICCA 2012 - December 26-28, Cairo, Egypt.

[25] Karl, Andrew, 2008. Pseudorandom Numbers: Generation, Statistical Measures, Monte Carlo Methods, and Implementation in C++. Senior Thesis, Department of Mathematics, University of Notre Dame.

[26] Weisstein, Eric W., "Stirling Number of the Second Kind", From Math World-A Wolfram Web Resource-http://mathworld.wolfram.com/StirlingNumberoftheSecond Kind.html.