Theoretically optimal computing frontiers for rapid multiplication through decomposition

Nikolay Raychev

Abstract - The productivity of the computation systems is determined to a high degree by the speed of multiplication, which is a basic element in multiple applications for digital signal processing, quantum computations, etc. The various approaches for design of asymptotically rapid algorithms for multiplication have developed in parallel with the evolution of the effective multiplicator architectures. This article examines different approaches for optimal rapid multiplication..

Index Terms— boolen function, circuit, composition, encoding, gate, phase, quantum.

1. INTRODUCTION

The algorithm for multiplication is a major factor in the computation systems both for digital signal processing, filters, etc. So that the implementation of these applications can be improved by optimizing various parameters of the multiplication algorithms, as power, speed, area and fault tolerant property. According to the Moore's Law in 2020 the main components of the computer memory will reach the size of the atom. In such scales the theory on which are built the present computers will stop functioning and through the quantum computers will have to be reinvented the theory of the computer sciences.

The algorithm for multiplication involves generating and adding a partial product. So that the implementation of the multiplication algorithm depends on the number of the partial products and the speed of the adder. This article deals with the study and comparison of the different algorithms for multiplication mainly from the standpoint of speed, quantum costs and **fault tolerant property**. The purpose of the study is to find and offer the best theoretical results in terms of speed, power and quantum price.

The detection of asymptotically fast algorithm for multiplication is one of those easily available, but difficult to solve problems that are time-consuming. Although the problem is difficult, perhaps it is best presented by asymptotically the fastest known algorithm with time for performance $O(n\log(n)\,2^{3log^{*}n})$ instead of the considered to be optimal $O(n\log(n))$, it hides different small useful guidelines for optimization. For example, here is one insignificant guideline: squaring is not less resource consuming than multiplying. If a person can square quickly, then he can multiply quickly, by rewriting $a \cdot b$ in the following form: $\frac{1}{4}(a+b)^2-(a-b)^2$. This asymptotic equation is convenient, because some algorithms for multiplication are easier to understand compared to the squaring algorithms, because:

$$(a \ll n+b)^2 = (a^2 \ll 2n) + (((a+b)^2 - a^2 - b^2) \ll n) + b^2)$$

Although there was not much of a progress achieved concerning the issue with rapid multiplication, can be examined some interesting guidelines for optimization.

2. CONVOLUTION AND PRINCIPAL ROOTS

The fastest known algorithms for multiplication are based on the convolution theory. They use a change of the basis, to turn cyclical convolutions into pointwise products. In this way the difficult part of multiplying, adding up all small sub-factors, can be reduced up to a cyclic convolution.

The essential thing that is necessary in order to make the convolution theorem to work is a principal root of unitarity. With such a root can be created a set of basis vectors $v_0, v_1 \ldots v_{n-1}$, where there are no cross-links when performing a cyclic convolution. In other words, when calculating the cyclic convolution of two basis vectors $v_a * v_b$ is obtained 0. This is useful, because it means that the sums from the convolution $(v_a + v_b) * (v_a + v_b)$ will be simplified into $({v_a}^{*2} + {v_b}^{*2})$ instead of ${v_a}^{*2} + {v_b}^{*2} + 2v_a * v_b$ The loss of the part $2v_a * v_b$ reduces the fractionation when recurring, which is important when it comes to asymptotic complexity.

What exactly is a principal root of unitarity? Formally, the n-th principal root of unitarity λ must satisfy two properties:

1. A unitarity must be obtained when raising on the n-th power. Otherwise it would not be a root of unitarity
   $\lambda^n = 1$

2. If all powers of λ or every second power, or every third power, or every k power are summed up, must be obtained 0. This is the part without cross-links
   $\sum_{i=0}^{n-1} \lambda^{ki} = 0$ for every $k \in [1, n)$

For example, the number two is a fourth principal root of unitarity, when working modulo five:

- Exponentiating to one: $2^4 = 16 = 1 (mod\ 5)$
- The sum of the powers is zero: $2^0 + 2^1 + 2^2 + 2^3 = 1 + 2 + 4 + 8 = 15 = 0\ (mod\ 5)$
- The sum of every second power is zero: $2^0 + 2^2 + 2^4 + 2^8 = 1 + 4 + 16 + 64 = 85 = 0\ (mod\ 5)$
- The sum of every third power is zero: $2^0 + 2^3 + 2^6 + 2^9 = 2^0 + 2^3 + 2^2 + 2^1 = 15 = 0\ (mod\ 5)$

A more commonly used example of a principal root of unitarity is the complex root of unitarity. A specific example is that the complex number $\cos(\tau/3)+i\sin(\tau/3)$ is the third principal root of unitarity. More generally $e^{i\tau/n}$ is always an n-th principal root of unitarity and forms the basis of the Fourier transform.

After having a principal root of unitarity we can now construct a basis without cross-links.

A useful basis

Let there be given an n-th principal root of unitarity λ. It is known that λ satisfies some useful properties related to the sums of the powers, the sums of the even powers, etc., so let's use that. The entries of the vectors are taken directly from the lists of things that must give zero. More specifically, the j-th entry of the κ-th vector will be λ^{ki}. That gives us a set of basis vectors:

$v_0 = [1,1,1,\dots,1]$
$v_1 = [1,\lambda,\lambda^2,\dots,\lambda^{n-1}]$
$v_2 = [1,\lambda^2,\lambda^4,\dots,\lambda^{(n-1)2}]$
...
$v_k = [1,\lambda^k,\lambda^{2k},\dots,\lambda^{(n-1)k}]$
...
$v_{n-1} = [1,\lambda^{n-1},\lambda^{n-2},\dots,\lambda]$

Let's now consider what is happening, when computing the cyclic convolution of two basis vectors $v_a * v_b$ c a ≠ b. The i-th entry of the resulting vector will be equal to:

$$(v_a * v_b)_i = \sum_{j=0}^{n-1} \lambda^{aj}\,\lambda^{(i-j)b} = \lambda^{bi} \sum_{j=0}^{n-1} \lambda^{(a-b)j}$$

But in the last term $a-b$ is constant and is multiplied with iteration variable. This is one of the sums, which should result to zero, if λ is a principal root of unitarity! Therefore $v_a * v_b = 0$, when $a \neq b$. When $a = b$, instead of this are summed n copies of $\lambda^0 = 1$ and are multiplied by the original value of the vector. So in the end is obtained a scaling: $v_a{}^{*2} = n v_a$

The above facts suggest significant optimizations when it comes to re-expressing a cyclic convolution with respect to a special basis v. A vector x may be decomposed into $x = a_0 v_a + a_1 v_1 + \dots + a_{n-1} v_{n-1}$, and then x^{*2} will be simplified in the following way:

$$\begin{aligned} x^{*2} &= (a_0 v_0 + a_1 v_1 + \dots + a_{n-1} v_{n-1})^{*2} \\ &= (a_0 v_0)^{*2} + (a_1 v_1)^{*2} + \dots + (a_{n-1} v_{n-1})^{*2} \\ &= n(a_0{}^2 v_0 + a_1{}^2 v_1 + \dots + a_{n-1}{}^2 v_{n-1}) \end{aligned}$$

In other words, if it effectively can be converted to and from the special basis (usually through adaptation of an algorithm for Fast Fourier Transform), then a given number can quickly be squared (or to be multiplied two numbers) by converting to the special basis, as each coefficient is squared individually, then is converted back.

In practice: Schönhage–Strassen Algorithm

The algorithm of Schönhage–Strassen (SSA) is an algorithm for multiplication, based on a convolution, and has been the fastest known for decades. SSA works in the scope of the integers modulo $2^{2^s} + 1$ Let's define $m = 2^s$ This is how it is worked modulo $2^m + 1$

It turns out that two is 2^m-th principal root of unitarity of that range. This is ultimately the reason SSA is so fast: The change of the basis can be done with shifts instead of multiplications. Instead of taking the time $O(n \log^2(n))$, SSA only uses time $O(n\ \log(n))$ for its change of the basis. Although the change on the basis of SSA takes time $O(n\ \log(n))$, the entire time for running of SSA still is $O(n \log(n)\ \log(\log(n)))$ What is happening? Must be something with the selected principal root of unitarity, completely overlapping too quickly.

In the field with $2^m + 1$ elements, it takes only m doublings, in order to reach 2^m, the value identically equal to -1. After m more doublings, effectively multiplying by -1, again is obtained 1. SSA uses a principal root of unitarity of order $2^{s+1} = 2m$ and that order is logarithmic with respect to the field size $2^m + 1$

The logarithmic order of the principal root of unitarity has a major consequence: limiting how many pieces can be used when splitting up a number to be squared. The purpose is to represent each piece as a linear combination of vectors from a basis with dimension equal to the order of the principal root of unitarity, but if there are more pieces, then there must be some direction which is not covered. An

n-dimensional vector space can not be stuck into an (n-d)-dimensional vector space, without losing information.

Let's divide the input number to a k pieces with a size of $\frac{n}{k}$. A field with at least k roots is needed in order to be processed all those pieces, so $2m$ must be at least k. In addition, the field is necessary to be large enough in order to hold $\frac{n}{k}$ bits for each piece, and that means m must be at least $\frac{n}{k}$ (plus a bit more). The last restriction requires the use of very small pieces, but the previous one limits strongly how small they can be.

Both opposing restrictions comply to $k \approx \sqrt{n}$, so at the end are used $\sqrt{n}$ pieces with a size of $\sqrt{n}$. It takes $\log(\log(n))$ repeated square roots, to go from n to a basis with constant size and from there comes the additional factor $\log(\log(n))$ in the complexity of implementation.

Contrary to the above it is interesting what can be done, if working modulo a prime p with a $p-1$-th principal root of unitarity. The restrictions for the size of the pieces and the number of roots will look like $p \geq k$ и $p \geq 2^{\frac{n}{k}}$ These new restriction lead to $k \approx \frac{n}{\log(n)}$ so that logarithmically large pieces can be used.

The logarithmically large pieces are great because they allow for shortening the recursion.

Performing many small squarings fast

If $^{n}/_{\log(n)}$ squarings with a size of $\log(n)$ have to be performed, they may be performed for a time $O(\log(n))$. The trick to do this is to sort the input data into ascending order (but the original order must be also kept) and then to record each possible square number, while iterating the input data.

```
def q_stream_square(inputs):
  """
  Squares n inputs of size w for time O(w n log(n) + w 2^w).
  """
  q_indexed = zip(inputs, range(len(inputs)))  # O(w n)
  q_ordered = sorted(indexed, key=lambda e: e[0])  # O(w n log(n))

  # O(n w + w 2^w)
  counter = 0
  q_counter_squared = 0
  q_ordered_squared = []
  for e in q_ordered:
    while counter < e[0]:
      # This internal cycle takes totally O(w 2^w).
      # Works at most 2^w times and cost w each time.
      q_counter_squared += counter
      counter += 1
    q_counter_squared += counter
    q_ordered_squared.append((q_counter_squared, e[1]))
  q_restored = sorted(q_ordered_squared, key=lambda e: e[1])  # O(w n log(n))
  return [e[0] for e in q_restored]  # O(w n)
```

This means that if a change of a basis can be performed with application of the convolution theorem for a time $O(\log(n))$, and at the end there are $O(^{n}/_{\log(n)})$ pieces of size $\log(n)$ then the multiplication can be done for a time $O(n\log(n))$.

The above does not apply to SSA, because working modulo $2^{2^s}+1$ does not give enough pieces.

Degenerate orders

Let's assume that is working with the scope of integers modulo 3^n. Therefore, it happens that 2 satisfies all the requirements to be the $2 \cdot 3^n$-th principal root of unitarity for that scope. Does this mean that the convolution theorem can be applied, in order to perform a multiplication cheaply? Not exactly.

In the special basis for turning of a convolution into point-wise multiplication, ${v_a}^{*2}$ is equal to nv_a, where n is the order of the root. In the case of integers modulo 3^n with principal root of unitarity two, this comes out as $2 \cdot 3^{n-1}v_k$... But the multiplication includes a large power of three, and the powers of three do not have a reciprocal value, when working modulo 3^n. In this scope can not be undone the multiplication, created by the convolution theorem! In an attempt to compute a convolution in this way, the entire information is just pushed out of the scope.

Therefore the Schönhage–Strassen algorithm requires the use of a field with a size of double power of two such as $2^{2^s}+1$, instead of 2^n+1. The principal root of unitarity (two) has a power 2^n, when working modulo 2^n+1, but 2^n may not have a reciprocal value in such a context. For example, when working modulo 2^9+1, the application of the convolution theorem will cause a multiplication by 9, but $2^9+1 = 513 = 57 \cdot 9$, such that this multiplication can not be undone. In this way a convolution would throw out information.

When n is 2^n also a power of two, then for sure there is a reciprocal value, because the principal root of unitarity passes through each power of two on the cycle back to 1. Therefore, SSA adhere to fields of size $2^{2^s}+1$. (Additional benefit from the requirement the number of pieces to be a power of two is that the simple Cooley-Tukey algorithm may be reworked for the change of the basis, instead of using something more complicated.)

Thus not only is necessary a quick change of the basis and a principal root of unitarity of a large order, but care must be

taken from where comes this size. If the order shares coefficients, with the size of the scope, it will not work.

Let's now examine a few more things that do not work.

Tried things

The title of the article says the problem with the rapid multiplication is not solved. Here are listed several things, which are tried and do not work.

1. Caching of the squares.
The caching of the multiplications up to numbers of size $\log(n)$ would take space $0(n^2)$, but caching only the squares takes $O(n)$. A lookup-table can be made and squarings of size $\log(n)$ can be performed for a constant time.

Unfortunately, despite the fact that this gives useful ideas, there are practical and theoretical reasons why this lookup-table will not work.

In practice, a given computer can already square numbers up to 128 bits for a constant time, and simply there would not be a space to store a table with more than 2^{128} elements.

In theory, the lookup-table creates a conflict with the rules of the abstract machine that is used. If working in a RAM model with logarithmically big words, then the squarings can already be cached for a constant time. If working with a bit complexity, then the requests are not a constant time, because is needed a $O(\log n)$ time only for reading the bits of the offset. If thinking about multi-tape Turing machines, then they do not have a random access.

In the end, there is no actual benefit, i.e. it does not work.

2. Use of the field of integers modulo a prime p.
The algorithm SSA is based on the use of an alternative arithmetic system, the integers modulo $2^{2^s}+1$. Perhaps can be applied also better systems?

There are primes, where two is p-1-th principal root of unitarity modulo p. That's as proportionally large as the order of a root of unitarity will become; it is cycled through any other value in the field before returning to 1! This, however, means that the powers of two can not satisfy any property for making the coefficients cheap for application.
Too expensive, i.e. does not work.

3. And why not module $(2^n+1)^2$ or $(2^n-1)^2$?
The hope for this idea is that the squaring of the size will square the order of the root, and will not ruin everything. Unfortunately, the hope is in vain.

Instead of squaring the order of the root, e.g. from $2m$ to $4m^2$ the squaring of the scope seems to multiply the order of the root by the same coefficient that the size of the scope was multiplied by. That is a problem, because it means that the size of the scope is not already relatively prime to the order of the root, in other words end of information upon convolution.

Gives wrong answers, i.e. does not work.

4. Module 3^n?
As already explained, the size of the field is not relatively prime to the order of the root.

Gives wrong answers, i.e. does not work.

5. And what if working modulo $lcm(2^n+1, 2\cdot 2^n+1)$?
This particular scope is tempting.

In the first place, that takes quadratically many doublings, in order to reach 1, which would allow alteration of SSA to use instead of $\sqrt{n}\sqrt[3]{n^2}$ pieces, and thus to avoid the above factor $\log(\log n)$.

On the second place, the operations are cheap. A number x may be represented as a pair of numbers $[x\ mod(2^m+1), x\ mod(2\cdot 2^m+1)]$ and later to recover its value thanks to the Chinese remainder theorem. Within that representation, adding or multiplying two values is done point-wise, i.e. $[2,3]\cdot[5,7]=[2\cdot 5, 3\cdot 7]$ Combined with the fact that this representation turns each power of two into a pair of powers of two, may directly be used tricks of SSA for cheap performance of rotations.

Then what is the problem? Two is a root of unitarity of that scope, but not a principle root of unitarity.

There is a cross-talk, i.e. it does not work.

6. Using the Hadamard transformation.
The Hadamard transformation is extremely similar to the rapid Fourier transform, but can be done in time $O(n\log n)$ without multiplication thanks to the rapid Walsh-Hadamard transform. So maybe something interesting will happen, if the convolution theorem is applied to the basis of Hadamard?

Actually really something interesting is happening! In the end, must be calculated
$y_j = \sum_{i=0}^{n-1} x_i\, x_{j\oplus i}$.

Only that the objective is to calculate $y_j = \sum_{i=0}^{n-1} x_i\, x_{j-i}$. It is close, but it is not the result sought.

There are cross-links, i.e. it does not work.

3. SUMMARY

The algorithms for rapid multiplication use a change of basis and the convolution theorem in order to turn many-to-many multiplication into one-to-one multiplication.

If a way is found to break a number of $n/\log(n)$ to pieces of size log$n$ and can be performed a change of basis for a time $O(n\log n)$, then it may be used a sorting and recording of all squares up to log$n$ for creation of an overall algorithm for multiplication $O(n\log n)$.

If a principal root of unitarity of order n is selected and n does not have a reciprocal value in the working context it is very likely not to obtain a solution.

REFFERENCES

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, 1st ed. (Cambridge University Press, Cambridge, UK, 2000).
[2] P. W. Shor, SIAM Journal on Computing 26, 1484 (1997).
[3] L. K. Grover, Physical Review Letters 79, 325 (1997).
[4] C. H. Bennett and G. Brassard, in *Proceedings of IEEE international Conference on Computers, Systems and Signal Processing, Bangalore, India* (IEEE Press, New York, 1984), p. 175.
[5] A. K. Ekert, Phys. Rev. Lett. 67, 661 (1991).
[6] C. Elliott, New Journal of Physics 4, 46 (2002). 12
[7] C. Elliott, D. Pearson, and G. Troxel, in *Proceedings of the ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 25-29, 2003, Karlsruhe, Germany.* (PUBLISHER, ADDRESS, 2003), pp. 227–238.
[8] C. Elliott, IEEE Security & Privacy 2, 57 (2004).
[9] C. Elliott *et al.*, in *Current status of the DARPA quantum network (Invited Paper)*, edited by E. J. Donkor, A. R. Pirich, and H. E. Brandt (SPIE, ADDRESS, 2005), No. 1, pp. 138–149.
[10] J. H. Shapiro, New Journal of Physics 4, 47 (2002).
[11] B. Yen and J. H. Shapiro, IEEE Journal of Selected Topics in Quantum Electronics 9, 1483 (2003).
[12] S. Lloyd *et al.*, SIGCOMM Comput. Commun. Rev. 34, 9 (2004).
[13] I.-M. Tsai and S.-Y. Kuo, IEEE Transactions on Nanotechnology 1, 154 (2002).
[14] S.-T. Cheng and C.-Y. Wang, IEEE Transactions on Circuits and Systems I: Regular Papers 53, 316 (2006).
[15] J. C. Garcia-Escartin and P. Chamorro-Posada, Phys. Rev. Lett. 97, 110502 (2006).
[16] M. Oskin, F. T. Chong, and I. L. Chuang, Computer 35, 79 (2002).
[17] D. Copsey *et al.*, IEEE Journal of Selected Topics in Quantum Electronics 9, 1552 (2003).
[18] C. H. Bennett and S. J. Wiesner, Physical Review Letters 69, 2881 (1992).
[19] X. S. Liu, G. L. Long, D. M. Tong, and F. Li, Phys. Rev. A 65, 022304 (2002).
[20] A. Grudka and A. W´ojcik, Phys. Rev. A 66, 014301 (2002).
[21] C.-B. Fu *et al.*, JOURNAL OF THE KOREAN PHYSICAL SOCIETY 48, 888891 (2006).
[22] A. Winter, IEEE Transactions on Information Theory 47, 3059 (2001).
[23] H. Concha, J.I.; Poor, IEEE Transactions on Information Theory 50, 725 (2004).
[24] M. Fujiwara, M. Takeoka, J. Mizuno, and M. Sasaki, Physical Review Letters 90, 167906 (2003).
[25] J. R. Buck, S. J. van Enk, and C. A. Fuchs, Phys. Rev. A 61, 032309 (2000).
[26] M. Huang, Y. Zhang, and G. Hou, Phys. Rev. A 62, 052106 (2000).
[27] B. J. Yen and J. H. Shapiro, in *Two Problems in Multiple Access Quantum Communication*, edited by S. M. Barnett *et al.* (AIP, ADDRESS, 2004), No. 1, pp. 25–28.
[28] B. J. Yen and J. H. Shapiro, Physical Review A (Atomic, Molecular, and Optical Physics) 72, 062312 (2005).
[29] B. Sklar, IEEE Communications Magazine 21, 6 (1983).
[30] B. Sklar, *Digital Communications*, 2nd ed. (Prentice Hall, Upper Saddle River, New Jersey 07458, 2000).
[31] P. D. Townsend, Nature 385, 47 (1997).
[32] V. Fernandez *et al.*, in *Quantum key distribution in a multi-user network at gigahertz clock rates*, edited by G. Badenes, D. Abbott, and A. Serpenguzel (SPIE, ADDRESS, 2005), No. 1, pp. 720–727.
[33] Nikolay Raychev. Dynamic simulation of quantum stochastic walk. In International jubilee congress (TU), 2012.
[34] Nikolay Raychev. Classical simulation of quantum algorithms. In International jubilee congress (TU), 2012.
[35] Nikolay Raychev. Interactive environment for implementation and simulation of quantum algorithms. CompSysTech'15, DOI: 10.13140/RG.2.1.2984.3362, 2015
[36] Nikolay Raychev. Unitary combinations of formalized classes in qubit space. International Journal of Scientific and Engineering Research 04/2015; 6(4):395-398. DOI: 10.14299/ijser.2015.04.003, 2015.
[37] Nikolay Raychev. Functional composition of quantum functions. International Journal of Scientific and Engineering Research 04/2015; 6(4):413-415. DOI:10.14299/ijser.2015.04.004, 2015.
[38] Nikolay Raychev. Logical sets of quantum operators. International Journal of Scientific and Engineering Research 04/2015; 6(4):391-394. DOI:10.14299/ijser.2015.04.002, 2015.
[39] Nikolay Raychev. Controlled formalized operators. In International Journal of Scientific and Engineering Research 05/2015; 6(5):1467-1469, 2015.
[40] Nikolay Raychev. Controlled formalized operators with multiple control bits. In International Journal of Scientific and Engineering Research 05/2015; 6(5):1470-1473, 2015.
[41] Nikolay Raychev. Connecting sets of formalized operators. In International Journal of Scientific and Engineering Research 05/2015; 6(5):1474-1476, 2015.

[42] Nikolay Raychev. Indexed formalized operators for n-bit circuits. International Journal of Scientific and Engineering Research 05/2015; 6(5):1477-1480, 2015.
[43] Nikolay Raychev. Converting the transitions between quantum gates into rotations. International Journal of Scientific and Engineering Research 06/2015; 6(6): 1352-1354. DOI:10.14299/ijser.2015.06.001, 2015.
[44] Nikolay Raychev. Quantum algorithm for non-local coordination. International Journal of Scientific and Engineering Research 06/2015; 6(6):1360-1364. DOI:10.14299/ijser.2015.06.003, 2015.
[45] Nikolay Raychev. Universal quantum operators. International Journal of Scientific and Engineering Research 06/2015; 6(6):1369-1371. DOI:10.14299/ijser.2015.06.005, 2015.
[46] Nikolay Raychev. Ensuring a spare quantum traffic. International Journal of Scientific and Engineering Research 06/2015; 6(6):1355-1359. DOI:10.14299/ijser.2015.06.002, 2015.
[47] Nikolay Raychev. Quantum circuit for spatial optimization. International Journal of Scientific and Engineering Research 06/2015; 6(6):1365-1368. DOI:10.14299/ijser.2015.06.004, 2015.
[48] Nikolay Raychev. Encoding and decoding of additional logic in the phase space of all operators. International Journal of Scientific and Engineering Research 07/2015; 6(7): 1356-1366. DOI:10.14299/ijser.2015.07.003, 2015.
[49] Nikolay Raychev. Measure of entanglement by Singular Value decomposition. International Journal of Scientific and Engineering Research 07/2015; 6(7): 1350-1355. DOI:10.14299/ijser.2015.07.004, 2015.
[50] Nikolay Raychev. Quantum algorithm for spectral diffraction of probability distributions. International Journal of Scientific and Engineering Research 08/2015; 6(7): 1346--1349. DOI:10.14299/ijser.2015.07.005, 2015.
[51] Nikolay Raychev. Reply to "The classical-quantum boundary for correlations: Discord and related measures". Abstract and Applied Analysis 11/2014; 94(4): 1455-1465, 2015.
[52] Nikolay Raychev. Reply to "Flexible flow shop scheduling: optimum, heuristics and artificial intelligence solutions". Expert Systems; 25(12): 98-105, 2015.
[53] Nikolay Raychev. Classical cryptography in quantum context. Proceedings of the IEEE 10/2012, 2015.