A DNA Arithmetic Logic Unit

F. de SANTIS, G. IACCARINO
Department of Informatica e Applicazioni "R.M. Capocelli"
University of Salerno
Via S. Allende, 84081 – Baronissi
ITALY
fds@unisa.it geniac@tiscali.it http://www.dia.unisa.it/professori/fds/home/index.html

Abstract: - We propose *in vitro* algorithms using Dna strands to perform arithmetic and logical operations. The salient feature of our technique is that we use the same Dna structure to implement each algorithm, as well as to represent inputs and outputs. This allows to reuse output strands as input for others computations. Massive parallelism and great quantitative of Dna in solution provide a lot of advantages: among all the possibility to perform mathematical operations with huge binary numbers which are not representable in a conventional computer.

Key-Words: - ALU, Overflow, Underflow, Floating Point Arithmetic, Dna computing

1 Introduction

One of the earliest attempts to perform arithmetic operations using Dna is shown in [3]. It describes a first algorithm which adds two positive binary numbers by means of Dna strands. This technique yields the correct result for the addition itself, but unfortunately the output strand structure is vastly different from the input strand structure; moreover, the algorithm is feasible only with a few bits numbers. In [5], Ogihara et al. propose a method to realize any Boolean circuit by means of DNA strands: it presents a lot of important theoretical implications but it appears poor of implementative power due to the impossibility of realizing so large Boolean circuits. A more recent technique, described in [1], performs arithmetic operations between positive numbers by means of a recursive procedure such that input and output strands have exactly the same structure. It is suitable for arithmetic operations on integer with fixed bit numbers since the Dna solution can not contain a greater numbers of molecular bits.

In this paper we propose Dna algorithms for logical and arithmetic operations, which work on different size binary numbers, using the same molecular structure for the input and output representation, as well as it happens in a conventional computer. Moreover, we resolve overflow and underflow problems and achieve higher precision in the binary numbers representation. Eventually, we describe all the algorithms and the bio-hardware we need to design a Dna Arithmetic-Logic Unit.

2 Conventional Computer

As it is well known, arthmetic and logical operations are done in a conventional computer by the Arithmetic-Logic Unit (ALU). The unfeasibility to represent binary numbers greater than the maximum or smaller than the minimum allowed by the size of the word machine constitutes a serious difficulty when massive mathematical calculations are requitred since it hinders to perform arithmetic operations with huge integer or long floating point numbers. A not representable number is usually rounded off or truncated. Nowdays, IEEE Standard 754 floating point is the most common representation for real numbers on conventional computers, including Intel-based PC's, Macintoshes and Unix platforms. IEEE floating point numbers have tree basic components: sign, exponent and mantissa. Let α be a generic floating point number. It can be written as:

(10) $$\pm s \times f \times 2^{\pm e}.$$

where s is the number sign, f the mantissa, and e the exponent. The standard uses a nomalized mantissa, a polarized exponent, and a 32 or 64 bit word for the relative encoding (single or double precision, respectively). When a number is greater than the maximum which can be represented, a conventional computer returns an overflow error; when a number is smaller than the minimum which can be represented, it returns an underflow error. Moreover, in IEEE 754 standard there are at least four numerical ranges where results of arithmetic operations might not be represented.

In the following we will show how to implement a Dna based ALU working on logical and arithmetic operations without any problem of overflow and underflow.

3 DNA Algorithms

Before presenting *in vitro* procedures to perform logical operations and integer/ floating point arithmetic, we define a *representation model* for the Dna encoding of binary strings.

3.1 Dna Encoding of Binary Strings

Each binary number is represented by a set of integers indicating the positions where bits are set to 1 [2]. Thus, each binary number is represented as a set of test tubes (each one as a multi set of strings over Γ = {A,C, G, T}) of DNA double strands from *1* to *n*, such that the DNA strand for integer *j* is:

$$ds_X = \updownarrow S_i P_1 \ldots P_k (\alpha_1 \ldots \alpha_z)^j E_0 \qquad (1)$$

with $1 \le j \le 8$. S_i encodes the test tube that contains the binary strand; $P_1 \ldots P_k$ represents the byte in which it is contained the bit; $(\alpha_1 \ldots\alpha_z)^j$ represents the offset into the byte; E_0 is the final sequence, the same for each Dna double strand. An example of Dna double strand is the following:

$$\updownarrow (\mathrm{AAGCTCT}\ ^5)^i\ \mathrm{AAGCTT}\ (\mathrm{CTGCATG}\ ^5)^K\ \mathrm{CTGCAG}\ (\mathrm{GAATTGC}\ ^5\mathrm{T}\ ^5\mathrm{G}\ ^5\mathrm{C})^J\ \mathrm{GAATTC}$$

The sequence AAGCTT is the restriction site for HandIII; CTGCAG is the restriction site for PstI; GAATTC is the restriction site for EcoRI. GAATTC represents the last sequence E_0. In this representation, each sequence has different size with respect to its position in the string. Note that $\alpha_1 \ldots \alpha_z$ is tree times longer than S_i and P_k, so that the strands encoding the last position bits of the byte, will be always the longest in the solution. In this way they will be easily recognizable through only one gel electrophoresis. In this case, this is true if $k \le 4$. In general, it must result that $\alpha_1 \ldots \alpha_z$ is *L-1* times longer than S_i and P_k, where *L* is the maximum value of *k*. This is an important assertion for the next biological operations. Thus, the set of test tube $T[\alpha]_m \ldots T[\alpha]_1$, representing the binary number α, is:

$$T[\alpha]_m \ldots T[\alpha]_1 = \{ds_X\!: x \in X[\alpha]\}. \qquad (2)$$

3.2 Logical Operations

Let $T[\alpha]_m \ldots T[\alpha]_1$ and $T[\beta]_m \ldots T[\beta]_1$ be, the test tubes that encode binary strings α and β respectively, with $m << n$. We propose the following Dna algorithms for logical operations.

3.2.1 NOT Operation

For each $T[\alpha]_i$ the following steps are performed:
Step1. Synthesize the test tube T_i, containing the Dna double strands which encodes all the positions in the string.
Step2. Extract, by affinity purification, all up-strands from $T[\alpha]_i$ (using $\downarrow S_i$) and down-strands from T_i (using $\uparrow S_i$).
Step3. Mix, in T_i, these two extracts so that complementary strands can get annhealed to form stable double strands.
Step4. Extract all single-strands from T_i and pour them in a new test-tube, $T'[\alpha]_i$. The single strands can be complemented by PCR using $\uparrow S_i$ as a primer. At the end of this process, the set of test tubes $T'[\alpha]_m \ldots T'[\alpha]_1$ contains the final result $\neg\alpha$.

3.2.2 AND Operation

For each $T[\alpha]_i$ and $T[\beta]_i$ the following steps are performed:
Step1: Extract all up-strands from $T[\alpha]_i$ (using $\downarrow S_i$) and down-strands from $T[\beta]_i$ (using $\uparrow S_i$).
Step2: Mix, in $T[\beta]_i$, these two extracts, so that complementary strands can get annhealed to form stable double strands.
Step3 : Extract Dna single strands from $T[\beta]_i$. The set of test tubes, $T[\beta]_m \ldots T[\beta]_1$, contains the final result $\alpha \wedge \beta$.

3.2.3 OR Operation

For each $T[\alpha]_i$ and $T[\beta]_i$:
Step1: Pour the content of $T[\alpha]_i$ in- $T[\beta]_i$: The set of test tubes, $T[\beta]_m \ldots T[\beta]_1$, represents the final result $\alpha \vee \beta$.

3.2.4 XOR Operation

For each $T[\alpha]_i$ and $T[\beta]_i$:
Step1: Extract all up-strands from $T[\alpha]_i$ (using $\downarrow S_i$) and down-strands from $T[\beta]_i$ (using $\uparrow S_i$).
Step2: Mix, in $T[\beta]_i$, these two extracts so that complementary strands can get annhealed to form stable double strands.
Step3: Extract all single-strands from $T[\beta]_i$ and pour them in a new test-tube, $T'[\alpha]_i$. The single strands can be complemented by PCR, using $\uparrow S_i$ as a primer. At the end of this process the set of test tubes $T'[\alpha]_m \ldots T'[\alpha]_1$ contains the final result $\alpha \oplus \beta$.

3.2.5 Complexity Analysis

All the logical operations shown above require a precise number of bio-step. The complexity depends exclusively from the number of test tubes representing the binary numbers. Thus the expected number of bio-step for any one of the logical operations shown above is $O(m)$; it becomes $O(1)$, if the number of bits is fixed and contained in just one test tube.

3.3 Integer Arithmetic

Let us now describe how integer arithmetic can be done.

3.3.1 Mathematical Model

As it is shown in [1], we can implement addition and multiplication operations as recursive procedures. Let $\alpha = \alpha_n \ldots \alpha_1$ and $\beta = \beta_n \ldots \beta_1$ be two binary strings, where $\alpha_i, \beta_i \in \{0,1\}$ $\forall i = 1 \ldots n$ and α_1, β_1 are the respective least significances bits. We can define the recursive addition and multiplication procedures for two positive binary numbers, as follows:

Addition

Define $X[\alpha] = \{i\!: \alpha_i = 1\}$ and $X[\beta] = \{j\!: \beta_j = 1\}$ results:

$$\mathbf{Add}(\alpha, \beta) = \mathit{Val}(\mathit{RecursiveAdd}(X[\alpha], X[\beta])); \qquad (3)$$

where

$$\begin{aligned} \mathit{RecursiveAdd}(Y,Z) &= Y \text{ if } Z = \phi \\ &= Z \text{ if } Y = \phi \end{aligned}$$

=RecursiveAdd(($Y \oplus Z$),$(Y \cap Z)^+$) otherwise.

(4) $(Y \cap Z)^+ = \{x+1 : x \in (Y \cap Z)\}$, and

(5) $(Y \oplus Z) = \{x : x \in Y \cup Z \;\; but \;\; x \notin Y \cap Z\}$.

Multiplication

The multiplication procedure can be realized using progressive additions of α values, left shifted:

(6) $$\alpha \times \beta = \sum_{1 \le j \le n, b_j = 1} \alpha \times 2^{j-1}$$

Since a multiplication with a 2^i binary number is performed by a left shift by i positions, we have:

(7) $$X\left[2^j \times \alpha\right] = \left(X[\alpha]\right) + j$$

So the multiplication operation results in:

(8) **Mul**(α, β) = **Add**({ *Val*(X[α] + (*j*-1)) } $_{\beta_j=1}$)

where:

(9) **Add**(α,β,δ) = **Add**(**Add** (α,β), δ) *etc.*

see [1].

3.3.2 Integer Addition

For each $T[\alpha]_i$ and $T[\beta]_i$ with $i = 1...m$, the following steps are performed.

Step1. Check whether any of $T[\alpha]_i$ or $T[\beta]_i$ is empty. If that's true, then $T[\alpha+\beta]_i$ is equal to the *not empty* test tube between $T[\alpha]_i$ or $T[\beta]_i$. Else go to step2.

Step2. Divide the double strands in $T[\alpha]_i$ and $T[\beta]_i$ and extract, through affinity purification, all up-strands from $T[\alpha]_i$ (using $\downarrow S_i$) and down-strands from $T[\beta]_i$, (using $\uparrow S_i$).

Step2.1. Mix these two extracts so that complementary strands can get annhealed to form stable double strands.

Step2.2. Extract single strands from the test tube. Let $T[\alpha]_i$ be the test tube containing double strands of Dna ($T[\alpha]_i \oplus T[\beta]_i$); and $T[\beta]_i$ the test tube containing single strand of Dna ($T[\alpha]_i \cap T[\beta]_i$).

Step3. *Increment by one*. This step implements $(T[\alpha]_i \cap T[\beta]_i)^+$ operation. Add the restriction enzyme EcoRI to $T[\beta]_i$ to cut all the double strands at their 3' end. At the end of this process, the double strands in test tube are:

$\updownarrow S_i\left(\mathrm{CTGCATG}\,^5\right)^k \mathrm{CTGCAG}\left(\mathrm{GAATTGC}\,^5\mathrm{T}\,^5\mathrm{G}\,^5\mathrm{C}\right)^j \mathrm{G} \downarrow \mathrm{AATT}$

Now add to the test tube the following sequence with ligation enzyme:

$$\uparrow \mathrm{AATTGC}\,^5\mathrm{T}\,^5\mathrm{G}\,^5\mathrm{CGAATTC}$$

The result is the following Dna double strand:

$\updownarrow S_i\left(\mathrm{CTGCATG}\,^5\right)^k \mathrm{CTGCAG}\left(\mathrm{GAATTGC}\,^5\mathrm{T}\,^5\mathrm{G}\,^5\mathrm{C}\right)^j \mathrm{GAATT} \uparrow \mathrm{GC}\,^5\mathrm{T}\,^5\mathrm{G}\,^5\mathrm{CGAATTC}$

These strands can, now, be polymerized to form the following double strands:

$\updownarrow \left(\mathrm{AAGCTCT}\,^5\right)^i \mathrm{AAGCTT}\left(\mathrm{CTGCATG}\,^5\right)^k \mathrm{CTGCAG}\left(\mathrm{GAATTGC}\,^5\mathrm{T}\,^5\mathrm{G}\,^5\mathrm{C}\right)^{j+1} \mathrm{GAATTC}$

which represent $x \in (X[\alpha] \cap X[\beta])^+$.

Step3.1. For each operation of this typology, at the most *k* bit (number of bytes of S_i) can pass to the next byte of the string, and only one bit passes from S_i to S_{i+1} test tube. With the help of the gel electrophoresis (see Fig. 1), separate these strands from all the other in the test tube. Let T_i' be the test tube containing the *k* bits, and T_i" the test tube containing the unique bit for S_{i+1}. These strands are:

$$\updownarrow S_i\left(\mathrm{CTGCATG}\,^5\right)^k \mathrm{CTGCAG}\left(\mathrm{GAATTGC}\,^5\mathrm{T}\,^5\mathrm{G}\,^5\mathrm{C}\right)^9 \mathrm{GAATTC}$$

Add the restriction enzyme SalI to T_i' to cut all the double strands in solution. The resultant strands are:

$$\updownarrow \left(\mathrm{AAGCTCT}\,^5\right)^i \mathrm{AAGCTT}\left(\mathrm{CTGCATG}\,^5\right)^k \mathrm{C} \downarrow \mathrm{TGCA}$$

Now add to the test tube the following sequence and ligation enzyme:

$$\uparrow \mathrm{TGCATG}\,^5\mathrm{CTGCAG}\left(\mathrm{GAATTGC}\,^5\mathrm{T}\,^5\mathrm{G}\,^5\mathrm{C}\right)^1 \mathrm{GAATTC}$$

The resultant Dna double strands in solution are the following:

$$\updownarrow S_i\left(\mathrm{CTGCATG}\,^5\right)^{k+1} \mathrm{CTGCAG}\left(\mathrm{GAATTGC}\,^5\mathrm{T}\,^5\mathrm{G}\,^5\mathrm{C}\right)^1 \mathrm{GAATTC}$$

Add the restriction enzyme HandIII to T_i" to cut all the double strands at their 3' end. The resultant is the following:

$$\updownarrow \left(\mathrm{AAGCTCT}\,^5\right)^i \mathrm{A} \downarrow \mathrm{AGCT}$$

Now, add to the test tube the following sequence and ligation enzyme:

$\uparrow \mathrm{AGCTCT}\,^5\mathrm{AAGCTT}\left(\mathrm{CTGCATG}\,^5\right)^1 \mathrm{CTGCAG}\left(\mathrm{GAATTGC}\,^5\mathrm{T}\,^5\mathrm{G}\,^5\mathrm{C}\right)^1 \mathrm{GAATTC}$

The result is the following:

$\updownarrow \left(\mathrm{AAGCTCT}\,^5\right)^{i+1} \mathrm{AAGCTT}\left(\mathrm{CTGCATG}\,^5\right)^1 \mathrm{CTGCAG}\left(\mathrm{GAATTGC}\,^5\mathrm{T}\,^5\mathrm{G}\,^5\mathrm{C}\right)^1 \mathrm{GAATTC}$

Step4. Go back to step1.

3.2.3 Integer Subtraction

The subtraction operation α-β can be done using the following idea:

Step1. By gel electrophoresis of the test tubes $T[\alpha]_m$ and $T[\beta]_m$, determine whether α≥β or β≥α. Assume that α≥β.

Step2. Construct a set of test tubes $T_m...T_1$ which consist of Dna double strands for all $i \in \{1...n\}$.

Step3. $\forall i=1...m$, obtain $T^1_i = T[\beta]_i - T_i$. The Dna procedure is the same of the NOT operation, described above.

Steo4. $\forall i=1...m$, Perform integer addition operation with $T[\alpha]_i$ and T^1_i and keep the result in T^1_i.

Step5. Let $T^{[1]}_1$ be the test tube containing only one structure of Dna strands:

$\updownarrow$ S_1CTGCATG ^{5}CTGCAGGAAT TGC ^{5}T ^{5}G ^{5}CGAATTC that encodes the position 1. Perform addition operation with $T^{[1]}{}_1$ and $T^1{}_i$, $\forall i=1...m$.

Step6. Extract, from $T^{[1]}{}_m$, the strands that encodes the position $n+1$. The residual set of test tubes give the result α-β.

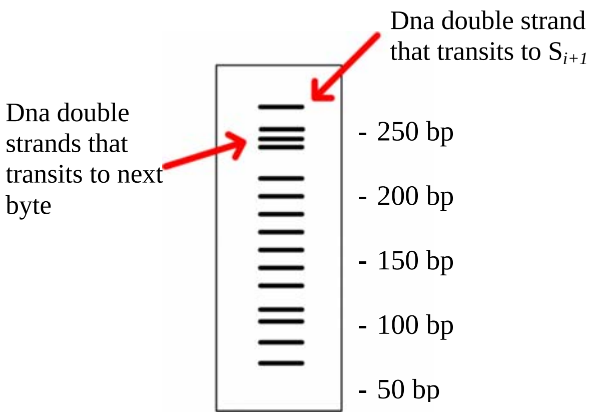
Dna double strand that transits to S_{i+1}
Dna double strands that transits to next byte
- 250 bp
- 200 bp
- 150 bp
- 100 bp
- 50 bp

Fig. 1 – An example of S_i gel-image.

3.3.4 Integer Multiplication

For each $T[\alpha]_i$ and $T[\beta]_i$ with $i = 1...m$, the following steps are performed.

Step1. For each $j\in X[\beta]$, construct the test tubes $T_j[\alpha]$ similar to Step3 for integer addition. If $j > 2$ we add the following sequence to the solution:

$\uparrow$ AATTGC ^{5}T ^{5}G ^{5}C(GAATTGC ^{5}T ^{5}G ^{5}C)$^{j-2}$ GAATTC ;

if j=2, add the sequence $\uparrow$ AATTGC ^{5}T ^{5}G ^{5}CGAATTC . For j=1, $T_1[\alpha]=T[\alpha]_1$, so it is the test tube that encodes S_1.

Step2. If the test tubes obtained in step1 are $T_{j1}[\alpha]$, $T_{j2}[\alpha]$, $T_{j3}[\alpha]$, ..., do the following:

Step2.1. Perform integer addition, described above, concurrently with successive pairs of tubes ($T_{j1}[\alpha]$,$T_{j2}[\alpha]$), ($T_{j3}[\alpha]$,$T_{j4}[\alpha]$) etc. Let the result be kept in $T^1{}_{j1}[\alpha]$, $T^1{}_{j2}[\alpha]$, $T^1{}_{j2}[\alpha]$ etc.

Step2.2. Repeat step2.1 until single tube *T* is obtained.

3.3.5 Complexity Analysis

Addition - As shown in [1], the expected number of bio-step is $O(\log_2 n)$ for each test tube forming the binary numbers. Thus, the complexity of this algorithm is $O(m\cdot\log_2 n)$; it becomes $O(\log_2 n)$, if the number of bits is finite and contained in just one test tube.

Subtraction - Steps 1 and 6 require $O(1)$. Step2 is the same for each computation, so it can be pre-computed in $O(m)$ bio-steps. Steps 3 and 5 require $O(m)$. Step4 requires $O(m\cdot\text{lon}_2\ n)$. In total the algorithm expects this number of bio-step: $O(1)+O(m)+O(m\cdot\text{lon}_2 n)$ which is equal to $O(m\cdot\text{lon}_2\ n)$. It becomes $O(\text{lon}_2\ n)$, if the number of bits is fixed, and it is contained in just one test tube.

Multiplication - The expected number of integer addition, for each test tube *i*, is $\log_2 n$ -2 [2]. Each addition spends $O(\log_2 n)$, so the expected number of bio-step, for each test tube, will be $O((\text{lon}_2 n)^2)$. In total it is $O(m\cdot(\text{lon}_2 n)^2)$.

3.4 Floating Point Arithmetic

As mentioned above, the IEEE 754 standard uses a normalized mantissa and polarized exponent in the representation of floating point numbers. We use a similar representation for our Dna floating point numbers. The exponent's value is polarized and *f* is normalized in $0\leq f<1$. The value of the polarization number is 2^q-1, where *q* is the number of bit of the exponent. Using the Dna representation of binary string, shown above, we can divide the set of test tubes for α as follow: $\alpha = T[\alpha]_{sign}\ T[\alpha]_{exp}\ T[\alpha]_{m....}T[\alpha]_1$, where $T[\alpha]_{sign}$ encodes the sign of α. $T[\alpha]_{exp}$ contains molecular bits for the exponent. $T[\alpha]_{m....}T[\alpha]_1$ is the set of test tubes that encodes mantissa *f*. As it is happens in IEEE 743 standard , we need to represent with Dna molecules some particular values. This value are 0, ±∞ and NaN (*Not a Number*). Usually, these values are the results of arithmetic operations between particular operands. In the following , the Dna representation of these particulars values is described:

(11) $\alpha=0 \rightarrow T[\alpha]_{sign}=\phi,\ T[\alpha]_{exp}=\phi,\ T[\alpha]_m=...=T[\alpha]_1=\phi;$

(12) $\alpha=\pm\infty \rightarrow T[\alpha]_{sign}\neq\phi,\ T[\alpha]_{exp}\neq\phi,\ T[\alpha]_m=...=T[\alpha]_1=\phi;$

(13) $\alpha=\text{NaN} \rightarrow T[\alpha]_{sign}\neq\phi,\ T[\alpha]_{exp}=\phi,\ T[\alpha]_m=...=T[\alpha]_1=\phi.$

We will show Dna procedures to perform addition, subtraction and multiplication with two floating point numbers.

3.4.1. Floating Point Addition

The following steps are performed.

Step1. By gel electrophoresis of the test tubes $T[\alpha]_{exp}$ and $T[\beta]_{exp}$, determine whether $e_\alpha\geq e_\beta$ or $e_\beta\geq e_\alpha$. Assume that $e_\alpha\geq e_\beta$.

Step2. Increase the position of the molecular bits in $T[\alpha]_{m....}T[\alpha]_1$ as times as the value of e_α-e_β. This step is the same of Step3 and Step3.1. of integer addition algorithm, shown above. The result is the left shift of the bits of the mantissa of α toward greatest positions.

Step3. Perform integer addition operation with $T[\alpha]_{m....}T[\alpha]_1$ and $T[\beta]_{m....}T[\beta]_1$. The result is a set of this kind: $T[\alpha+\beta]_{m....}T[\alpha+\beta]_1$.

Step4. *Normalization*. By gel electrophoresis of the test tube $T[\alpha+\beta]_m$, compare the longest double strand in solution with the longest bit in $T[\alpha]_m$. If the new strands is longer than the oldest, then $e_{\alpha+\beta}$= $e_{\alpha+\beta}$+1; else the value is already normalized.

3.4.2 Floating Point Subtraction

The following steps are performed.

Step1. By gel electrophoresis of the test tubes $T[\alpha]_{exp}$ and $T[\beta]_{exp}$, determine whether $e_\alpha \geq e_\beta$ or $e_\beta \geq e_\alpha$. Assume that $e_\alpha \geq e_\beta$.
Step2. Increase the position of the molecular bits in $T[\alpha]_{m....}T[\alpha]_1$ as times as the value of $e_\alpha - e_\beta$. The result is the left shift of the bits of the mantissa of α toward greatest positions.
Step3. Perform integer subtraction operation with $T[\alpha]_{m....}T[\alpha]_1$ and $T[\beta]_{m....}T[\beta]_1$. The result is a set of test tubes of this kind: $T[\alpha-\beta]_{m....}T[\alpha-\beta]_1$.
Step4. *Normalization*. the value is already normalized because $0 \leq f < 1$, so even if there are a lot of 0-bits, they aren't represented in solution, so that the normalization is not required.

3.4.3 Floating Point Multiplication

Step1. If all test tubes $T[\alpha]_m...T[\alpha]_1$ or $T[\beta]_m...T[\beta]_1$ are empty, then the resultant value of the operation is 0. In fact $T[\alpha]_{sign}=T[\alpha]_{exp}=T[\alpha]_m=...=T[\alpha]_1=\phi$ encodes the value 0. Else go to step2.
Step2. Perform the integer addition algorithm with $T[\alpha]_{exp}$ and $T[\beta]_{exp}$.
Step3. Perform the multiplication algorithm with $T[\alpha]_{m....}T[\alpha]_1$ and $T[\beta]_{m....}T[\beta]_1$. The result is a set of test tubes of this kind: $T[\alpha\cdot\beta]_{m....}T[\alpha\cdot\beta]_1$.
Step4. *Normalization*. the value is already normalized because $0 \leq f < 1$, so even if there are a lot of 0-bits, they aren't represented in solution, so that the normalization is not required.
Step5. Compare $T[\alpha]_{sign}$ and $T[\beta]_{sign}$. The final test tube $T[\alpha\cdot\beta]_{sign}$ will determine the sign of value $\alpha\cdot\beta$:
If $T[\alpha]_{sign} = T[\beta]_{sign} = \phi$ then $T[\alpha]_{sign} = \phi$ (*positive*).
If $T[\alpha]_{sign} = T[\beta]_{sign} \neq \phi$ then $T[\alpha]_{sign} = \phi$ (*positive*).
If $T[\alpha]_{sign} \neq T[\beta]_{sign}$ then $T[\alpha]_{sign} \neq \phi$ (*negative*).

3.4.4 Complexity Analysis

Addition – The expected bio-steps needed in addition operation derive from the bio-step required from each computational step. In step1 we make only one gel electrophoresis, so the complexity is $O(1)$. In step2 we perform an integer subtraction, $O(\log_2 q)$, and a left shift of the mantissa $O(q)$. Then the complexity in step2 is $O(\log_2 q)+O(q)$, where q is the number of bits required for the representation of the exponent. In step3 we perform an integer addition, that takes $O(m\cdot\log_2 n)$. In step4 we make a gel electrophoresis and an increment of 1; so the complexity is $O(\log_2 q)$. In conclusion, the expected bio-steps in this algorithm are: $O(1)+O(\log_2 q)+O(q)+ O(m\cdot\log_2 n)+O(\log_2 q)$ which is equal to $O(m\cdot\log_2 n)$. It becomes $O(\log_2 n)$, if the bits of the mantissa are fixed and contained in an just one test tube.
Subtraction – The expected bio-steps needed in subtraction are the same as in floating point addition. The only difference depends on the integer subtraction operations done in step3. This complexity is $O(m\cdot\log_2 n)$, so that the expected bio-steps in floating point subtraction are $O(m\cdot\log_2 n)$. It becomes $O(\log_2 n)$, if the bits of the mantissa are fixed, and contained in an just test tube.
Multiplication – The expected bio-steps in steps 1 and 2 are $O(\log_2 q)$. In step3 we perform an integer multiplication, that requires $O(m\cdot(\mathrm{lon}_2 n)^2)$. In steps 4 and 5 we perform only $O(1)$ biological operations. In conclusion, the expected bio-steps in this algorithm are: $O(\log_2 q)+O(m\cdot(\mathrm{lon}_2 n)^2)+ O(1)$ that is equal to $O(m\cdot(\mathrm{lon}_2 n)^2)$. It becomes $O(\mathrm{lon}_2 n)^2$, if the bits of the mantissa are fixed, and contained in just one test tube.

4 Conclusion

The purpose of this paper was to show a mathematical application of Dna computing. We have described *in vitro* procedures to perform logical and arithmetic operations using Dna strands. Hence, we consider a genetic laboratory as a Dna made Arithmetic-Logic Unit, where human operators implement bio-chemistry procedures to perform mathematical operations. The power of the Dna Computing consists in the capability to represent, and compute, huge binary numbers, or highly small ones which are impossible to consider in a conventional computer. With a Dna-based *ALU,* we can perform logical and arithmetic operations on any binary number, whatever it is its bit number. In other words, we are able to calculate mathematical operations with unlimited decimal digits. It is worthy noticing that " unlimited" does not mean "endless", but unfixed number of bits. If we consider that 50g of Dna contains 10^{33} molecules, it is clear that in few grams of DNA we can encode a great deal of molecular bits. Moreover, since we encode only 1-bits, the number of bit in solution grows even more. This feature is a beautiful remedy for computational problems, like overflow, underflow, rounding and truncation, which all depend on the fixed number of bits reserved to the representation in conventional computing.

References:

[1] R. Barua, Binary Arithmetic for DNA Computer, *8th International workshop on DNA Based Computers: Dna Computing*, 2002, pp. 124-132.
[2] S. Biswas, *Computing with Bio-Molecules. Theory and Experiment*, Ed G. Paun , 1998.
[3] F. Guarnieri, Making DNA add, *Science,* No. 273, 1996, pp. 220-223.
[4] W. Kahan, IEEE Standard 754 for Binary Floating Point Arithmetic, *Lecture Notes on status of IEEE 754*, University of California Berkeley CA, May 1996.
[5] M. Ogihara, Simulating Boolean Circuits on a DNA Computer, *Tech Report TR631*, Department of C.Sc., University of Rochester, Aug. 1996.