Binary Arithmetic for DNA Computers

Rana Barua[1]
Division of Theoretical Statistics and Mathematics,
Indian Statistical Institute,
203 B.T. Road, Calcutta 700 035, India.
e-mail: rana@isical.ac.in
and
Janardan Misra[2]
Texas Instruments India Ltd.
Wind Tunnel Road
Bangalore 560017, India

Abstract

We propose a (recursive) DNA algorithm for adding two binary numbers which require $O(\log n)$ bio-steps using only $O(n)$ different type of DNA strands, where n is the size of the binary string representing the larger of the two numbers. The salient feature of our technique is that the input strands and the output strands have exactly the same structure which makes it fully procedural unlike most methods proposed so far. Logical operations of binary numbers can easily be performed by our method and hence can be used for cryptographic purpose.

[1]corresponding author
[2]work done at ISI

Binary Arithmetic for DNA Computers

Rana Barua[‡] Janardan Misra[§]

March 6, 2002

1 Introduction

One of the earliest attempts to perform arithmetic operations (addition of two positive binary numbers) using DNA is by Guarneiri *et al* [11], utilizing the idea of encoding differently bit values 0 and 1 as single-stranded DNAs, based upon their positions and the operand in which they appear. This enabled them to propagate carry successfully as horizontal chain reaction using intermediate place holders because of the presence of appropriate complementary substrands, which annealed together. PCR then allowed one to insert correct value of carry and to further propagate it. Though their technique yields the correct result of addition of two given binary numbers, it is highly non-procedural in nature since the output strands are vastly different in structure from the input strands (which themselves are coded differently).

The later attempts were by Vineet Gupta *at al*[14]. They performed logic and arithmetic operations using the fixed bit encoding of the full corresponding truth tables. They construct the strands for bits in first operand (level one) and corresponding to each bit value (0 or 1), all possible bit values (00, 01, 10, 11)[1] in second operand(level two) such that in the next phase when first operand - strands are pored into the pot containing all possible second operand - strands (including the correct one) annealing results in a structure which can be interpreted according to the type of the operation. In case of arithmetic operation, in later stages of computation, they add all possible intermediate results and successively propagate carry from the lowest weighted bit to the highest weighted bit.

Though the encoding works well with logic operations, the arithmetic operation does not seem to be so easy as the technique requires that all

[‡]Indian Statistical Institute, Calcutta, India; e-mail:rana@isical.ac.in

[§]Texas Instruments, Bangalore, India; e-mail:janmisra@india.ti.com

[1]In fact they use along with usual dnts other nucleotides like Uracil(U), 2,6-diaminopurine(P) to achieve some additional complementary structures

the possible intermediate results be coded and added manually one by one during processing. This is a labor intensive and time consuming job.

Other later attempts are due to Z. Frank Qiu and Mi Lu [16], which use substitution operation to insert results (by encoding all possible outputs of bit by bit operation along with second operand) in the operand strands. Though they propose to extend their method to higher radix like octal, decimal etc; the possible number of encoding of different intermediate results seems to be exponential. Moreover, the cleansing operation which makes the output similar to first input, for reuse in further operations, is not an error resistant operation.

Ogihara, Ray and Amos, Dunne [15, 1] present methods to realize any Boolean circuit (with bounded fan-in) using DNA strands in a constructive fashion. Here the problem is that of constructing large Boolean circuits for arithmetic operations, (manually or automating the process) rendering the technique of theoretical importance only.

Other new suggestions to perform all basic arithmetic operations are by Atanasiu [4] using P systems, and by Frisco [9] using splicing operation under general H systems, and by Hubert and Schuler [13]

In this article, we describe a simple, new and efficient recursive DNA computing technique for basic arithmetic operations like addition, multiplication and subtraction of binary numbers, based upon their set representations. The salient features of our technique are the following.

1. Our method is fully procedural *i.e.*, the structure of the output strands is exactly similar to that of the input strands. Thus the result can be further reused without any changes, making iterative as well as parallel operations feasible.

2. The number of different DNA strands required is at most of the order of size of the binary number. That means it avoids the problem of increasing volume.

3. The number of bio-steps required by the technique for addition is, on the average, $O(\log_2 n)$ and for multiplication it is $O((\log_2 n)^2)$, where n is the size of the binary numbers.

4. All logical operations on binary numbers can be performed very easily using our method.

2 Recursive DNA Arithmetic

2.1 Underlying Mathematical Model

Let $\alpha = \alpha_n \ldots \alpha_1$ and $\beta = b_n \ldots \beta_1$ be two n -bit binary numbers, where α_i, $\beta_i \in \{0,1\}$ for all $1 \leq i \leq n$ and α_1, β_1 denote the least significant bits. Let

$$X[\alpha] = \{i{:}\, \alpha_i = 1\}\, and\, X[\beta] = \{j{:}\, \beta_j = 1\};$$

i.e. they are the sets containing the positions where binary representation these numbers sets bits to 1. For any set Z of integers and any integer i, we define

$$Z + i = \{z + i{:}\, z \in Z\}\, and\;\; let\, Z^+ = Z + 1.$$

For any two sets of integers X_1, X_2 we define

$$X_1 \oplus X_2 \;\; = \;\; \{x{:}\, x \in X_1 \cup X_2 \;\; but \;\; x \notin X_1 \cap X_2\} \;\; (symmetric \;\; difference).$$

We denote by $Val(X)$ the binary number represented by set X (of positive integers); *e.g.* $Val(X[\alpha]) = \alpha$ and $Val(X[\beta]) = \beta$ and $Val(\phi) = 0$.
In terms of this symbolism we can state the abstract recursive procedures for addition and multiplication of the two given positive binary numbers as follows.

Addition.
$\mathbf{Add}(\alpha, \beta) = Val(RecursiveAdd(X[\alpha], X[\beta]))$; where

$$\begin{aligned} RecursiveAdd(Y, Z) &= Y \text{ if } \;\; Z = \phi \\ &= Z \text{ if } \;\; Y = \phi \\ &= RecursiveAdd((Y \oplus Z), (Y \cap Z)^+ \;); \text{ other-} \end{aligned}$$

wise.

It is easy to see that this procedure terminates and that for two binary numbers α, β, $\mathbf{Add}(\alpha, \beta)$ represents $\alpha + \beta$. This follows from the fact that

$$\alpha + \beta = \alpha \oplus \beta + 10 \times (\alpha \wedge \beta),$$

where $\oplus$ denotes bitwise addition modulo 2 and $\wedge$ denotes bitwise multiplication.(Note that 10 above is the binary representation of 2.)

Multiplication. The multiplication procedure can be realized, using successive additions of left shifted α's, according to the following formula.

$$\alpha \times \beta = \sum_{1 \leq j \leq n, b_j = 1} \alpha \times 2^{j-1},$$

where we also use α *etc.* to denote the integer whose binary representation is α. Since multiplication of a binary number with power of 2, say 2^j, is obtained by a left shift of the number by j, we have

$$X[2^j \times \alpha] = (X[\alpha]) + j.$$

Hence, we have

$$\mathbf{Mul}(\alpha, \beta) = \mathbf{Add}(\{Val(X[\alpha] + (j-1))\}_{\beta_j=1}),$$

where $\mathbf{Add}(\alpha, \beta, \gamma) = \mathbf{Add}(\mathbf{Add}(\alpha, \beta), \gamma)$ *etc.*

Subtraction operation for integer arguments can be performed as 2's complement addition ([12] p 23).

Division. Once we can perform addition and subtraction operation then mapping of division operation in terms of these can be done using any of the standard digital arithmetic techniques ([12] p 250). For instance we may consider nonrestoring division ([12] p 253) which requires an average of n additions or substractions.

2.2 DNA Algorithm

Since the procedure described above is recursive in nature and as can be seen easily in context of currently available DNA tool - kit operations and other high level operations as suggested in [6], the most important operation to be realized is incrementing all the integers in the sets like $X[\alpha]$ by one. Actually the coding of numbers and various other steps basically rest upon the ease with which this step can be realized. Keeping this point in mind we propose the following DNA algorithm:

DNA Encoding of Binary Numbers

Note that each binary number is represented by a set of integers which are positions where bits are set to 1. Thus each binary number is represented as a test tube (a multi set of strings over $\Gamma = \{A, C, G, T\}$) of DNA double strands encoding the integers (positions where bit is set to 1) from 1 to n, such that the DNA strand for integer i is (cf. [5] for notation)

$$ds_i = \updownarrow S_0(GAATTGC^5)^i GAATTCS_1.$$

(Note that $\updownarrow GAATTC$ is the restriction site for EcoRI.) Here S_0, S_1 may be any suitable 20 to 30 base-pair long DNA double strand not containing $\updownarrow GAATTC$ as a substrand.

(An alternate way of encoding position i, for minimizing errors due to sliding, is discussed later in section 2.5)

Thus the test tube $T[\alpha]$ representing binary number α is

$$T[\alpha] = \{ds_i : \ i \in X[\alpha]\}.$$

We first present the DNA-based implementation for addition.

1. **Addition**

   Step0. Check whether any of $T[\alpha]$ or $T[\beta]$ is empty. If yes, then the other tube contains the final result (which can be obtained by detecting the presence of all different strands using gel electrophoresis or extraction technique).
   Else go to step 1.
   Initially $T[\alpha]$ and $T[\beta]$ represent the test tubes encoding α and β respectively.

   Step1. Melt the double strands in $T[\alpha]$ and $T[\beta]$ to extract up-strands (using $\downarrow S_0$) from $T[\alpha]$ and down-strands (using $\uparrow S_0$) from $T[\beta]$. Now mix these two extracts so that complementary strands can get annealed to form stable double strands. As can be seen, the resulting double strands in the tube are exactly those coming from $T[\alpha] \bigcap T[\beta]$ and single strands are those coming from $T[\alpha] - T[\beta]$ (up-strands containg $\uparrow S_0$) and from $T[\beta] - T[\alpha]$ (down-strands with $\downarrow S_0$). Using standard DNA toolkit operations, single strands can be separated from double strands and then, using PCR, the single strands can be complemented using S_0, S_1 as primers. Denote by $T[\alpha]$ the tube containing these double strands obtained after PCR (i.e. $T[\alpha] \oplus T[\beta]$) and by $T[\beta]$ the annealed double strands (i.e. $T[\alpha] \bigcap T[\beta]$).

   Step2.*(Increment by One)*. Add restriction enzyme EcoRI to $T[\beta]$ to cut all the double strands at their $3'$ end. This restriction enzyme activity leaves double strands with $5'$ hanging ends, of the form

   $$\updownarrow S_0(GAATTGC^5)^i G \downarrow AATT.$$

   Now up-strands

   $$\uparrow AATTGC^5GAATTC$$

   and ligation enzyme are added to the test tube which results in the strands of the form $\updownarrow S_0(GAATTGC^5)^i GAATT \uparrow GC^5GAATTCS_1$. These strands can now be polymerased to form

   $$\updownarrow S_0(GAATTGC^5)^{i+1} GAATTCS_1.$$

   Thus $T[\beta]$ contains strands representing $(X[\alpha] \bigcap X[\beta])^+$.

   Step3. Go back to step0.

2. **Multiplication**

   Step1. For each $j \in X[\beta], \ j \geq 2$, construct test - tubes $T_j[\alpha]$ similar to Step 2 above with the difference that for annealing we add $\uparrow AATTGC^5(GAATTGC^5)^{j-2}GAATTCS_1$ instead of adding $\uparrow AATTGC^5GAATTCS_1$ (add it when j = 2).
   For $j = 1$, take $T_1[\alpha] = T[\alpha]$.

   Step2. If the test tubes obtained in step1 are $T_{j_1}[\alpha], T_{j_2}[\alpha], T_{j_3}[\alpha] \cdots$, do the following:
   Step2.1 Perform Addition operation (described above) concurrently with successive pairs of tubes $(T_{j_1}[\alpha], T_{j_2}[\alpha]), (T_{j_3}[\alpha], T_{j_4}[\alpha]) \cdots$ Let the result be kept in $T^1_{j_1}[\alpha], T^1_{j_2}[\alpha], \ldots$.
   Step2.2 Repeat Step2.1 until a single tube T is obtained.

3. **Subtraction**

   The subtraction operation can be done utilizing ideas from the conventional digital arithmetic, that is to say as per 2's complement method. To perform $\alpha - \beta$ we do the following:
   Step0 By gel electrophoresis, determine whether $\alpha \geq \beta$ or $\beta \geq \alpha$. Assume $\alpha \geq \beta$.

   Step1. Construct $T[\alpha], T[\beta]$ and T that consists of ds_i for all $i \in \{1, \ldots, n\}$.

   Step2. Obtain $T_1 = T - T[\beta]$ as described above in Addition operation or as a set extract operation described in [6].

   Step3. Perform addition operation with $T[\alpha]$ and T_1 and keep the result in T_1.

   Step4. Perform addition operation with T_1 and $T[1]$ (the latter consisting of only one type of DNA strands *viz.* $\updownarrow S_0GAATTGC^5GAATTC$ encoding position 1).

   Step5. Extract the DNA strands encoding $n + 1$. The residual test tube gives the desired result.

4. **Logical operations** on binary strings can easily be performed by our method.

   To obtain the bitwise **OR** of α and β form the test tubes $T[\alpha]$ and $T[\beta]$ and mix them together. The resulting tube represents $\alpha\,\mathbf{OR}\,\beta$.

   To obtain bitwise **XOR**, construct $T[\alpha] \oplus T[\beta]$ as in Step1 of *Addition*. The resulting tube represents $\alpha\,\mathbf{XOR}\,\beta$.

The bitwise **AND** is obtained by forming $T[\alpha] \bigcap T[\beta]$ as in Step1 of *Addition* using only set extract operations.

To obtain the **NAND** of two n-bit strings α and β, simply construct the test tubes $T[\alpha], T[\beta]$ and the tube T' consisting of strands encoding *all* the integers from $1, \ldots, n$. Then obtain the tube T'' representing $T[\alpha] \bigcap T[\beta]$ as in Step1 of *Addition*. Now obtain $T = T' - T''$ by set extraction as in [6]. T represents the desired result.

2.3 Use in Cryptography

Due to the simplicity of performing XOR (needing only constant number of biosteps), one can easily implement the Vernam one-time-pad (cf. [17]). Let α (in binary) be the plaintext and κ the random key of the same length as α. The random key can be constructed by randomly synthesising a strand consisting of two nucleotides, say A and G standing for 0 and 1 respectively. To encrypt, the sender constructs the test tube $T = T[\alpha] \oplus T[\kappa]$ and transmits T. On receiving T, the receiver decrypts by constructing $T' = T \oplus T[\kappa]$. One can recover α from T' by reading its contents. In view of the ease with which both encrypting and decryption can be carried out (using only a few extractions), we feel that our method is much simpler than what is proposed in Gehani, LaBean and Reif [10].

2.4 Complexity Analysis

Time Complexity

Addition. As each level of recursion in addition operation involves a fixed number of bio steps, therefore the total number of steps depends on the number of recursion levels in the abstract model. We shall compute the expected number of bio-steps for two random n-bit numbers α, β. Since the probability that at each position bit will be set to 1 (or 0) is 1/2, expected number of 1s' in any randomly chosen binary number of length n are n/2. Similarly probability that at any position both the numbers set the bit to 1 is $(1/2) \times (1/2) = (1/4)$. Therefore, expected number of positions where both the numbers set bits to 1 is $n/4$, which is the expected size of $X[\alpha] \bigcap X[\beta]$ and consequently of $(X[\alpha] \bigcap X[\beta])^{+}$. Similarly, the probability that at any position both the numbers set bits differently is $(1/2).(1/2) + (1/2).(1/2) = (1/2)$ so that the expected number of 1's in $\alpha \oplus \beta$ as n/2. This is same as the expected number of integers in $X[\alpha] \oplus X[\beta]$. Now for 2nd level of recursion, the probability that position number i is present in both the sets from the 1st step is $(1/2) \times (1/4) = (1/8)$. Hence, expected number of integers after set intersection and shifting by one is $n/8$; while expected number of integers in symmetric difference of the sets is $n/2$ since, probability that an integer is present in only one of the two sets is $(1/2).(1/4) + (1/2).(3/4) = (1/2)$. Following the same argument, it can be seen that after ith level of recursion, the expected size of set resulting after set intersection and shifting will be $n/2^{i+1}$. Therefore, the expected number of recursion levels will be $[\log_2 n] - 1$. Thus the expected number of bio-steps needed in the addition operation is $O(\log_2 n)$.

Multiplication. In case of multiplication, the expected number of addition that has to be performed is $(\log n - 2)$. Hence the expected number of bio-steps will be $O((\log n)^2)$.

Subtraction. Since subtraction involves only two additions and a constant number of bio-steps, the expected number of bio-steps in this case is also $O(\log n)$.

Logical Operations. Each logical operation needs a constant (at most 7 or 8) number of bio-steps and is independent of the size of the strings (except for preprocessing).

Volume Complexity At no stage do we require some strand to be destroyed or filtered out. So the total number of strands remains more of less constant.

2.5 Errors

Errors in DNA computing experiments are of primary concern and the issue has been looked into by many researchers [7, 2, 3, 8]. We shall consider the possible errors that may take place in the Addition algorithm. The first source of error is the extract operation which is not error-free. At the cost of more time and space, one may use *compound extract* instead of simple extract as explained in [8]. This could reduce the error to a tolerable level. A more serious source of error seems to be sliding or partial annealing [18] which could take place because of periodic nature of our coding. Partial annealing can be considerably reduced if one adopts the following coding.

$$ds_i = \updownarrow S_0 B_0 S_1 C^5 B_1 S_1 C^5 \ldots . B_{i-1} S_1 C^5 B_i S_1.$$

One can then carry out Step2 of the *Addition* algorithm, using only extracts, by the method of [6] concurrently for all strands as follows.

Heat the test tube and then extract upstrands

$$\uparrow S_0 B_0 S_1 C^5 B_1 S_1 C^5 \ldots . B_{i-1} S_1 C^5 B_i S_1$$

using the downstrands $\downarrow S_0$. Then add downstrands $\downarrow B_i S_1 C^5 B_{i+1} S_1$, *for all* $i < n$. Annealing takes place to form strands of the form
$\uparrow S_0 B_0 S_1 C^5 B_1 S_1 C^5 \ldots . B_{i-1} S_1 C^5 \updownarrow B_i S_1 \downarrow C^5 B_{i+1} S_1$.
Polymerase with then yield complet double strands of the form

$$\updownarrow S_0 B_0 S_1 C^5 B_1 S_1 C^5 \ldots . B_{i-1} S_1 C^5 B_i S_1 C^5 B_{i+1} S_1.$$

Thus every strands ds_i in the tube will be replaced by ds_{i+1}. The remaining partially annealed or unannealed strands, if any, may be filtered out using extracts several times.

3 Conclusion:

We have present methods for carrying out arithmetic and logical operations which can be easily implemented in the DNA computing paradigm. Moreover, we have discussed ways of reducing errors by changing our codings and using only compound extract. This is, however, at the expense of more time and space. Since it is simple to perform XOR using only constant number of bio-steps, it has the potential for being used in cryptograhic implementation of Vernam one-time-pad.

Acknowledgement: The first author would like to thank Zhongwei Li of *Dupont* for many useful discussions that led to significant improvements.

References

[1] M.Amos and P.E.Dunne, DNA Simulation of Boolean Circuits, **Tech Report CTAG-97009**, Dept of Computer Science, University of Liverpool, Dec 1997.

[2] M.Amos, S.Wilson, D.A.Hodgson, G.Owenson and A.Gibbons, Practical Implementation of DNA Computation. In: *Proc 1st International Conference of Unconventional Models of Computation*, Aukland, N.Z., Jan 1998, pp 1-18.

[3] Y.Aoi, T.Yoshinobu, K.Tanizawa, K.Kinoshita and H.Iwasaki, Ligation Errors in DNA Computing. In: *Proc 4th DIMACS Workshop on DNA Based Computers*, U Penn, 1998, pp 181-187.

[4] A.Atanasiu, Arithmetic with Membrames. In: *Proc of the Workshop on Mutiset Processing*, Curtea de Arges, Romania, Aug 2000, pp 1-17.

[5] S.Biswas, A Note on DNA Representation of Binary Strings. In: *Computing with Bio-Molecules. Theory and Experiments*, Ed G.Paun, 1998, pp 153-157.

[6] D.Boneh, C.Dunworth, R.Lipton and J.Sgall, On Computational Power of DNA, **Princeton CS Tech Report No. CSTR49995**, 1995.

[7] D.Boneh, C.Dunworth, J.Sgall and R.Lipton, Making DNA Computers Error Resistant. In: *Proc 2nd DIMACS Workshop on DNA Based Computers*, Princeton, 1996, pp 102-110.

[8] K.Chen and E.Winfree, Error Correction in DNA Computing: Misclassification and Strand Loss. *Proc of the 5th DIMACS Workshop on DNA Based Computers*, MIT, Cambridge, 1999, pp 49-63.

[9] P.Frisco, Parallel Arithmetic with Splicing. *Romanian Journal of Information Science and Technology*, **3**, 2000, pp 113-128.

[10] A.Gehani, T.H.LaBean and J.H.Reif, DNA-based Cryptography, In: *Proc of the 5th DIMACS Workshop on DNA Based Computers*, MIT, Cambridge, 1999.

[11] F.Guarneiri, M.Fliss and C.Bancroft, Making DNA Add. *Science* **273**, 1996, pp 220-223.

[12] J.P.Hayes, *Computer Architecture and Organization.* McGraw-Hill International, Singapore, 2*nd* ed. 1988.

[13] H.Hug and R.Schuler, DNA Based Parallel Computation of Simple Arithmetic. In: *Proc of 7th DIMACS Workshop on DNA Based Computers*, Tampa, 2001, pp 159-166.

[14] V.Gupta, S.Parthasarathy and M.J.Zaki, Arithmetic and Logic Operations with DNA. In:*Proc of 3rd DIMACS Workshop on DNA Based Computers*, U Penn 1997, pp 212-220.

[15] M.Ogihara and A.Ray, Simulating Boolean Circuits on a DNA Computer. **Tech Report TR631**, Department of C.Sc., University of Rochester, Aug 1996.

[16] Z.F.Qiu and M.Lu, Arithmetic and Logic Operations with DNA Computers, *Proc of 2nd IASTED International Conference on Parallel and Distributed Computing and Networks*, Brisbane, 1998, pp 481-486.

[17] D.R.Stinson, *Crypytography: Theory and Practice*, CRC Press, Boca Raton, 1995.

[18] M.Yamamoto, J.Yamashita, T.Shiba, T.Hirayama, S.Takiya, K.Suzuki, M.Munekata and A.Ohuchi, A Study on the Hybridization Process in DNA Computing, In: *Proc of the 5th DIMACS Workshop on DNA Based Computers*, MIT, Cambridge, 1999, pp101-110.