The Design of Sorters Based on DNA for Bio-Computers [*]

Hung-Yuan Wang, Chang-Biau Yang, and Kuo-Si Huang
Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan
cbyang@cse.nsysu.edu.tw

Yow-Ling Shiue
Institute of Biomedical Sciences,
National Sun Yat-sen University, Kaohsiung, Taiwan

Abstract

In the past few years, several articles have been devoted to the study of molecular computing based on DNA in order to implement algorithms for solving some NP-complete problems and simulate logic gates in silicon-based computers. A great deal of effort has been made on using DNA to implement simple logic gates, such as simple 1-bit comparators and simple adders, or to solve NP-complete problems, such as the Hamiltonian path problem, the travelling salesperson problem and the satisfiability problem. All of the methods rely on the capability of DNA computing which could perform computation in huge parallelism to produce all possible solutions where the answer may be derived from. In this paper, we will first design a full bit-serial comparator that can perform the feedback operation. Then, we will design a word-parallel bit-serial sorter which uses our comparators as the elementary building components. Our design of sorters can be applied to any sorting network, such as bitonic sorter and odd-even merge sorter.

Keywords: molecular computing, DNA computing, enzyme, sorter, comparator

1 Introduction

In the past few years, several articles have been devoted to the study of *molecular computing* based on DNA in order to implement algorithms for solving some NP-complete problems and simulate logic gates in silicon-based computers. [1, 7, 16] In molecular computing, the input data and output data encoded by deoxyribonucleic acid (DNA) sequences, and biochemical procedures in tubes are used as arithmetic or control operations. Such scheme is also called *bio-computing* or *DNA computing*.

There are four different kinds of nucleotides in DNA, which can be labelled as *A* (Adenine), *G* (Guanine), *C* (Cytosine) and *T* (Thymine), often called *bases*. Each data element encoded with DNA is described as a linear finite sequence composed of *A*, *G*, *C* and *T*. Consequently, it is true that we could use DNA to encode the string or pattern meaningful for us.

In addition to that DNA can be employed to encode the data, one of the capabilities of DNA is huge parallelism. We could think distinct DNA strands of different data in the presence of the biological processes of computational operations in a silicon-based computer. Distinct DNA strands in the same tube are affected simultaneously when we add some specially designated substances in the tube to carry out some biological processes. Consequently, one biological process could be carried out to affect all DNA strands at the same time. It corresponds to that a single instruction stream is executed by all processors to manipulate their local data synchronously. According to the architecture classification of Flynn, DNA computing can be classified into the class of *Single Instruction Multiple Data* (SIMD) [2]. Therefore, we could say DNA computing has the capability of parallel processing.

There are two main strategies in DNA computation. One is to implement the brute force algorithm for solving NP-complete problems which are still difficult to be solved in silicon-based computers [8, 9, 11, 12, 14, 18], and another is to simulate the function of logical gates or circuits prepared for bio-computers [3, 15, 17].

Adleman [1] used DNA computing to solve the Hamiltonian path problem with seven cities, and Lipton [12] used a similar method to solve the satisfiabil-

[*]This research work was partially supported by the National Science Council of the Republic of China under contract NSC-90-2213-E-110-015.

ity problem. These methods consist of two main steps. The first step is to produce all possible solutions, and then, the second step is to extract the real solutions out. To finish the first step, something has to be done in the beginning. We have to define the DNA data form of a given question for representing the relationship between the input data. Since the DNA data form is defined and designed, we need only pour the prepared DNA data in a test tube, and all possible solutions will be produced. Afterwards, take them to pass through the several different extracting steps. If any DNA remains, we can say that the answer exists.

For implementing a bio-computer, there are several articles have been devoted to simulate simple logic gates or circuits of a silicon-based computers, such as NAND gates, 1-bit comparators [3] and adders [6]. The methods of simulating hardware rely on the capability of parallel computation [14, 15, 17]. These logical gates or circuits on the same level will be put in the same test tube, so that all bio-operations can make all components process their local data in the same test tube.

For decreasing the inaccuracy rate of DNA strand reaction, solid phase experiments were adopted extensively in the later studies. In a solid phase experiment, the DNA strands are attached to the surface of a test tube to avoid the distinct data to influence each other, while previous studies adopted liquid phase experiments that all DNA data float around the liquid. Morimoto et al. [13] successfully used the liquid phase experiment to propose a method for solving the Hamiltonian path problem. Amos [3] also finished NAND gates in DNA form with the solid phase experiment. This paper also adopts solid phase of experiments to design our comparators, which are basic components for building a sorter. Here, a sorter is a circuit network which can arrange a set of input data into increasing, or decreasing, order.

This paper is organized as follows. In Section 2, we will introduce basic biological experiments for DNA computing. In Sections 3 and 4, we will illustrate how to design a bit-serial comparator and how to encode the data form with DNA. In Sections 5 and 6, we will execute the comparator and integrate the comparators into a sorter based on the sorting network of *bitonic sort*. Section 7 gives the conclusion of the paper.

2 Basic Operations of DNA Computing

To design a sorter with DNA, we may need some biological operations as follows.

1. Hybridization and annealing – Make two DNA strands form a double strand if they are complementary. Let S be a DNA strand, then the complementary strand of S is denoted as $C(S)$.

2. Synthesis and PCR – *Synthesis* can synthesize the desired DNA strand with any base sequences and easily amplify them in a great quantity by polymerase chain reaction (PCR) technology.

3. Restriction enzymes digestion – One double-stranded DNA can be cut into two parts by *restriction enzymes* if the double-stranded DNA contains a *restriction site*, which is a special DNA sequence. *Restriction enzymes* are special proteins which can attack *restriction sites* and cut them at some specific positions.

4. Ligation – *Ligase* is an enzyme that can repair or seal *nicks* in a DNA sequence with duplex structure [10]. Notice that in a double-stranded DNA sequence, the *nicks* do not result from the lack of any nucleotides in a continuous DNA sequence, but rather result from two neighboring nucleotides that do not join together.

5. Primer extension – Make two imperfect double strands to be a complete double strand, as shown in Figure 1.

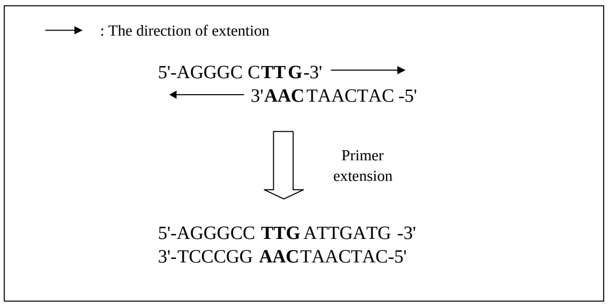


Figure 1: Primer extension.

6. Gel electrophoresis – Distribute all DNA strands into different groups according to their lengths or molecular weights.

7. Methyled DNA – It is a normal DNA (strand) attached by methyl-group [5] via the process of *DNA methylation*. Hybridization and annealing of methyled DNA are still normal. However, most methyled DNA could not be recognized by restriction enzymes any more with only some exceptions. Here, we use $M(s)$ to denote that DNA strand s is attached by methyl-group.

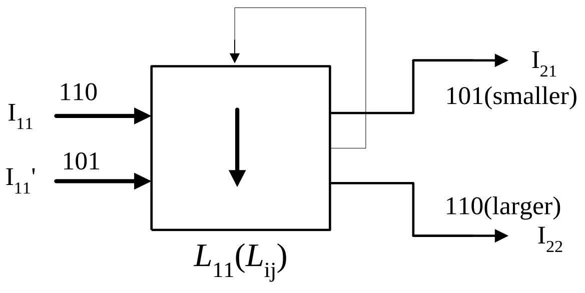
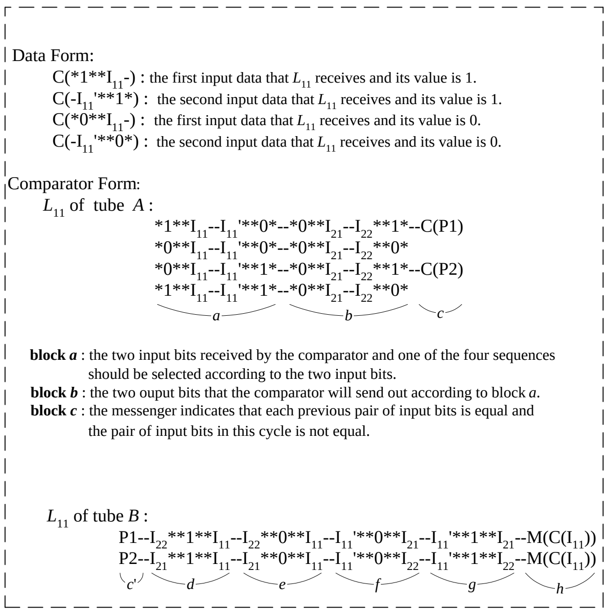


Figure 2: A comparator with bit pipeline.

8. **Affinity purification –** The process can help us to pick up the desired single strands with a specific sequence tag from a test tube [9].

9. **Graduated PCR (GPCR) –** This is one of the methods which can read out the single DNA strands [6, 13].



Data Form:

C(*1**I_{11}-) : the first input data that L_{11} receives and its value is 1.
C(-I_{11}'**1*) : the second input data that L_{11} receives and its value is 1.
C(*0**I_{11}-) : the first input data that L_{11} receives and its value is 0.
C(-I_{11}'**0*) : the second input data that L_{11} receives and its value is 0.

Comparator Form:

L_{11} of tube A :

*1**I_{11}--I_{11}'**0*--*0**I_{21}--I_{22}**1*--C(P1)
*0**I_{11}--I_{11}'**0*--*0**I_{21}--I_{22}**0*
*0**I_{11}--I_{11}'**1*--*0**I_{21}--I_{22}**1*--C(P2)
*1**I_{11}--I_{11}'**1*--*0**I_{21}--I_{22}**0*

block *a* : the two input bits received by the comparator and one of the four sequences should be selected according to the two input bits.
block *b* : the two ouput bits that the comparator will send out according to block *a*.
block *c* : the messenger indicates that each previous pair of input bits is equal and the pair of input bits in this cycle is not equal.

L_{11} of tube B :

P1--I_{22}**1**I_{11}--I_{22}**0**I_{11}--I_{11}'**0**I_{21}--I_{11}'**1**I_{21}--M(C(I_{11}))
P2--I_{21}**1**I_{11}--I_{21}**0**I_{11}--I_{11}'**0**I_{22}--I_{11}'**1**I_{22}--M(C(I_{11}))

Figure 3: The design of a comparator.

3 A Bit-Serial Comparator

A sorter is a circuit network used to solve the sorting problem. The kernel component of a sorter is a comparator, which can compare two integers. Here, we design a bit-serial comparator, which has two input lines and two output lines. As shown in Figure 2, the smaller number gets to the top output line, and the larger one gets to the bottom output line when the arrow in the box is in the down direction, which means increasing order. We can use a sorting network consisting of many comparators to sort all given numbers. In other words, the comparators arranged in a particular network can achieve the purpose of sorting.

Our bit-serial comparator receives two input bits serially from higher bits to lower bits and the two output bits give the answer serially, too. The comparison task can be done according to the following two rules when a pair of bits reach:

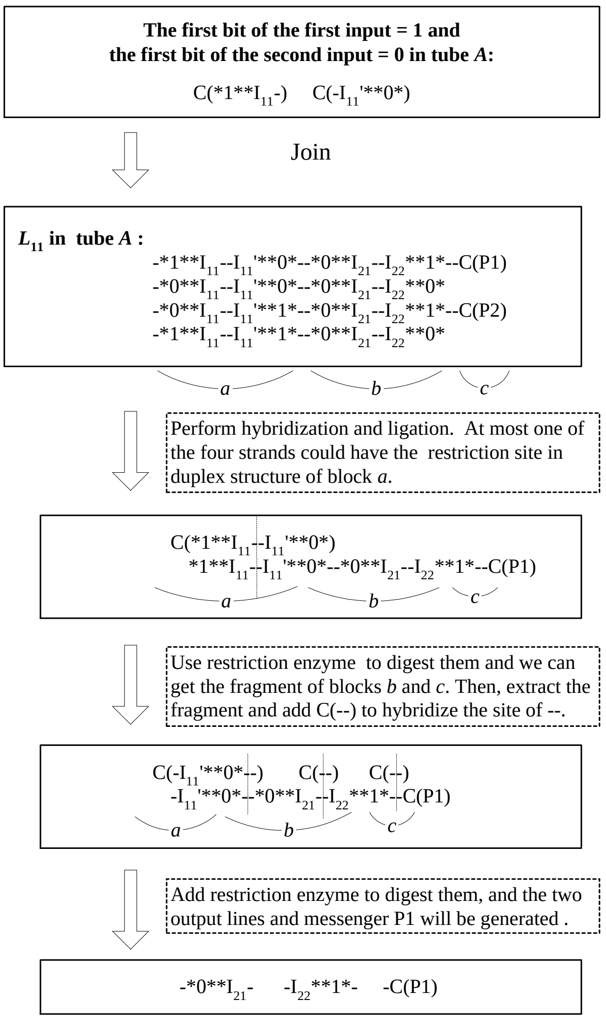
Rule I. If the pair of bits is equal, the output lines remain unchanged. Because the bits are equal, the bits are directly sent out to output lines. The most important is that the next pair of bits received will be compared with these two rules again, except that the winner and loser have been decided by the previous (more significant) bits.

Rule II. If they are not equal, bit containing 0 (loser) is sent out to the top output line. And for the comparison in each of the following cycles, each subsequent bit of the loser will be sent to the top output line. Naturally, the bit containing 1 (winner) is sent out to the bottom output line. Once Rule II is applied, Rule I would not be used any more.

4 The Encoding of the Comparator

We define the DNA data form of 1-bit input as $C(-I'_{ij} **V*)$ or $C(-I_{ij} **V*)$, which means the input given to the jth comparator on the ith level, and V represents the value of either 0 or 1, which is the input to the comparator, as shown in Figure 3. The mark $'$ indicates one of the inputs in the gate, and another input is without $'$. The symbol ** is a special DNA sequence that is one kind of restriction sites. Here, the symbol ** represents the restriction site consisting of *CCGG*. The left * is *CC* while the right * is *GG*, and the symbol -- represents another restriction site in Figure 3.

We now define the DNA strands prepared in tube *A*. Since we adopt the *solid phase* experiment to design our sorter, the DNA strands is stuck on the surface of tube *A*. As shown in Figure 3, the DNA strands are divided into three blocks. Block *a* represents the complementary strands of input for matching the two input bits and receives them to anneal. That is, the two input bits will choose one of the four DNA strands which can perfectly match the two input bits for hybridization. This is because that only one DNA strand in tube *A* satisfies the Waston-Crick rule. Here, block *a* of the comparator represents the receiver for the DNA data form and it conforms to Waston-Crick complementation to be able to form double strands. Figure 4 is an


The first bit of the first input = 1 and the first bit of the second input = 0 in tube A:
C(*1**I11-) C(-I11'**0*)
Join
L11 in tube A :
-*1**I11--I11'**0*--*0**I21--I22**1*--C(P1)
-*0**I11--I11'**0*--*0**I21--I22**0*
-*0**I11--I11'**1*--*0**I21--I22**1*--C(P2)
-*1**I11--I11'**1*--*0**I21--I22**0*
a
b
c
Perform hybridization and ligation. At most one of the four strands could have the restriction site in duplex structure of block a.
C(*1**I11--I11'**0*)
*1**I11--I11'**0*--*0**I21--I22**1*--C(P1)
a
b
c
Use restriction enzyme to digest them and we can get the fragment of blocks b and c. Then, extract the fragment and add C(--) to hybridize the site of --.
C(-I11'**0*--) C(--) C(--)
-I11'**0*--*0**I21--I22**1*--C(P1)
a
b
c
Add restriction enzyme to digest them, and the two output lines and messenger P1 will be generated .
-*0**I21- -I22**1*- -C(P1)

Two output streams appear successfully and the messenger P1 occurs.

Figure 4: The reaction without feedback in tube A for comparing the two input bits which are not equal.

example to illustrate the reaction in tube A where the first input bit is 1 and the second input bit is 0.

In block b, there are also two data forms which link each other by some restriction site. While block a represents the two input bits of the comparator, block b represents the two output bits. It is natural that at most two output bits will be produced since at most one DNA sequence in tube A will be selected, as shown in Figure 4. Therefore, we can get the output bits from block b. Note that the feedback operation does not have effect yet here.

The messenger, encoded by block c, is very important for us to perform the feedback to affect the input of next cycle in the comparator. Block c is corresponding to the block c' of tube B and they are complementary.

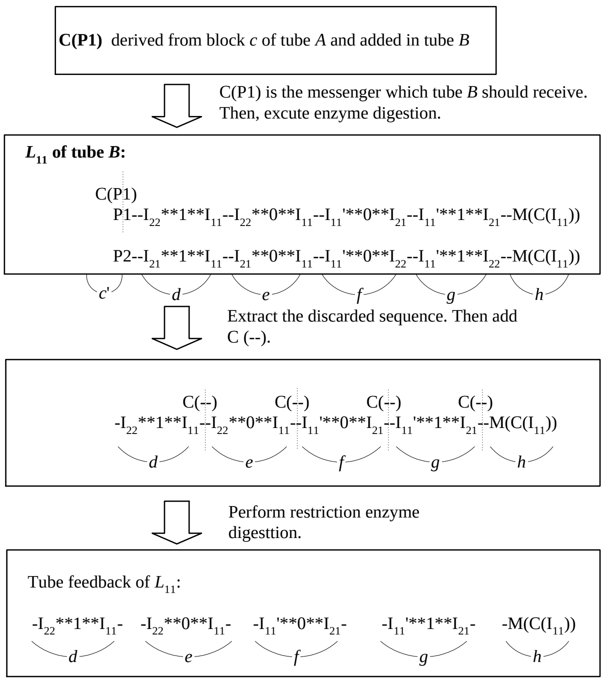

C(P1) derived from block c of tube A and added in tube B
C(P1) is the messenger which tube B should receive. Then, excute enzyme digestion.
L11 of tube B:
C(P1)
P1--I22**1**I11--I22**0**I11--I11'**0**I21--I11'**1**I21--M(C(I11))
P2--I21**1**I11--I21**0**I11--I11'**0**I22--I11'**1**I22--M(C(I11))
c'
d
e
f
g
h
Extract the discarded sequence. Then add C (--).
C(--) C(--) C(--) C(--)
-I22**1**I11--I22**0**I11--I11'**0**I21--I11'**1**I21--M(C(I11))
d
e
f
g
h
Perform restriction enzyme digesttion.
Tube feedback of L11:
-I22**1**I11- -I22**0**I11- -I11'**0**I21- -I11'**1**I21- -M(C(I11))
d
e
f
g
h

Figure 5: The process of feedback.

The designs of blocks c and c' satisfy two conditions. One is that the pair is the only one in both tubes A and B simultaneously. And the other is that the pair should have a restriction site so that they can be cut off by some restriction enzyme, as shown in Figure 5. Here, we use the third restriction site. The way is feasible that block c is composed of the third kind of restriction site and I_{ij} or I'_{ij} could imply who is the winner in the two input. For example, $C(P1)$ contains I_{11} and $C(P2)$ contains I'_{11}. In Figure 3, block c will be produced only in the situation that each previous (more significant) pair of input bits is equal and the pair of input bits is not equal in this cycle.

Tube B performs the task for the important role of implementing the function of feedback. Because $C(P1)$ and $C(P2)$ imply that the winner of these two compared words has bee decided, all the subsequent bits of the two inputs need not be compared again. They can be sent directly to the target comparator of the next level. The products, blocks d, e, f and g in Figure 5, play the role of switchmen for this work. Blocks d and e can accept the value 0 or 1 from I_{11} and transfer them to the data form, which will be accepted by comparator I_{22}. Blocks e and f play the same role except that they transfer the data form from I'_{11} to the data form for comparator I_{21}. Therefore, the comparator on the next level where the output should go is also decided. In other words, a switchman provides a guide of data flow paths. Of course, the four blocks should

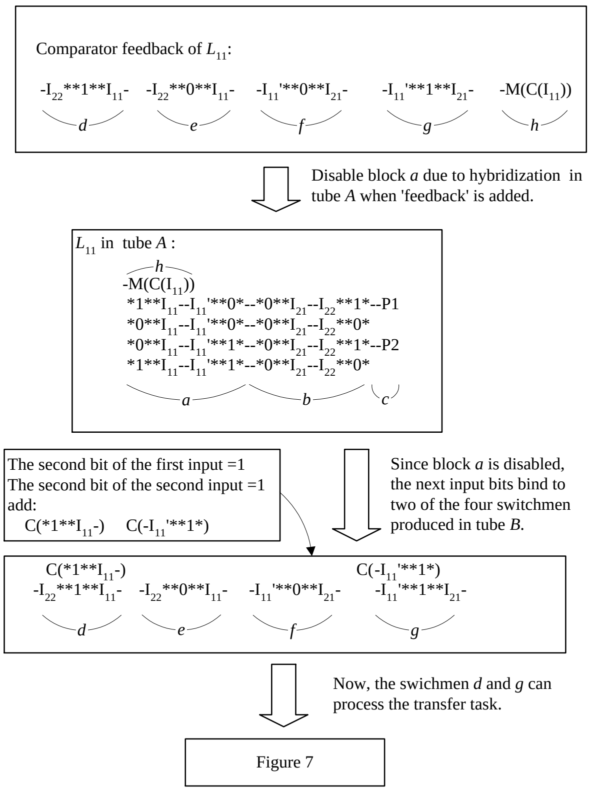

Comparator feedback of L_{11}:
$-I_{22}**1**I_{11}-$ $-I_{22}**0**I_{11}-$ $-I_{11}'**0**I_{21}-$ $-I_{11}'**1**I_{21}-$ $-M(C(I_{11}))$
d e f g h
Disable block a due to hybridization in tube A when 'feedback' is added.
L_{11} in tube A :
h
$-M(C(I_{11}))$
$*1**I_{11}--I_{11}'**0*--*0**I_{21}--I_{22}**1*--P1$
$*0**I_{11}--I_{11}'**0*--*0**I_{21}--I_{22}**0*$
$*0**I_{11}--I_{11}'**1*--*0**I_{21}--I_{22}**1*--P2$
$*1**I_{11}--I_{11}'**1*--*0**I_{21}--I_{22}**0*$
a b c
The second bit of the first input =1
The second bit of the second input =1
add:
$C(*1**I_{11}-)$ $C(-I_{11}'**1*)$
Since block a is disabled, the next input bits bind to two of the four switchmen produced in tube B.
$C(*1**I_{11}-)$ $C(-I_{11}'**1*)$
$-I_{22}**1**I_{11}-$ $-I_{22}**0**I_{11}-$ $-I_{11}'**0**I_{21}-$ $-I_{11}'**1**I_{21}-$
d e f g
Now, the swichmen d and g can process the transfer task.
Figure 7

Figure 6: The combination of next input bits with switchmen.

be separated first by the restriction enzyme.

In fact, block *h* is a very important assistant to implement the work of 'switch', when the comparator starts the steps to switch the value of the input bits to the next comparator. It is essential to raise some behavior that the input could not act on the prepared strands to be processed in tube *A*, as shown in Figure 4. And they should help them to act on the four blocks as well as switchmen added from tube *B* as shown in Figure 6. It means when the gain or loss has been determined, the input bits do not need to pass through comparator again and the comparator should be switched directly according to where the inputs come from. In respect of the behavior, block *h* could be considered as the role of the 'blocker' induced by the messenger, such as block *c* of tube *A* in Figure 5. Block *h* blocks the action of the comparison of subsequent inputs, as shown in Figure 6, and inputs will anneal to the 'switchman' instead of annealing to block *a*. The tasks of switchmen are shown in Figure 7.

How do the switchmen perform their tasks? It is easy to know that the four blocks have the potential to do the task of 'switch', when we take their design into account. The design consists of three parts, where the data come from, where the data should go and what

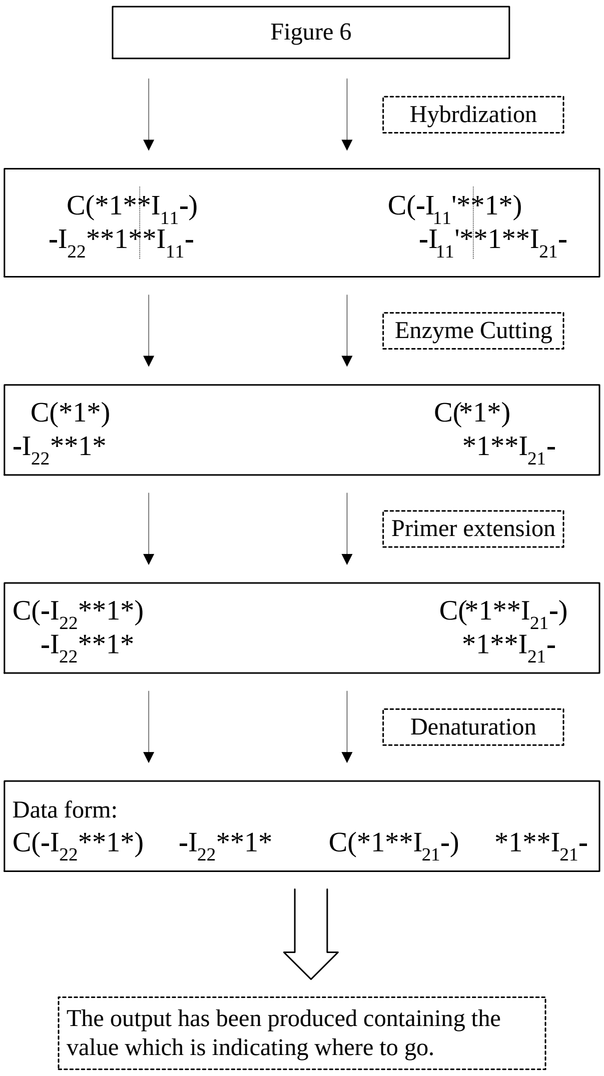


Figure 6
Hybrdization
$C(*1**I_{11}-)$ $-I_{22}**1**I_{11}-$
$C(-I_{11}'**1*)$ $-I_{11}'**1**I_{21}-$
Enzyme Cutting
$C(*1*)$ $-I_{22}**1*$
$C(*1*)$ $*1**I_{21}-$
Primer extension
$C(-I_{22}**1*)$ $-I_{22}**1*$
$C(*1**I_{21}-)$ $*1**I_{21}-$
Denaturation
Data form:
$C(-I_{22}**1*)$ $-I_{22}**1*$ $C(*1**I_{21}-)$ $*1**I_{21}-$
The output has been produced containing the value which is indicating where to go.

Figure 7: The work of switchmen.

value the data element contains. As shown in Figure 7, we could use the skill to assign the value to another comparator by generating new data for the output.

5 A Biological Sorter

Our methods of comparator design, as shown before, are enough to be adopted to build our sorters, which is a word-parallel bit-serial sorting network. Now, in order to summarize how the sorter works, we assume that m integers of n bits are given as the input. We would arrange them in increasing order. The comparators of the same level are prepared in the same tube:

Input data preparation: Prepare all bits of the integers according to our data form, and put them in different tubes by the bit-position. The highest bits are put in the test tube labeled 0, the next highest bits are put in the tube labeled 1, and so on, then the lowest bits are put in the test tube labeled $(n-1)$. We therefore will pour the input data from tube 0 to tube $(n-1)$ serially. Every tube contains m data elements.

The sorter preparation: The comparators on the first level of the sorting network are prepared corresponding to the input, the comparators on the second level are prepared corresponding to the output of comparators on the previous level.

The following gives the operations in one cycle for all comparators to perform the comparisons of data bits on their level.

Step 1: Put the results of tube C in tube A if the comparison in this cycle is not for the most significant bits.

Step 2: Put the input, which are the data bits that should be compared on the level, in tube A, as shown in Figure 4.

Step 3: Process the task of feedback, as shown in Figure 6 and Figure 7.

Step 4: Extract the output from the results of Step 2 and Step 3. The output will become the input of comparators on the next level.

Step 5: Extract the messengers from the results of Step 2 and Step 3, and put them in tube B for feedback, as shown in Figure 5.

Step 6: Reserve the results of Step 5 in tube C.

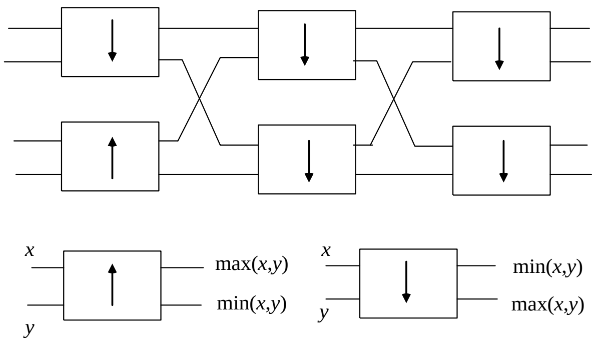


Figure 8: The network of a bitonic sorter, which can sort four elements.

Repeat the cycle n times and then all comparators on their level will finish their work. Note that the comparators on different levels in the sorter are pipelined. As soon as the comparators on the last level have finished, the sorter has finished. That is, the sorting is complete. At the moment, the output has been attached, in increasing order, to the surface of the result tube. The final work is to take the output and read it out.

6 Implementation of the Bitonic Sorters

It is known that a sorter is composed of comparators. Since we finish the design of a biological comparator and the definition of the DNA data form, we can arrange the comparators to form a bitonic sorter [4]. Figure 8 shows an example of the network of a bitonic sorter which can sort four elements. It is noticed that the layout of the lines between the comparators are fixed for the sorter. Thus, the links among our biological comparators have to be arranged first. In the design of a sorter, as shown in Figure 9, block b in tube A and the switchmen of tube B indicate that where the output shall go. Block a in tube A indicates which data the comparator should receive. Therefore, our comparators can be arranged in various sorting networks, such as bitonic sort and odd-even merge sort .

In Figure 9, the function of tube A is to receive the feedback from tube C, to receive the input data and to reserve the result of the reaction in tube A, and then to pass the result to tube B. The feedback would affect the reaction in tube A when the content of tube C is not empty. The result contains the output and the message which indicates if the comparator starts the feedback in next cycle. The function of tube B is to send the output of the comparator, and to produce the switchman for performing the feedback which influences the next input if tube B receives the message for starting feed-

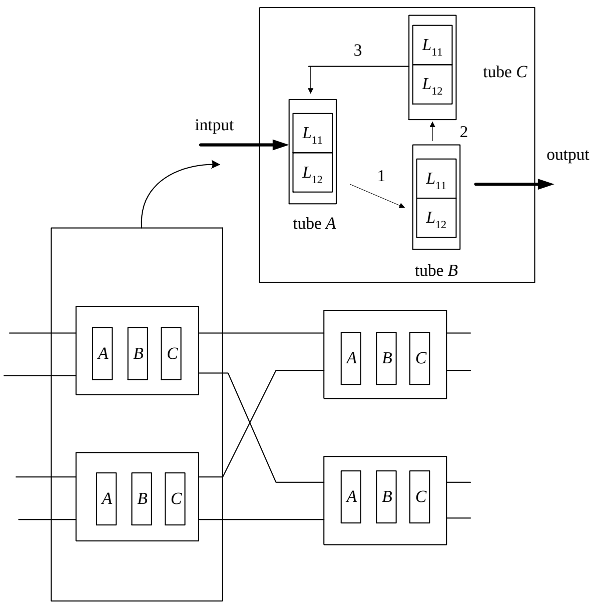


Figure 9: The design of the bitonic sorter based on bit-serial comparators. Note that the comparators on the same level are prepared in the same tubes (including three tubes).

back. Tube C is the tube containing the contents of feedback which keeps the switchman and the blocker which will be put in tube A.

Repeat the cycle as mentioned in Section 5 until all bits have been compared in all comparators on all levels. In the sorting, we would know the output is derived from those comparators on the last level. Consequently, the result of the sorting can be obtained.

7 Conclusion

Implementation of hardware components, such as adders or comparators, and computational functions, such as the solutions for the Hamiltonian path problem, the satisfiability problem or the traveling salesperson problem, based on DNA sequences or molecular sequences becomes a new research direction recently. In this paper, our goal is to design an efficient sorter. We first design a full bit-serial comparator which can perform the feedback operation. Then, with the comparators, we can build a sorter, which can be applied to any sorting network, such as bitonic sort or odd-even merge sort.

Using molecular computing or DNA computing, two issues need to be considered. The first one is how to reduce the inaccuracy of the reaction in DNA strands, which may occur on similar DNA sequences. It may be improved by properly designing DNA encoding schemes to amplify the difference among DNA strands. Another is how to reduce the required time on real biological experiments. The degree for overcoming these two difficulties actually depends on the improvement of the biochemical technology.

We believe that there is still some room for DNA computing to solve more difficult problems with some other methods, for example, the Ant System. In the future, the idea of designing solutions or algorithms based on DNA computing should be applied to more complex systems, such as a complete bio-computer.

References

[1] L. M. Adleman, “Molecular computation of solutions to combinatorial problems,” *Science*, Vol. 266, pp. 1021–1024, Nov. 1994.

[2] S. G. Akl, *The Design and Analysis of Parallel Algorithms*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, first ed., 1989.

[3] M. Amos, P. E. Dunne, and A. Gibbons, “DNA simulation of boolean circuits,” *Genetic Programming 1998: Proceedings of the Third Annual Conference, University of Wisconsin, Madison, Wisconsin*, pp. 679–683, 1998.

[4] K. E. Batcher, “Sorting networks and their applications,” *In Proceedings of the AFIPS Spring Joint Computer Conference, Atlantic City, NJ, USA*, Vol. 32, pp. 307–314, 1968.

[5] R. H. Garrett and C. M. Grisham, *Biochemistry*. Emily Barrosse, John J. Vondeling, second ed., 1998.

[6] F. Guarnieri and M. Fliss, “Making DNA add,” *Science*, Vol. 273, pp. 220–223, July 1996.

[7] M. Hagiya, “From molecular computing to molecular programming,” *Proceedings 6th DIMACS Workshop on DNA Based Computers, held at the University of Leiden*, p. 87, June 2000.

[8] T. Hinze and M. Sturm, “A universal functional approach to DNA computing and its experimental practicability,” *Proceedings 6th DIMACS Workshop on DNA Based Computers, held at the University of Leiden, Leiden, The Netherlands, 13 - 17 June 2000*, p. 257, 2000.

[9] J.-S. Hwu and R.-J. Chen, “DNA solution to the traveling salesman optimization problem,” *1998 International Computer Symposium Workshop on Algorithms, Tainan, Taiwan*, pp. 17–19, dec. 1998.

[10] Jonoska and Karl, “Ligation experiments in computing with DNA,” *Proceedings of 1997 IEEE International Conference on Evolutionary Computation, University Place Hotel Indianapolis*, pp. 261–266, 1997.

[11] S. Y. Lila Kari, Greg Gloor, “Using DNA to solve the bounded post correspondence problem,” *Theoretical Computer Science*, Vol. 231, pp. 193–203, 2000.

[12] R. J. Lipton, “Using DNA to solve NP-complete problems,” *Science*, Vol. 268, pp. 542–545, Apr. 1995.

[13] N. Morimoto, M. Arita, and A. Suyama, “Solid phase DNA solution to the Hamiltonian path problem,” *Proceedings of the 3rd DIMACS Workshop on DNA Based Computers, University of Pennsylvania, U.S.A*, pp. 83–92, 1997.

[14] E. Ogihara and A. Ray, “Executing parallel logical operations with DNA,” *Proceedings of the Congress on Evolutionary Computation, Washington, DC*, Vol. 2, pp. 972–979, June 1999.

[15] M. Ogihara and A. Ray, “DNA-based self-propagating algorithm for solving bounded-fan-in boolean circuits,” *Genetic Programming 1998: Proceedings of the Third Annual Conference, University of Wisconsin, Madison, Wisconsin*, pp. 725–730, April 1998.

[16] S.-Y. Shin, B.-T. Zhang, and S.-S. Jun, “Solving traveling salesman problems using molecular programming,” *Proceedings of the Congress on Evolutionary Computation, Washington, DC*, Vol. 2, pp. 994–1000, 1999.

[17] S. P. V. Gupta and M. J. Zaki, “Arithmetic and logic operations with DNA,” *Proceedings of the 3rd DIMACS Workshop on DNA Based Computers, held at the University of Pennsylvania*, pp. 212–220, June 1997.

[18] H. Wu, “An improved surface-based method for DNA computation,” *BioSystems*, Vol. 59, pp. 1–5, 2001.