Available online at www.sciencedirect.com
ScienceDirect
Progress in Natural Science 19 (2009) 377–388
Progress in Natural Science
www.elsevier.com/locate/pnsc

Arithmetic computation using self-assembly of DNA tiles: subtraction and division

Xuncai Zhang [a,b], Yanfeng Wang [b], Zhihua Chen [a], Jin Xu [a,*], Guangzhao Cui [b]

[a] The Key Laboratory of Image Processing and Intelligent Control, Department of Control Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, China
[b] School of Electrical and Electronic Engineering, Zhengzhou University of Light Industry, Zhengzhou 450002, China

Received 9 April 2008; received in revised form 18 July 2008; accepted 21 July 2008

Abstract

Recently, experiments have demonstrated that simple binary arithmetic and logical operations can be computed by the process of self-assembly of DNA tiles. In this paper, we show how the tile assembly process can be used for subtraction and division. In order to achieve this aim, four systems, including the comparator system, the duplicator system, the subtraction system, and the division system, are proposed to compute the difference and quotient of two input numbers using the tile assembly model. This work indicates that these systems can be carried out in polynomial time with optimal $O(1)$ distinct tile types in parallel and at very low cost. Furthermore, we provide a scheme to factor the product of two prime numbers, and it is a breakthrough in basic biological operations using a molecular computer by self-assembly.

Keywords: Self-assembly; Subtractor; Divider; Tile assembly model; DNA tiles; DNA computing

1. Introduction

Since the seminal work of Adleman on the Hamiltonian path problem [1], the field of DNA-based computing has experienced a flowering growth and leaves us with a rich legacy. Given its vast parallelism and high-density storage, DNA computing approaches have been employed to solve many combinatorial optimization problems. However, most of these proposals implement the computation by performing a series of biochemical reactions on a set of DNA molecules, which requires human intervention at each step. Thus, each of the steps require a large amount of time compared with the computation steps of an electronic computer.

The self-assembly process, defined as the autonomous organization of components into structurally well-defined aggregates, is characterized by numerous beneficial attributes. It is cost effective, versatile, facile, and the process occurs towards the system's thermodynamic minima, resulting in stable and robust structures [2]. There are numerous different mechanisms by which self-assembly of molecules and nanoclusters can be accomplished, such as electrostatic and surface forces, chemical interactions, hydrophobic and hydrophilic interactions, and biomolecule-mediated self-assembly techniques. Simple self-assembly schemes have been already widely used in chemical syntheses, and it has been suggested that more complicated schemes will ultimately be useful for circuit fabrication, nanorobotics, and amorphous computing [3,4]. Winfree et al. first proposed the idea of computation by self-assembled tiles. Because DNA tiles can be more easily "programmed" to incorporate the constraints of a given

* Corresponding author. Tel.: +86 27 87543563; fax: +86 27 87543130.
E-mail address: jxu@mail.hust.edu.cn (J. Xu).

1002-0071/$ - see front matter © 2008 National Natural Science Foundation of China and Chinese Academy of Sciences. Published by Elsevier Limited and Science in China Press. All rights reserved.
doi:10.1016/j.pnsc.2008.07.013

problem, it is possible to exercise some degree of control in the test tubes, thus avoiding the considerable waste of materials that characterize the traditional approach, and therefore parallel computation can be enhanced by the self-assembling process where information is encoded in DNA tiles. The relation of DNA computation to self-assembling structures was developed in the mid 1990s through the large theoretical and experimental work of Winfree et al. [5,6], Seeman [7], Reif [8], and Rozenberg and Spaink [9]. Currently, computational systems based on self-assembly were demonstrated in both 1-D arrangements called "string tiles" [5,10] and 2-D lattices of DNA [6]. Other stable forms of nucleic acids include Z-DNA, non-migrating Holliday junctions, and duplexes with triple crossovers or "pseudoknots" [7,11,12]. For 2-D self-assembly, Winfree proposed the tile assembly model [6], and demonstrated that it is Turing universal by showing that a tile system can simulate Wang tiles [13], which Robinson has shown to be universal [14]. The tile assembly model is a formal model for the self-assembly of molecules that are constrained to self-assemble on a square lattice. It is an extension of the theory of tiling by Wang tiles [13] to include a specific mechanism for growth based on the physics of molecular self-assembly.

The application and adaptation of those results using DNA have been pursued by various groups. The first experimental demonstrations of computation using DNA tile assembly were in Ref. [12], which gave a two-layer, linear assembly of TX tiles that executed a bit-wise cumulative XOR computation. Barish et al. [15] provided a DNA implementation of a tile system that copies an input and counts in binary. Cook et al. [16] used the tile assembly model to implement arbitrary circuits. Similarly, Rothemund et al. [17] demonstrated DNA implementation of a tile system that computes the XOR function, resulting in a Sierpinski triangle. Brun [18] proposed and theoretically studied the systems that compute the sums and products of two numbers using the tile assembly model. He found that in the tile assembly model, adding and multiplying can be done using $O(1)$ tiles, and that both computations can be carried out in a linear time. Then, he combined these systems to create two new systems with more complex behavior for factoring integer numbers and for deciding subset sum problems [19,20].

Here, we propose two systems to compute the difference and quotient of two input numbers using the tile assembly model with $O(1)$ tiles, and these systems can be carried out in polynomial time. Furthermore, we provide a scheme to factor the product of two prime numbers, and the design protocols to execute integer factoring are also described.

2. DNA tiles and self-assembly

DNA nanostructures provide a programmable methodology for bottom-up nano-scale construction of patterned structures, utilizing macromolecular building blocks called DNA tiles based on branched DNA. These tiles have sticky ends that match the sticky ends of other DNA tiles, facilitating further assembly into larger structures known as DNA tiling lattices.

2.1. Physical implementation of Wang tiles

Computation and self-assembly are connected by the theory of tiling, of which Wang tiles [13] are a prime example. The theory of Wang tiles showed that square tiles with colored edges can emulate a Turing machine, if they are allowed to assemble in a way that would cover the plane according to the additional rule of the same color edges having to face each other. Branched DNA molecules provide a direct physical motivation for the tile assembly model. There is also a logical equivalence between DNA sticky ends and Wang tile edges. Winfree and Seeman [10] introduced DNA tiles to demonstrate the feasibility of computing through the self-assembly of DNA tiles. These DNA tiles have unpaired ends of DNA strands sticking out, and through these "sticky" ends they can attach themselves with other tiles having the Watson–Crick complementary sticky ends. Thus, these tiles can stick with one another to assemble into complex superstructures, and through this process they can compute in a way similar to Wang tiles. A variety of different DNA tile types had been used in previous assemblies, including DX molecules [5], TX molecules [21], and parallelograms produced from Holliday junction analogues [11]. Fig. 1 gives some structures of DNA tiles.

2.2. Molecular self-assembly processes

Molecular self-assembly is the ubiquitous process by which simple objects autonomously assemble into intricate complexes. It has been suggested that intricate self-assembly processes will ultimately be used in circuit fabrication, nanorobotics, DNA computing, and amorphous computing [22].

There are three basic steps that define a process of molecular self-assembly [23]: ① *Molecular recognition*, elementary molecules selectively bind to others. ② *Growth*, elementary molecules or intermediate assemblies are the building blocks that bind to each other following a sequential or hierarchical assembly. Cooperativity and non-linear behavior often characterize this process. ③ *Termination*, a built-in halting feature is required to specify the completion of the assembly. In practice, their growth is interrupted by physical and/or environmental constraints. Molecular self-assembly has the following three characteristics [23]: ① Molecular self-assembly is a *time-dependent process* and because of this, temporal information and kinetic control may play a role in the process before thermodynamic stability is reached. ② Molecular self-assembly is also a *highly parallel process*, where many copies of different molecules bind simultaneously to form intermediate complexes. ③ Another characteristic of a molecular self-assembly is that the *hierarchical build-up* of complex assemblies allows one

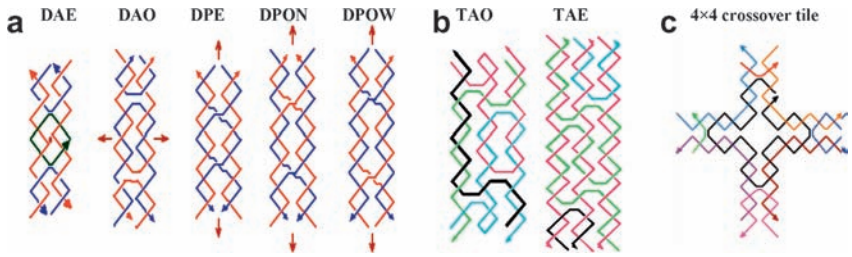

Fig. 1. DNA tiles. (a) The DX molecules; (b) the TX molecules; (c) the other DNA tile known as the 4×4 crossover tile.

to intervene at each step, either to suppress the following one, or to orient the system towards a different pathway.

2.3. Models for algorithmic self-assembly

The abstract tile assembly model, which provides a rigorous framework for analyzing algorithmic self-assembly, was originally proposed by Rothemund and Winfree [24]. It extends the theoretical model of tiling by Wang [13] to include a mechanism for growth based on the physics of molecular self-assembly. This model considers the assembly of rigid square objects or tiles.

Formally, a *tile* over a set of *binding domains* Σ is a 4-tuple $\{\sigma_N, \sigma_E, \sigma_S, \sigma_W\} \in \Sigma^4$, indicating the binding domains on the north, east, south and west sides. A *position* is an element of $\mathbb{Z}^2$. The set of directions $D = \{N, E, S, W\}$ is a set of four functions from positions to positions, i.e. $\mathbb{Z}^2$ to $\mathbb{Z}^2$ such that for all positions (x, y), $N(x, y) = (x, y + 1)$, $E(x, y) = (x + 1, y)$, $S(x, y) = (x, y - 1)$, and $W(x, y) = (x - 1, y)$. The positions (x, y) and (x', y') are neighbors if $\exists d \in D$ such that $d(x, y) = (x', y')$. For a tile t, for $d \in D$, we refer to $bd_d(t)$ as the binding domain of tile t on the d's side. According to this definition, tiles may not be rotated. A special tile $empty = \{null, null, null, null\}$ represents the absence of all other tiles.

The binding domains determine the interaction between tiles, that is, when two tiles may be placed next to each other. A function g: Σ^2 to $\mathbb{R}$, where $null \in \Sigma$, is a *strength function* that denotes the strength of the binding domains, which may be 0, 1, or 2 (called *null, weak*, and *strong* bonds, respectively). If $\forall \sigma, \sigma' \in \Sigma$, then $g(\sigma, \sigma') = g(\sigma', \sigma)$ and $g(null, \sigma) = 0$.

Let T be a finite set of tile types containing the *empty* tile. A *configuration* of T is a map from $\mathbb{Z}^2$ to T. Given a set of seed tile types Γ, a seed configuration of Γ is a map from $\mathbb{Z}^2$ to Γ. A tile system is a quadruple $\mathbb{S} = (T, S, g, \tau)$, where T, g, S are the same as those mentioned above, and $\tau \geqslant 0$ is the temperature.

If F is a configuration, then within system $\mathbb{S}$, a tile t can attach to F at position (x, y) and produce a new configuration F' if: ① $F(x, y) = empty$; ② $\Sigma_{d \in D} g(bd_d(t), bd_{d^{-1}}(F(d(x, y)))) \geqslant \tau$, where d^{-1} denotes the d's opposite direction, for example, N and S are opposite directions, E and W are opposite directions; ③ $\forall (u, v) \in \mathbb{Z}^2, (u, v) \neq (x, y) \Rightarrow F'(u, v) = F(u, v)$; ④ $F'(x, y) = t$. That is, a tile can attach to a configuration only in empty positions and if the total strength of the appropriate binding domains on the tiles in neighboring positions meets or exceeds the τ.

Given a tile system $\mathbb{S} = (T, S, g, \tau)$, if the above-mentioned conditions are satisfied, one may attach tiles of T to S. Configurations produced by repeated attachments of tiles from T are said to be produced by $\mathbb{S}$ on S. If this process terminates, then the configuration achieved when no more attachments are possible is called the final configuration. If for all sequences of tile attachments, all possible final configurations are identical, then $\mathbb{S}$ is said to produce a unique final configuration on S.

2.4. Programming self-assembly of DNA tilings

A tiling is an arrangement of a few tiles that fit together perfectly in the infinite plane. Programming DNA self-assembly of tilings amounts to the design of the pads (sticky ends) of DNA tiles, ensuring that only the adjacent pads of neighboring tiles are complementary, so that tiles assemble together as intended. The use of pads with complementary base sequences allows the neighbor relations of tiles in the final assembly to be intimately controlled; thus the only large-scale superstructures formed during assembly are those that encode valid mappings of input to output. The progress of self-assembly of DNA tilings can be carried out by the following four steps [23]: ① Mixing the input oligonucleotides to form the DNA tiles, ② allowing the tiles to self-assemble into superstructures, ③ ligating strands that have been co-localized, ④ performing a single separation to identify the correct output.

3. The construction of subtractor and divider

Subtraction and division are fairly basic operators for any computing model. Here, we will describe two tile systems: subtraction and division. As we know, a division operation is carried out by successive compare, shift, and subtract operations. This indicates that compare and shift operations must be finished before the corresponding subtraction operation is done. Therefore, we will first define two small systems that perform pieces of necessary computation building up division function. The first system, which is called the comparator system, compares the size of two input numbers (this system will be used before the corresponding subtraction operation is done in the divider tile system). The second system, which is called the duplica-

tor system, copies appointed information when the minuend is smaller compared with the subtrahend in the divider tile system. Then, a subtractor system will be described. Finally, we will add a few other tiles that ensure that computations go as planned, and combine these systems to create a new system that executes the division operation for two input numbers; the quotient and remainder will be obtained. All the systems use $O(1)$ distinct tiles. As for the proofs of the systems' correctness, we can refer to the unique final configuration corollary [18]. The corollary states that if all the tiles in a system have unique east-south or west-south binding domain pairs, then on some seed configurations, that system always produces a unique final configuration.

Suppose that an unsigned integer p is represented as a k-bit number, $p_{k-1}\ldots p_0$, where the value of each bit p_i is either one or zero for $0 \leqslant i \leqslant k-1$. The bits p_{k-1} and p_0 represent, respectively, the most significant bit and the least significant bit for p. Here, we will use four types of tiles to encode the two input numbers in all systems which will be described. Let the set of these tiles be $\Gamma = \{D_{00} = \langle 00, null, null, null\rangle, D_{01} = \langle 01, null, null, null\rangle, D_{10} = \langle 10, null, null, null\rangle, D_{11} = \langle 11, null, null, null\rangle\}$. A graphical representation of Γ is shown in Fig. 2, the value in the middle of each tile t represents that tile's $v(t)$ value. For these tiles with two-bit binding domains bd, let bd^L be the first bit and bd^R be the second bit of the binding domain. Let p and q be two k-bit input numbers, then for tile $t \in \Gamma$, the $bd_N^L(t)$ encoding the value of the bit p_i is stored in the tile, and the $bd_N^R(t)$ encoding the value of the bit q_i is stored along with each bit p_i. The configuration for two k-bit input numbers is shown in Fig. 2(b), where p_iq_i denotes the conjoint of bit p_i and q_i. Two special tiles called boundary tiles encoding the symbols S_0 (with a *strong* bond and three *null* bonds) and E_0 (with a *weak* bond, and three *null* bonds) are used to denote the start and the end of input numbers, respectively. They set limits on the extent of the calculation or patterning, which will facilitate a modular approach to the process. The strength of the binding domains for the side with the double line is set to be 2. In terms of every system's computing order, the positions of S_0 and E_0 will be exchanged possibly. Therefore, we have a unique representation S for every two input binary numbers. This is the form in which we would store all numbers in our systems.

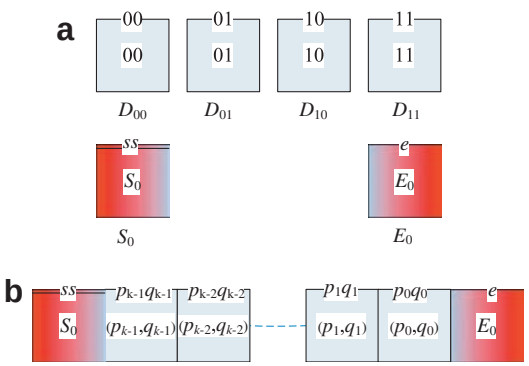

Fig. 2. The tiles representation of two input numbers. (a) A graphical representation of Γ; (b) the configuration for two k-bit input numbers.

3.1. Comparator tile system

The system compares the two input numbers and determines their relationship which is less than ($<$), greater than ($>$), or equals ($=$). It compares the two input numbers bit-by-bit from the highest bit to the lowest bit until the relationship is determined. Here, we will describe the comparator tile system which uses $O(1)$ distinct tiles in a linear time to compare the size of two input numbers.

Fig. 3 depicts the concept behind the tiles and the 12 actual tiles with south and west sides as the input sides and the east side as the output side in the comparator tile system. The symbols on the sides of the tiles indicate the binding domains of these sides. The value in the middle of each tile t indicates its $v(t)$ value. The concept includes variables a, b, and u; here, $a, b \in \{0, 1\}$, $u \in \{=, <, >\}$. Let $r(a, b)$ denote the relationship of variables a and b. The output value $R(u, a, b)$ depends on $r(a, b)$ and u, if the value of u is $=$, then the value of $R(u, a, b)$ is the same as that of $r(a, b)$; otherwise, the value of $R(u, a, b)$ is the same as that of u. The four boundary tiles used in the comparator tile system are shown in Fig. 4. Each tile's name is written on its top. The strength of the binding domains for the side with the double line is set to be 2. The seed configuration S_R for all the possible two input numbers and a sample seed configuration which encodes two numbers, $p = 1010011_2$ and $q = 1000101_2$, are also shown in Fig. 4. The start tile (boundary tile), which is described as the starting point, is on the left side. Fig. 4(d) gives the final configuration for the example of 1010011_2 and 1000101_2 with the result $>$ encoded on the right boundary tile b3.

Lemma 1. *Let* $\Sigma_R = \{00, 01, 10, 11, <, =, >\}$, *and* T_R *be the set of tiles as defined in Fig.* 3(*b*), *let* $g_R = 1$, $\tau_R = 2$, *and* S_R *be a seed configuration. Let* p *and* q *be the numbers to be compared,* k_p *and* k_q *be the sizes of* p *and* q *in bits, respectively. Let* $k = max(k_p, k_q)$. *If* $k_q < k$, *the number* q *needs to be padded to be* k *bits long with extra* 0 *in the* q*'s high bit, and if* $k_p < k$, *the number* p *needs to be padded to be* k *bits long with extra* 0 *in the* p*'s high bit. Let* $\mathbb{S}_R = (T_R, S_R, g_R, \tau_R)$, *then there exists some* $(x_0, y_0) \in \mathbb{Z}^2$ *such that* $S_R(x_0 + 1, y_0 - 1) = E_0$, $S_R(x_0 - k, y_0 - 1) = S_0$,

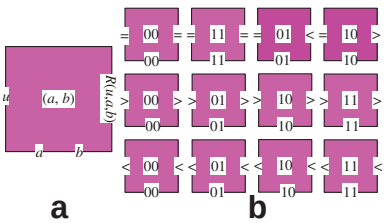

Fig. 3. The tiles in T_R. (a) The concept behind the tiles; (b) the 12 actual tiles.

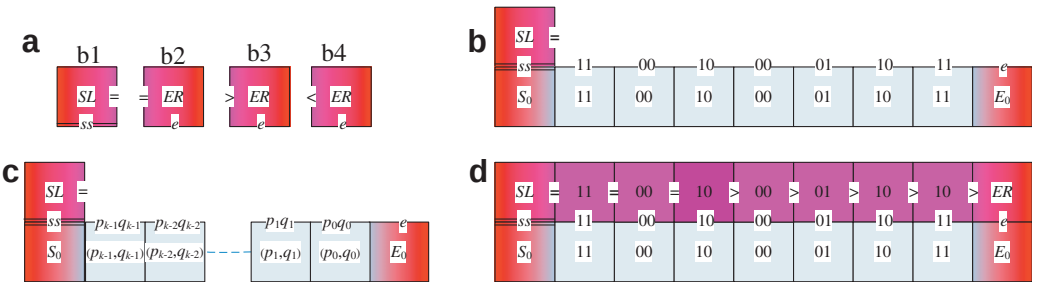

Fig. 4. Seed configuration S_R and a sample of $\mathbb{S}_R$. (a) Four boundary tiles; (b) the seed configuration S_R; (c) a sample seed configuration; (d) the final configuration for the example of 1010011_2 and 1000101_2 with the result >.

$S_R(x_0 - k,\ y_0) = SL$; for all $i \in \{0, 1, \ldots, k-1\}$, $bd_N(S_R(x_0 - i,\ y_0 - 1)) = p_iq_i$, for all other positions (x, y), $S_R(x, y) = empty$. Then, $\mathbb{S}_R$ produces a unique final configuration F_R based on S_R and can determine the relationship of two input numbers, and the assembly time is $O(k)$. In the following divider system, the divisor when compared with the dividend will be right shifted with a bit position.

Proof. Consider the tile system $\mathbb{S}_R$. Let $\Gamma_R = \{D_{00} = \langle 00, null, null, null\rangle, D_{01} = \langle 01, null, null, null\rangle, D_{10} = \langle 10, null, null, null\rangle, D_{11} = \langle 11, null, null, null\rangle, S_0 = \langle ss, null, null, null\rangle, E_0 = \langle e, null, null, null\rangle, SL = \langle null, =\text{ss}, null\rangle\}$. Let the seed configuration S_R: $\mathbb{Z}^2$ to Γ_R be

$$\begin{cases} S_R(1,-1) = E_0 \\ S_R(-k,-1) = S_0, \quad \text{and} \quad S_R(-k,0) = SL \\ \forall i \in \{0,\ldots,k-1\}, S_R(-i,-1) = Dp_iq_i \\ \text{For all other } (x,y) \in \mathbb{Z}^2, S_R(x,y) = empty \end{cases}$$

It is clear that there is only a single position where a tile may attach to S_R, and after that tile attaches there will be only a single position where a tile may attach, too. By induction, because $\forall t \in T_R$, the duple $\langle bd_S(t), bd_W(t)\rangle$ is unique, and because $\tau_R = 2$, it follows that $\mathbb{S}_R$ produces a unique final configuration F_R on S_R. □

Let $R_i(p, q)$ denote the relationship of $p_{k-1}\ldots p_i$ and $q_{k-1}\ldots q_i$. The value of $R_i(p, q)$ depends on $r(p_i, q_i)$ and $R_{i+1}(p, q)$. Formally, $R_k(p, q)$ is =, and if the value of $R_{i+1}(p, q)$ is =, then the value of $R_i(p, q)$ is the same as the value of $r(p_i, q_i)$; otherwise, the value of $R_i(p, q)$ is the same as the value of $R_{i+1}(p, q)$. For all $0 \leqslant i \leqslant k-1$, S_R and F_R agree on $(-i, -1)$, $(-k, -1)$ and $(-k, 0)$. Therefore, $bd_N(F_R(-i,\ -1)) = p_iq_i$. From the definition of S_R, the $bd_E(F_R(-k,\ 0))$ is =, the $bd_E(F_R(-(k-1),0)) = r(p_i,\ q_i)$. If $\exists j \leqslant k-1$, $p_j \neq q_j$, and for all $i > j$, p_i and q_i are equal, then for all $i > j$, $bd_E(F_R(-(i,\ 0)))$ is =, for all $0 \leqslant i \leqslant j$, $bd_E(F_R(-i,\ 0)) = r(p_j,\ q_j)$; otherwise, for all $0 \leqslant i \leqslant k-1$, $bd_E(F_R(-i,\ 0))$ is =. When the relationship is determined and right boundary tile $ER = \langle null, null, e, bd_E(F_R(0,\ 0))\rangle$ attaches to the position (1, 0), the comparing process will terminate. This indicates that $\mathbb{S}_R$ can produce a unique final configuration on S_R and give the relationship of two input numbers.

Because $\tau_R = 2$, only a tile with two neighbors may attach at any time in this system, and so no tile may attach until its left neighbor has. Thus, the assembly time is $k + 1$ steps to compare two k-bit numbers.

3.2. Duplicator tile system

As we know, the comparison operation starts only from the highest bit, while the subtract operation starts from the lowest bit in the division operations. In the process of division, after a round comparing (from the highest bit to the lowest bit) has been completed, if minuend (dividend) is less than subtrahend (divisor), the subtrahend (divisor) needs right shift with one bit. Then, we need to carry out

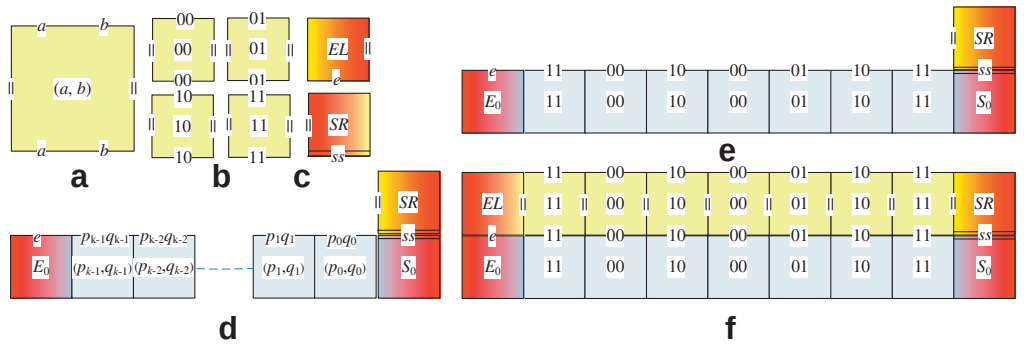

Fig. 5. The types of the tiles in $T_{\|}$ and an example of $\mathbb{S}_{\|}$. (a) The concept behind the tiles; (b) the actual tiles; (c) the boundary tiles in the system; (d) the seed configuration $S_{\|}$; (e) a sample seed configuration; (f) the final configuration for the example which encodes two numbers 1010011_2 and 1000101_2.

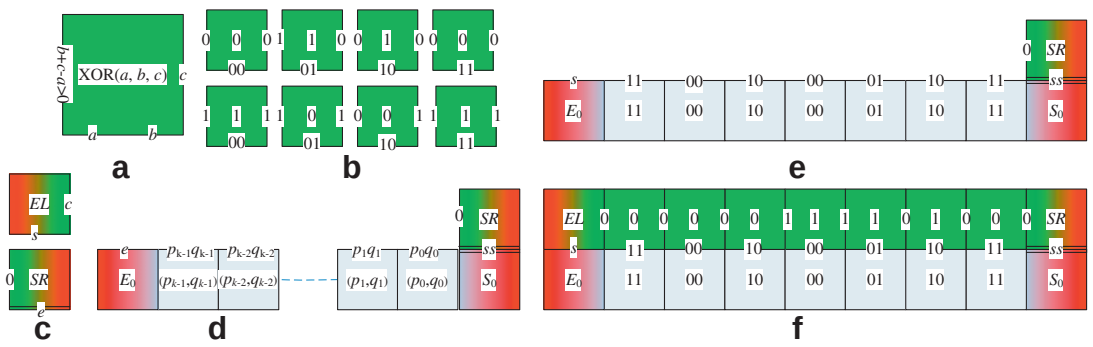

Fig. 6. The types of the tiles in $T_{\_}$ and an example of $\mathbb{S}_{\_}$. (a) The concept behind the computational tiles; (b) the actual tiles; (c) the boundary tiles in the system; (d) the seed configuration $S_{\_}$; (e) a sample seed configuration; (f) a final configuration of 1010011_2–1000101_2 with the solution 0001110_2 encoded on the top row.

a round copying which will copy the information shifted by the comparator system from the lowest bit to the highest bit, so a round comparing can begin from the highest bit again.

The duplicator tile system simply copies upwards the input from the bottom row. This is a straightforward system. The concept behind the tiles in the duplicator tile system with east and south sides as the input sides, and with the west side and the north side as the output sides is depicted in Fig. 5. The concept includes variables a, $b \in \{0, 1\}$. The symbols on the sides of the tiles indicate the binding domains of these sides. The value in the middle of each tile t indicates its $v(t)$ value. The 4 actual tiles and the two boundary tiles used in the duplicator tile system are also given in Fig. 5. The strength of the binding domains for the side with the double line is set to be 2. The seed configuration $S_{\|}$ for all the possible two inputs and a sample seed configuration which encodes two numbers $p = 1010011_2$ and $q = 1000101_2$ are shown in Fig. 5(d) and Fig. 5(e), respectively. Fig. 5(f) describes a sample execution of the $S_{\|}$ system.

Lemma 2. *Let* $\Sigma_{\|} = \{00, 01, 10, 11, \|\}$, *and* $T_{\|}$ *be the set of tiles as defined in Fig.* 5(*b*). *Let* $g_{\|} = 1, \tau_{\|} = 2$, *and* $S_{\|}$ *be a seed configuration. Let* $\mathbb{S}_{\|} = (T_{\|}, S_{\|}, g_{\|}, \tau_{\|})$, *then there exist some positions* $(x_0, y_0) \in \mathbb{Z}^2$ *such that*: $S_{\|}(x_0 + 1, y_0 - 1) = S_0, S_{\|}(x_0 - k, y_0 - 1) = E_0, S_{\|}(x_0 + 1, y_0) = SR$; *for all* $i \in \{0, 1, \ldots, k-1\}, bd_N(S_{\|}(x_0 - i, y_0 - 1)) = p_i q_i$, *for all other positions* (x, y), $S_{\|}(x, y) = empty$. *Then* $\mathbb{S}_{\|}$ *produces a unique final configuration* $F_{\|}$ *on* $S_{\|}$ *and can copy the input information. The assembly time is* $O(k)$.

Proof. Consider the tile system $\mathbb{S}_{\|}$. Let $\Gamma_{\|} = \{D_{00} = \langle 00, null, null, null\rangle, D_{01} = \langle 01, null, null, null\rangle, D_{10} = \langle 10, null, null, null\rangle, D_{11} = \langle 11, null, null, null\rangle, S_0 = \langle ss, null, null, null\rangle, E_0 = \langle e, null, null, null\rangle, SR = \langle null, null, ss, \|\rangle\}$. Let the seed configuration $S_{\|}$: $\mathbb{Z}^2$ to $\Gamma_{\|}$ be

$$\begin{cases} S_{\|}(1, -1) = S_0 \\ S_{\|}(-k, -1) = E_0, \quad \text{and} \quad S_{\|}(1, 0) = SR \\ \forall i \in \{0, \ldots, k-1\}, S_{\|}(-i, -1) = Dp_i q_i \\ \text{For all other } (x, y) \in \mathbb{Z}^2, S_{\|}(x, y) = empty \end{cases} \quad \square$$

Let that final configuration be $F_{\|}$. For all $0 \leqslant i \leqslant k-1$, $S_{\|}$ and $F_{\|}$ agree on positions $(-i, -1)$, $(-k, -1)$ and $(1, 0)$. Therefore, $bd_N(F_{\|}(-i, -1)) = p_i q_i$, $v(F(-i, 0)) = p_i q_i$, and $bd_W(F_{\|}(-i, 0)) = \|$. Consider the tile t that attaches in position $(0, 0)$. By the definition of $S_{\|}$, $bd_W(F_{\|}(1, 0)) = \|$, and by the inductive hypothesis, $bd_N(F_{\|}(0, -1)) = p_0 q_0$. Thus, t's east binding domain must be $\|$ and the south binding domain must be $bd_N(F_{\|}(0, -1))$. So, t must be $\{null, \|, v(F_{\|}(0, -1)), \|\}$. The same argument holds for the tile to the west of the tile, and so on until position $(-k, 0)$, where $F_{\|}(-k, 0)$ is the special tile encoding the symbols EL.

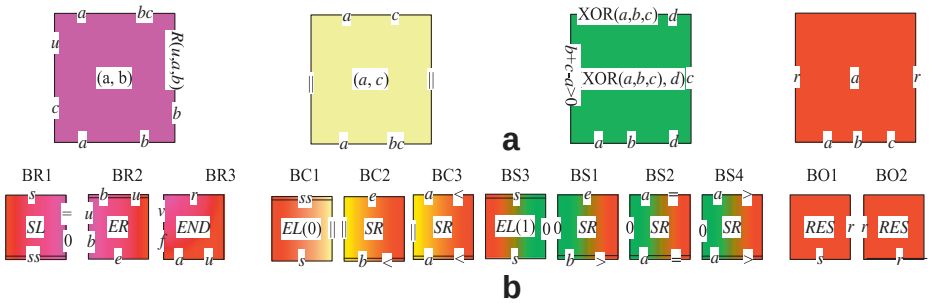

Fig. 7. The concepts behind the tiles in T_D and the boundary tiles including variables $a, f \in \{0, 1\}$, $b, c, d \in \{0, 1, 0'\}$, $u, v \in \{=, <, >\}$ and constants e, ee, s, ss. (a) Computational tiles; (b) boundary tiles corresponding to the computational tiles. Each tile's name is written on its top.

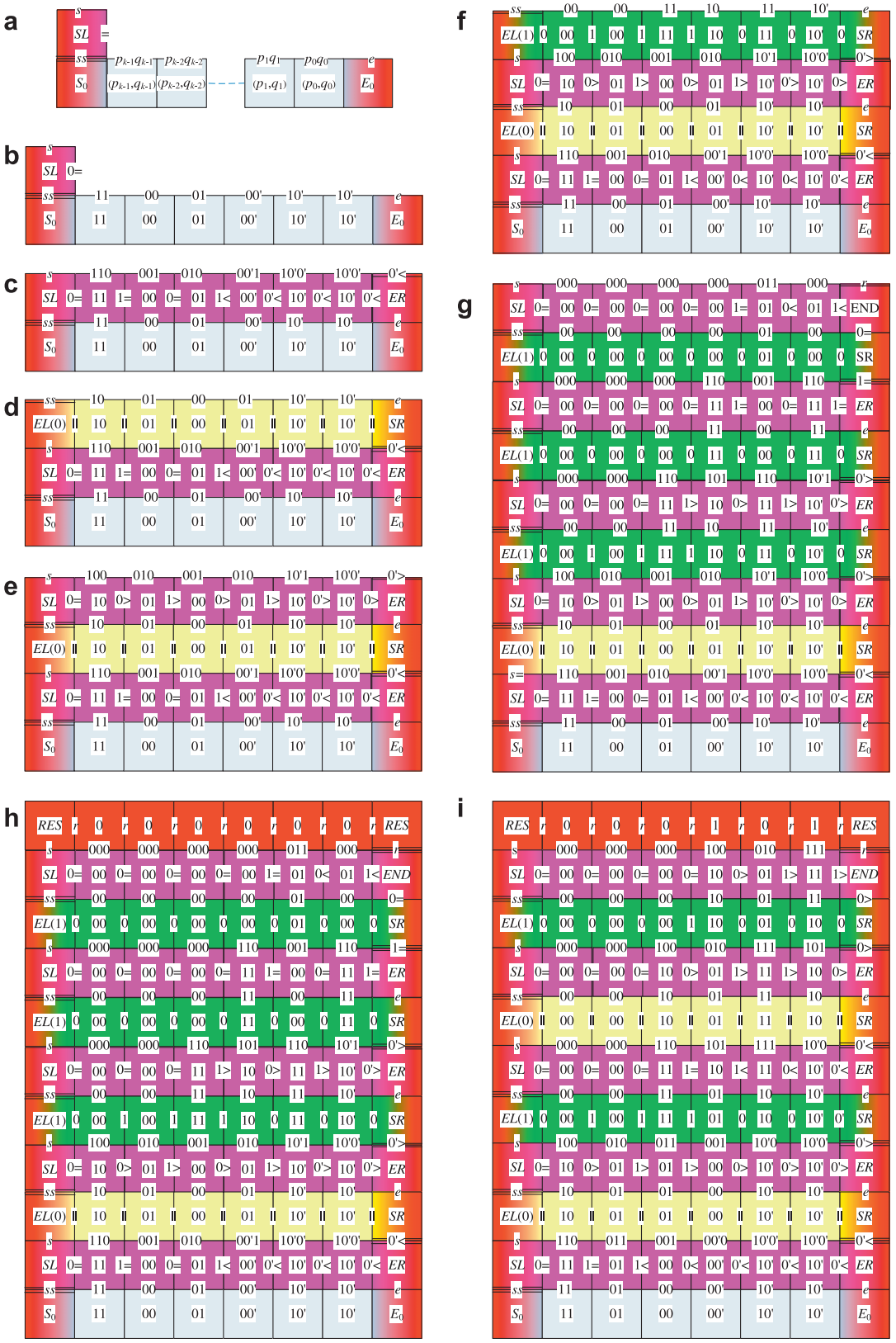

Fig. 8. Two examples of $\mathbb{S}_D$. (a) The seed configuration S_D; (b) a sample seed configuration which encodes two numbers in binary: $p = 100011_2$, $q = 101_2$; (c)–(g) the pre-final stage of the computation when the last layer representing the result has been computed; (h) the final stage of the computation, the leftmost column reads the solution 0111_2; (i) another example not divided exactly, where $p = 100011_2$, $q = 110_2$, the quotient and remainder equal 101_2.

It is obvious that only tiles of $T_{\|}$ are in $F_{\|}$. It follows that $\mathbb{S}_{\|}$ copies the information p and q. Because $\tau_{|} = 2$, only a tile with two neighbors may attach at any time, and therefore, no tile may attach until its right neighbor has. Thus, the assembly time for this system is $k + 1$ steps.

3.3. Subtractor tile system

The logic of the subtraction operation is identical to a series of one-bit full subtractors which consists of three inputs and two outputs. Two of the input bits represent minuend and subtrahend bits to be subtracted, respectively. The third input represents the borrow bit from the previous higher significant position. The first output gives the value of the difference for minuend and subtrahend bits to be subtracted. The second output gives the value of the borrow bit to minuend and subtrahend bits to be subtracted.

In the tile system, each solution tile takes two bits from the inputs on the south side and a borrow bit from the previous solution tile on the east side, and outputs the next borrow bit on the west side. The second output which gives the value of the difference for minuend and subtrahend bits will be described in terms of the type of tile. Then, $T\_$ has only eight tiles with the east and south sides as the input sides and the west side as the output side. The type of the tile is determined by the *sum* of the inputs, mod 2. Fig. 6(a) describes the concept behind tiles, the concept includes variables $a, b \in \{0, 1\}$. The eight tiles for all possible binding domains for the two input sides are given in Fig. 6(b). The 1 or 0 in the middle of the tile t is that tile's $v(t)$ value. Fig. 6(c) shows the two boundary tiles used in the subtractor tile system. They are used to denote the start and the end of computation. If minuend is less than subtrahend, then the variable c of the *EL* tile is 1, otherwise 0 (we only consider the minuend is greater than the subtrahend), the start tile *SR* is on the right side. The seed configuration $S\_$ for all the possible two input numbers, and a sample seed configuration which encodes two numbers: $p = 1010011_2$, $q = 1000101_2$ are also shown in Fig. 6. The borrow value from the $(i-1)$th bit to the ith bit is denoted by $c_i(p, q)$ (named "borrower"). We set $c_0(p, q) = 0$, because the borrower c_0 does not exist, and so $bd_W(SR)$ is 0. Fig. 6(f) shows the final configuration of $1010011_2 - 1000101_2$ with the solution 0001110_2 encoded on the top row.

Lemma 3. *Let* $\Sigma\_ = \{0,1,00,01,10,11\}$. *Let* $T\_$ *be the set of tiles as defined in Fig.* 6(*b*), *let* $g\_ = 1$, $\tau\_ = 2$, *and* $S\_$ *be a seed configuration. Let p and q be the numbers to subtract and let k and* k_q *be the sizes, in bits, of p and q, respectively. If* $k_q < k$, *the number q needs to be padded to be k bits long with extra* 0 *in the q's high bit. Let* $\mathbb{S}\_ = (T\_, S\_, g\_, \tau\_)$. *Then, there exists some* $(x_0, y_0) \in \mathbb{Z}^2$ *such that*: $S\_(x_0 + 1, y_0 - 1) = S_0$, $S\_(x_0 - k, y_0 - 1) = E_0$, $S\_(x_0 + 1, y_0) = SR$; *for all* $0 \leqslant i \leqslant k-1$, $bd_N(S\_(x_0 - i, y_0 - 1)) = p_i q_i$. *For all other positions* (x, y), $S\_(x, y) = empty$. *Then,* $\mathbb{S}\_$ *produces a unique final configuration* $F\_$ *on* $S\_$ *to compute the difference of numbers p and q. The assembly time is* $O(k)$.

Proof. Consider the tile system $\mathbb{S}\_$. Let $\Gamma\_ = \{D_{00} = \langle 00, null, null, null\rangle, D_{01} = \langle 01, null, null, null\rangle, D_{10} = \langle 10, null, null, null\rangle, D_{11} = \langle 11, null, null, null\rangle, S_0 = \langle ss, null, null, null\rangle, E_0 = \langle e, null, null, null\rangle, SR = \langle null, null, ss, 0\rangle\}$. Let the seed configuration $S\_: \mathbb{Z}^2$ to $\Gamma\_$ be

$$\begin{cases} S\_(1,-1) = S_0 \\ S\_(-k,-1) = E_0, \quad \text{and} \quad S\_(1,0) = SR \\ \forall i \in \{0,\ldots,k-1\}, S\_(-i,-1) = Dp_i q_i \\ \text{For all other } (x,y) \in \mathbb{Z}^2, S\_(x,y) = empty \end{cases}$$

It is obvious that there is only a single position where a tile may attach to $S\_$. After that tile attaches there will only be a single position where a tile may attach. By induction, because $\forall t \in T\_$, the duple $\langle bd_S(t), bd_E(t)\rangle$ is unique, and $\tau\_ = 2$. Thus, it follows that $\mathbb{S}\_$ produces a unique final configuration on $S\_$, and the assembly time of this system is $k + 1$ steps. □

Let that configuration be $F\_$. For all $0 \leqslant i \leqslant k-1$, $S\_$ and $F\_$ agree on positions $(-i, -1)$, $(-k, -1)$ and $(1, 0)$. Therefore, $bd_N(F\_(-i, -1)) = p_i q_i$, $v(F\_(-i, 0)) = (p - q)_i$ and $bd_W(F\_(-i, 0)) = c_{i+1}(p, q)$. From the definition of $S\_$, $bd_W(F\_(1, 0))$ is 0, named $c_0(p, q)$. We assume that $bd_W(F\_(-(i-1), 0)) = c_i(p, q)$, then $bd_W(F\_(-i, 0)) = c_{i+1}(p, q)$ and $v(F\_(-i, 0)) = (p - q)_i$. Let $t = F\_(-i, 0)$, the value $v(t)$ is $\mathrm{XOR}(bd_S^L(t), bd_E(t), bd_S^R(t))$. Since t binds with strength 2, $bd_S(t) = bd_N(F\_(-i, -1))$, $bd_E(t) = bd_W(F\_(-(i-1), 0))$, so $v(t)$ is the XOR of a_i, b_i, and $c_i(p, q)$. The ith bit of $p - q$ is exactly that of XOR. If $q_i + c_i(p, q) - p_i > 0$, the binding domain $bd_W(t)$ is defined as 1, and 0 otherwise. Thus, for all $1 \leqslant i \leqslant k-1$, $v(F\_(-i, 0)) = (p - q)_i$. It is obvious that those are the only tiles of $T\_$ in $F\_$.

3.4. Divider tile system

On many implementations, division is rather slow compared with integer multiplication or other integer arithmetic and logical operations. Division, in binary arithmetic, is generally accomplished by successive compare, shift and subtract operations. Therefore, the divider system is a combination of the subtractor system, the duplicator system and the comparator system. In the system, it will produce its remainder on the top row and its quotient on the leftmost column. According to its remainder, we can know the dividend is divisible by a given divisor.

Here, we illustrate how to combine subtractor, duplicator, with comparator systems to create a divider system. For the most part, each system uses a disjoint set of binding domains, sharing binding domains only where tiles from the different systems are designed to interact. As a result, tiles from each system have a particular set of positions where they can attach: tiles from T_R can only attach in even rows, tiles from $T_{\|}$ and $T\_$ can only attach in odd rows. In order to share binding domains of these tiles from the sets T_R, $T_{\|}$ and $T\_$, we redefined the concepts behind tiles. Fig. 7(a) shows all the concepts in the T_D, the pink

tiles perform comparing operations, the yellow tiles perform copying operations, viridian tiles perform subtracting operations, and red tiles are used to store the final result of the computation. The corresponding boundary tiles are given under these concepts, as shown in Fig. 7(b). These concepts include variables a, b, c, d, and u. Let p and q be the dividend and divisor, and k and k_q be the sizes, in bits, of p and q, respectively. If $k_q < k$, the number q needs to be padded to be k bits long with extra 0 in the q's low bit. For convenience, we let $0'$ denote the padded 0, where the $0'$ serve as the value 0 in calculation.

In this study, the division is accomplished as follows: First, a k-bit divisor is compared with a k-bit dividend, at the same time, the divisor is copied and shifted one bit position to the right relative to the dividend. If the dividend is smaller than the k-bit divisor, copying operations will be carried out. It copies only the dividend and shifted divisor. After the operations are carried out, a "zero" is deposited in the left boundary tile (quotient register) used for the duplicator tile system. Then, the $(k-1)$-bit divisor is compared to the dividend again. If the divisor again is greater, a "zero" is deposited in the left boundary tile on the top of the last deposited bit therein. However, the divisor should be smaller than or equal to the dividend. The divisor is subtracted from the portion of the dividend in question, leaving what is referred to as a partial remainder (PR). Moreover, a "one" is deposited in the boundary tile used for the subtractor system on the top of the last deposited bit.

This whole process repeats with the divisor compared with the PR. If the PR is greater than or equal to the divisor, the bit deposited in the quotient register is a "one" and a new PR is computed from the subtraction of the divisor from the old PR; otherwise, copying is done and a "zero" is deposited in the quotient register. In each case, the divisor is right shifted to become a new divisor. Each repetition is known as an "iteration". At the end of any iteration, the results of the division to that point in time can be found in the quotient register with the remainder in PR. The process may continue until all extra $0'$ bits in q have been removed.

Let us now look at an example of the division operation by our method. For simplicity, we will consider six bits number $p = 100011_2$ and three bits number $q = 101_2$. A sample which encodes two numbers in binary: $p = 100011_2$, $q = 101_2$ is shown in Fig. 8(h), the leftmost column reads the solution 0111_2. Fig. 8(i) shows another example not divided exactly, where $p = 100011_2$, $q = 110_2$, the quotient and remainder equal 101_2.

Lemma 4. *Let* $\Sigma_D = \{0, 1, 0', 00, 01, 10, 11, 00', 10', 0'1, 0'0, 0'0', 000, 001, 010, 011, 100, 101, 110, 111, 00'0, 00'1, 10'0, 10'1, 00'0', 10'0', 0=, 1=, 0<, 1<, 0>, 1>, 0'>, 0'=, 0'<\}$. *Let* T_D *be the set of tiles as defined in Fig.* 7(*a*), *let* $g_D = 1$, $\tau_D = 2$, *and* S_D *be a seed configuration. For all* $i \in \mathbb{N}$, *let* p_i, $q_i \in \{0, 1\}$, $c = p/q$. *Let* $\mathbb{S}_D = (T_D, S_D, g_D, \tau_D)$. *Then there exists some* $(x_0, y_0) \in \mathbb{Z}^2$ *such that*: $S_D(x_0 + 1, y_0 - 1) = E_0$, $S_D(x_0 - k, y_0 - 1) = S_0$, $S_D(x_0 - k, y_0) = SL$; *for all* $i \in \{0, 1, 2, \ldots, k-1\}$, $bd_N(S_D(x_0 - i, y_0 - 1)) = p_i q_i$; *for all other positions* (x, y), $S_D(x, y) = empty$. *Then,* $\mathbb{S}_D$ *produces a unique final configuration* F_D *on* S_D *and can compute the quotient of number p and number q.*

Proof. Consider the tile system $\mathbb{S}_D$. Let $\Gamma_D = \{D_{00} = \langle 00, null, null, null\rangle, D_{01} = \langle 01, null, null, null\rangle, D_{10} = \langle 10, null, null, null\rangle, D_{11} = \langle 11, null, null, null\rangle, D_{10'} = \langle 10', null, null, null\rangle, D_{00'} = \langle 00', null, null, null\rangle, S_0 = \langle ss, null, null, null\rangle, E_0 = \langle e, null, null, null\rangle, SL = \langle s, 0=, ss, null\rangle\}$, where E_0, S_0, and SL are boundary tiles. Then, the seed configuration S_D: $\mathbb{Z}^2$ to Γ_D is

$$\begin{cases} S_D(1,-1) = E_0 \\ S_D(k,-1) = S_0, \quad \text{and} \quad S_D(k,0) = SL \\ \forall i \in \{0,\ldots,k-1\},\ S_D(-i,-1) = Dp_iq_i \\ \text{For all other } (x,y) \in \mathbb{Z}^2, S_D(x,y) = empty \end{cases} \qquad \square$$

It is clear that there is only a single position where a tile may attach to S_D, and after that tile attaches there will only be a single position where a tile may attach, too. By induction, $\forall t \in T_D$, the $\langle bd_S(t), bd_E(t)\rangle$ or $\langle bd_W(t), bd_S(t)\rangle$ is unique. Because $\tau_D = 2$, it follows that $\mathbb{S}_D$ produces a unique final configuration on S_D. Let the final configuration be F_D.

Consider the seed configuration S_D (such as Fig. 8(b)) which is the same as Fig. 4(a), by Lemma 1, it can produce a unique configuration F_D^1 (such as Fig. 8(c)) on S_D and determine the relationship (RS1) of two input numbers. At the end of the comparing, a comparing boundary tile with compared result RS1 will attach to position (1, 0).

If the result RS1 is $<$, then the only copying boundary tile can attach at the top of that comparative boundary tile. By Lemma 2, it will produce a unique configuration F_D^2 (such as Fig. 8(d)) on F_D^1, and copy the dividend and the shifted divisor. At the end of the copying, a copying boundary tile will attach to position $-(k, 1)$. Thereafter, consider the tile t that attaches to position $(-k, 2)$, the $bd_S(t)$ must be ss. By the inductive hypothesis, because $\tau_D = 2$, t must be BR1, it goes to the next round comparison.

Otherwise, the only subtracting boundary tile can attach to the top of that comparing boundary tile. By Lemma 3, it can produce a unique configuration F_D^2 on F_D^1 and execute the subtraction. At the end of subtraction, a subtracting boundary tile will attach to position $(-k, 1)$. Thereafter, the tile t that attaches in position $(-k, 2)$, the $bd_S(t)$ must be ss. By the inductive hypothesis, because $\tau_D = 2$, t must be BR1. It goes to the next round comparison.

After all extra 0's are removed from input number q, the final round copy (or subtraction) and comparison have been implemented. Afterwards, the result tiles will attach in the last row. The computing is terminated.

Lemma 5. *The assembly time of $\mathbb{S}_D$ producing a unique final configuration on S_D is $O(k^2)$.*

Proof. Because $\tau_D = 2$, only a tile (not boundary tile) with two neighbors may attach at any time, and so no tile may attach until its left neighbor (or right neighbor) has. Let c be the quotient of p divided by q and let k_c be the size, in bits, of c. For each bit of c, a round compare and subtract (or copy) will be executed, and the time complexity of these operations is linear. Thus, the assembly time is $2k \times k_c$ steps. Because $k_c \leqslant k/k_q + 1$, the time complexity is $O(k^2)$. □

4. Application to factoring integers

Parallelism reveals itself in many ways in computation by self-assembly. Each superstructure may contain information representing a different calculation (global parallelism). Due to the extremely small size of DNA strands, as many as 10^{18} DNA tiling assemblies may be made simultaneously in a small test tube. Then, it is possible that biological computing will provide a huge amount of parallelism for dealing with the large scale computationally intensive problems in the real world [25].

4.1. Factoring integer

The problem of finding the prime factors of large composite numbers has always been of mathematical interest. With the advent of the RSA system it is also of practical importance, because the security of the RSA system depends on the difficulty of factoring the public keys. Factoring a positive integer N means finding positive integers u and v such that the product of u and v equals N, and such that both u and v are greater than 1. Factoring a composite integer is believed to be a hard problem. That is, of course, not the case for all composites (composites with small factors are easy to factor). As yet there is no firm mathematical ground on which this assumption can be based. The only evidence that factoring is hard consists of our failure so far to find a fast and practical factoring algorithm.

Trial division is the simplest algorithm for factoring an integer. We assume that u and v are nontrivial factors of N such that $u \times v = N$ and $u \leqslant v$. To perform the trial division algorithm, one simply checks whether $u|N$ for $u = 2, \ldots, \lfloor\sqrt{N}\rfloor$. When such a divisor u is found, then $u = N/v$ is also a factor, and a factorization has been found for N.

Recent experimental developments on algorithmic self-assembly using DNA tiles seem to offer the most promising path towards a potentially useful application of the DNA computing concept. In some implementations of the tile assembly model systems, many assemblies happen in parallel. In fact, it is often almost impossible to create only a single assembly, and thus we can take advantage of the parallelism of self-assembly to execute the trial division by the divider described in the previous section.

Assuming that in a divider, the length for N, the product of two prime numbers of k bits is $2k$ bits. When N is divided by M, M is one of the two prime numbers if the remainder is equal to zero. It is obvious that the division instruction is finished in parallel through checks by self-assembly of DNA tiles whether $M|N$ for $M = 2^{k-1} + 1, \ldots, \lfloor\sqrt{N}\rfloor$, where M is only odd.

4.2. Design protocols to execute integer factoring

Barua et al. [26] showed how the DNA self-assembly process can be used for computing finite field multiplication and addition, and they provided the detailed realization scheme using the TAE tile, the TAO tile and the rotated TX tile. Here, we can make use of a similar method for the factoring integer. In order to execute practically the integer factoring using the method described in the previous section, we design the following experimental protocol.

① *Inputting/outputting to tiling assemblies:* Input and output are critical to the practical use of DNA-based computing. The TX tiles have an interesting property, namely that certain distinguished single-stranded DNA (called scaffold or reporter strands) wind through all the tiles of a tiling assembly. These strands are used for building up a long strand which records inputs and outputs for the entire assembly computations [27]. Here, the numbers N and M which are in the form of ssDNA as in Fig. 9 can be converted to appropriate tile structures (as in Fig. 1).

② *Encoding:* To program a tiling assembly for the implementation by our design, a key computational challenge in DNA self-assembly is to design a series of suitable nucleotide DNA sequences to encode all the symbols used in our computation, such as S_0, E_0, $D(0, 0)$, and $D(0, 1)$. The encoded sequences (and their complementary sequences) must be sufficiently different from each other. Abundant algorithms and software have been developed to design DNA sequences which can correctly assemble into desired DNA secondary structures in the DNA computing community [28–30].

③ *Constructing the initial input DNA tiles:* The form of the inputs DNA strands is shown in Fig. 9. Now, we encode all possible numbers by joining together the different nucleotide sequences. Then put all the constituent single strands together to anneal and make up the tile.

Fig. 9. ssDNA representation of the two k-bit input numbers.

④ *Constructing the computation tiles and output tiles:* Fig. 7 shows the various computation and output tiles that have to be constructed. These tiles are constructed by first constructing the constituent strands; and then allowing them to anneal together to form the tile. Some copies of these strands would be kept in a storage pool. The strands in the storage pool may be duplicated periodically using PCR. When a new computation starts, the required tiles would be created from the constituent strands taken from the storage pool.

⑤ *Executing the computation and reading the result:* The computation can be executed in a small chamber or cell containing the computation tiles, into which the input tile assemblies are added. The temperature and other conditions of the chamber would be adjusted by external controls to allow for the annealing of complementary sticky ends. At the end of computation, Ligase is added to seal the joints and then, the temperature of the chamber is increased to break up the tile structure into single strands. The result strand is then extracted by affinity purification using the complement of the "RES" sequence. As a result, we will get an ssDNA of the same format as the input strands. To output the result of the computation we would use a modification of the standard sequence-reading operation that uses a combination of PCR and gel electrophoresis.

The actual implementation details are not discussed here since they fall outside the scope of this paper. However, we believe that we make no arbitrary hypotheses. In fact, our work is based on the assumptions and achievements that come with DNA tiling computation in general.

5. Conclusion

DNA self-assembly is expected to be useful in various applications. Our method for implementing arithmetic computation extends the technique used by Brun for binary addition and multiplication. We propose binary subtraction and division operations using the tile assembly model. Each computing operation in our systems is computed very fast through the self-assembly process and the output of one computation can be directly passed as input to another computation. Both systems use $O(1)$ input tiles and compute in $O(k)$ and $O(k^2)$ steps, respectively. Furthermore, we provide a scheme to factor the product of two prime numbers, and it is a breakthrough in basic biological operations using a molecular computer by self-assembly. This feature performs the computation in the presence of a well-defined border which set limits on the extent of the calculation or patterning; combining framed arrays will facilitate a modular approach to the process. The theoretical analysis of systems that can assemble shapes or compute simpler functions has led to experimental manifestations of such systems using DNA tiles. While most DNA computation suffers from high error rates, self-assembly has yielded results in error correction that may be used to decrease the error rates. Most experimental results in DNA computation have not appealed to the advantages of crystal growth; however, these early works on the fundamentals of self-assembly and the physical experimental evidence of actual DNA crystals suggest a bright future for DNA computation.

The field of nanotechnology holds tremendous promise. But many technical hurdles will have to be overcome before algorithmic self-assembly can be developed into a practical commercial technology. If the molecular and supramolecular word can be controlled at will, then it may be possible to achieve vastly better performance for computers and memories. It might open up a host of other applications in materials science, medicine and biology.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant Nos. 60373089, 60674106, 30570431, 30370356, and 60773122), the Program for New Century Excellent Talents in University (Grant No. NCET-05-0612), the Ph.D. Programs Foundation of Ministry of Education of China (Grant No. 20060487014), the Chenguang Program of Wuhan (Grant No. 200750731262), and HUST-SRF (Grant No. 2007Z015A).

References

[1] Adleman LM. Molecular computation of solutions to combinatorial problems. Science 1994;266:1021–4.
[2] Lehn JM. Supramolecular chemistry. Science 1993;260:1762–3.
[3] Adleman LM, Cheng Q, Goel A, et al. Combinatorial optimization problems in self-assembly. In: Annual ACM symposium on theory of computing (STOC); 2002. p. 23–32.
[4] Abelson H, Allen D, Coore D, et al. Amorphous computing. Commun ACM 2002;43:74–82.
[5] Winfree E, Eng T, Rozenberg G. String tile models for DNA computing by self-assembly. LNCS 2001;2054:63–88.
[6] Winfree E. Algorithmic self-assembly of DNA. Ph.D. Dissertation, California Institute of Technology; 1998.
[7] Seeman NC. DNA nanotechnology: novel DNA constructions. Annu Rev Biophys Biomol Struct 1998;27:225–48.
[8] Reif JH. Computing: successes and challenges. Science 2002;296:478–9.
[9] Rozenberg G, Spaink H. DNA computing by blocking. Theor Comput Sci 2003;292:653–65.
[10] Winfree E, Liu F, Wenzler LA, et al. Design and self-assembly of 2D DNA crystals. Nature 1998;394:539–44.
[11] Mao C, Sun W, Seeman NC. Designed two-dimensional DNA Holliday junction arrays visualized by atomic force microscopy. J Am Chem Soc 1999;121:5437–43.
[12] Mao C, LaBean TH, Reif JH, et al. Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. Nature 2000;407:493–6.

[13] Wang H. Proving theorems by pattern recognition I. Bell Syst Tech J 1961;40:1–42.
[14] Robinson RM. Undecidability and nonperiodicity for tilings of the plane. Inv Math 1971;12(3):177–209.
[15] Barish R, Rothemund P, Winfree E. Two computational primitives for algorithmic self-assembly: copying and counting. Nano Letters 2005;5(12):2586–92.
[16] Cook M, Rothemund P, Winfree E. Self-assembled circuit patterns. LNCS 2004;2943:91–107.
[17] Rothemund P, Papadakis N, Winfree E. Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biol 2004;2(12):2041–53.
[18] Brun Y. Arithmetic computation in the tile assembly model: addition and multiplication. Theor Comput Sci 2006;378:17–31.
[19] Brun Y. Nondeterministic polynomial time factoring in the tile assembly model. Theor Comput Sci 2008;395(1):3–23.
[20] Brun Y. Solving NP-complete problems in the tile assembly model. Theor Comput Sci 2008;395(1):31–6.
[21] LaBean TH, Yan H, Kopatsch J, et al. The construction, analysis, ligation and self-assembly of DNA triple crossover complexes. J Am Chem Soc 2000;122:1848–60.
[22] Carbone A, Seeman NC. Molecular tiling and DNA self-assembly. LNCS 2004;2950:61–83.
[23] Reif JH, LaBean TH, Seeman NC. Challenges and applications for self-assembled DNA nanostructures. LNCS 2001;2054:173–98.
[24] Rothemund P, Winfree E. The program-size complexity of self-assembled squares. In: ACM symposium on theory of computing (STOC); 2001. p. 459–68.
[25] Chang WL, Guo MY, Ho M. Fast parallel molecular algorithms for DNA-based computation: factoring integers. IEEE Trans Nanobiosci 2005;4(2):149–63.
[26] Barua R, Das S. Finite field arithmetic using self-assembly of DNA tilings. In: Proceedings of 2003 congress on evolutionary computation; 2003. p. 2529–36.
[27] Reif JH, LaBean TH, Sahu S, et al. Design, simulation, and experimental demonstration of self-assembled DNA nanostructures and motors. LNCS 2005;3566:173–87.
[28] Wei B, Wang Z, Mi Y. Uniquimer: software of de novo DNA sequence generation for DNA self-assembly – an introduction and the related applications in DNA self-assembly. J Comput Theor Nanosci 2007;4(1):133–41.
[29] Yin P, Guo B, Belmore C, et al. TileSoft: sequence optimization software for designing DNA secondary structures, TR-CS-2004-09.
[30] Iimura N, Yamamoto M, Tanaka F, et al. Sequence design for stable DNA tiles. LNCS 2006;4287:172–81.