Scalable Design of Logic Circuits Using an Active Molecular Spider System

Dandan Mo[1], Matthew R. Lakin[1,2,3], and Darko Stefanovic[1,3]

[1] Department of Computer Science, University of New Mexico
[2] Department of Chemical and Biological Engineering, University of New Mexico
[3] Center for Biomedical Engineering, University of New Mexico
{mdd,mlakin,darko}@cs.unm.edu

Abstract. As spatial locality leads to advantages of computation speed-up and sequence reuse in molecular computing, molecular walkers that exhibit localized reactions are of interest for implementing logic computations. We use molecular spiders, which are a type of molecular walkers, to implement logic circuits. We develop an extended multi-spider model with a dynamic environment where signal transmission is triggered locally, and use this model to implement three basic gates (AND, OR, NOT) and a mechanism to cascade the gates. We use a kinetic Monte Carlo algorithm to simulate gate computations, and we analyze circuit complexity: our design scales linearly with formula size and has a logarithmic time complexity.[4]

Keywords: molecular spiders, logic circuits, parallel evaluation, localized signal transmission.

1 Introduction

Molecular walkers are synthetic molecular machines inspired by natural biological motors. Previous studies [4,7,9,11,13] have shown that walkers can move directionally and autonomously on a pre-programmed track via localized reactions. Spatial locality can overcome the challenges of computation speed-up and sequence reuse in molecular computing where all the reactants diffuse freely in a mixed solution [2,5]. Hence, a walker system with inherent spatial locality has potential to perform more complex computational tasks. We investigate the computational power of a walker system by using it to implement scalable logic circuits.

We consider a molecular *spider* system, where a spider is a type of molecular walker. Molecular spiders [1,11,12] with varying number of

[4] Paper appears in the Proceedings of the 10th International Conference on Information Processing in Cells and Tissues, IPCAT 2015, San Diego, CA, September 2015; Springer LNCS Vol. 9303, pp. 13–28; corrected and reformatted.

legs move stochastically on a surface formed by sites containing DNA segments, and present biased behaviors due to different reactions with fresh sites (catalytic cleavage) and visited sites (dissociation). We extend previous models [1, 11, 12] to implement three basic logic gates (AND, OR, NOT), and cascade the gates to construct logic circuits. We use multiple spiders in the model, and we assume spiders behave unbiasedly with equal transition rates to all sites. Sites are divided into *normal sites* that are non-alterable and *functional sites* that can be altered via catalytic cleavage and/or strand displacement. We can encode signals into functional sites. Signal transmission [5, 6] is triggered locally when a spider interacts with a signal-carrying site, which dynamically changes the state of the spider or of the environment. We call this extended system an *active* molecular spider system.

In our design, each variable is represented by a moving spider that has two legs and one arm. The arm has two possible states, 0 or 1, representing the boolean value of the spider. Each gate is represented by a layout of different sites on a 2D lattice. In a single gate, spiders with different values will take different paths from their input locations. We arrange different functional sites on different paths, such that only the spider with the correct computation result will be directed to the output location via interactions between spiders and functional sites. On reaching the output location, a spider reports the computation result, and we call this spider the *reporting* spider. We cascade logic gates by connecting them such that only the reporting spider leaves the upstream gate and enters the downstream gate. We design a mechanism for *exit* from gates to implement gate cascades that allow parallel evaluation. As an example, Figure 7 shows a logic circuit where input spiders X and Y are initially placed at the input locations of two NOT gates, and the NOT gates are connected to the same AND gate via *exit* mechanisms. Spiders move within the circuit, and the spider reaching the output location reports the computation result.

Molecular circuits using *DNA Strand Displacement* [8] in a well-mixed solution use relatively high and low concentration of a species to represent Boolean values 1 and 0, or use two separate species in a dual-rail encoding. Here, we use spiders with arm state 1 or 0 to represent Boolean values, thus we remove potential ambiguity from result reporting. Since Boolean values are carried by spiders moving from an upstream gate to a connected downstream gate, all gates are designed individually, thus, modularity is ensured. Previous work on tethered circuits [2, 5] also ensures modularity and unambiguity, but takes a different approach where

Boolean values are represented by the existence of a sequence. Modularity is ensured by spatially isolating different gates on a surface, e.g., a DNA origami tile such that only gates in close proximity can interact with each other.

Previous work [3] has used a walker system to construct logic circuits with spatial locality, but it lacks modularity and is limited to sequential evaluation due to its design where the circuit constructed is in the form of a *Binary Decision Diagram* (BDD). A walker initially placed at the root node walks along a path unblocked by externally-added strands to reach a leaf node representing True or False, causing a fluorophore change to report the computation result. For practical reasons, this reporting strategy needs two parallel circuits that detect fluorophore change at the True nodes and False nodes respectively to avoid ambiguity. Our design uses the reporting spider to avoid reporting problems [3], and we support parallel evaluation. As a result, to evaluate an m-clause 3-CNF circuit, we need time $O(\log m)$ while the circuit [3] needs time $O(m)$. We use the same linear space complexity $O(m)$ as in the circuit [3], and it is easier to construct large circuits using our design because of its modularity.

Using an extended *active* multi-spider system, while keeping the advantages related to spatial locality, our design ensures modularity, unambiguity, and scalability. We will describe the model in Section 2, and introduce how to construct the logic circuits in Section 3 with simulation results and complexity analysis. A formal definition of the model is given in Section 4. We give conclusions and discuss current challenges and future work in Section 5.

2 Model Description

Our long-term goal is to realize the circuits we describe here with a physical implementation based on molecular spiders [4, 7]. Therefore, our model draws from the existing models of molecular spiders [9, 11] and extends them to describe the richer functionalities of the walkers we hope to build. In spite of these extensions, we will use the evocative term "spider" throughout the paper.

A molecular spider has a body and three limbs, two legs and an "arm", which it can use to attach to chemical sites on a surface. There is exclusion: at most one limb can be attached to a given site at a time. Different types of sites are laid out on a square lattice, $\mathbb{Z}^2$. A set of contiguous sites can form a *track* on which the spiders can move.

We model a spider's body as a single point, and the limbs as having equal length. This leads to the following postulated "hand-over-hand" gait [9]: at any given time, exactly two limbs are attached to the surface, and they are attached to nearest-neighbor sites. We call the sites a limb has bound to the *attachment points*. There are always two attachment points for each spider, and they are adjacent to each other. A moving step occurs when a spider detaches one of its limbs from an attachment point $p \in \mathbb{Z}^2$, and attaches to a site $p' \in \mathbb{Z}^2$. Figure 1 shows a transition step of a spider where there are four *reachable* sites that the spider can potentially transit to. However, a spider might not attach to a reachable site because whether a reachable site is *available* depends on the state of the site and of the limb, which will be discussed in Section 3 and Section 4. When multiple spiders are moving on the track, one spider cannot attach to a site occupied by another spider.

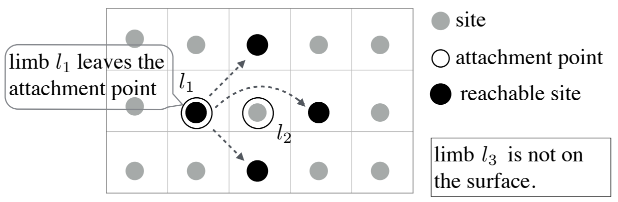
limb l_1 leaves the attachment point
l_1
l_2
site
attachment point
reachable site
limb l_3 is not on the surface.

Fig. 1. A spider has limb l_1 and limb l_2 attached to the surface. When limb l_1 detaches from the left attachment point, four sites represented by the black dots are reachable for limbs l_1 and l_3. The arrows show the transitions of a spider to other sites via *hand-over-hand* movement.

Spiders move stochastically on the track, interacting with the normal sites. If they attach to functional sites, signal transmission is triggered locally between two adjacent sites, or between a site and the spider attached to it. Changes to the sites and spiders may happen during a step, which is crucial in the construction of a logic circuit. In the next section, we will explain how to use different sites to construct three basic gates (AND, OR, NOT) and cascade them to construct a logic circuit.

3 Logic Circuit Construction

Each spider represents a Boolean variable. The value of the spider is indicated by its arm state, which is either 0 or 1. A logic circuit is formed by cascades comprising the basic logic gates (AND, OR, NOT). This combination of logic gates is complete for Boolean logic. A logic gate is an arrangement of different sites on a square lattice, including an output location and input locations. When spiders begin moving from the input locations, their interactions with the sites lead to changes to the sites and the spider values, which ends with one spider reaching the output location, and the value of this spider represents the computation result of the logic circuit. In this paper we do not concern ourselves with the issues of placement and routing of circuits in the plane, which are well studied in electronic circuit design.

3.1 Normal Sites and Functional Sites

We define the set of site types as $S = S_{norm} \cup S_{fun}$, where the *normal* sites $S_{norm} = \{s_l, s_1, s_0\}$ are non-alterable and the *functional* sites in S_{fun} are alterable. A normal site of type s_l binds to a spider's leg, and is used for the "wires" of a logic circuit. Sites of type s_0 and s_1 bind to the spider's arm if it has type 0 or 1, respectively. Sites of type s_0 and s_1 are placed at the beginning of two separate paths that branch out from a junction, directing a spider with different values to different paths (Figure 2).

The junction design is used in the constructions for all gate types. Each logic gate has a set of functional sites placed on the paths branching out from the junction. After the spiders take their own paths at the junction according to their values, they will encounter different functional sites. The interactions between the spiders and the functional sites cause changes to the spider and the sites, directing one spider to the output location, reporting the result of the gate computation.

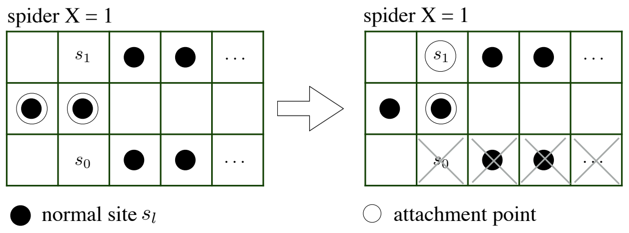

Fig. 2. If a spider has an arm type of 1, it binds to site s_1 at a junction. If a spider has an arm type of 0, it binds to site s_0 at a junction. Here a spider $X = 1$ follows the upper path by attaching to site s_1. It cannot follow the lower path.

Before going to the details of each gate, we first introduce some important features of functional sites. (1) A functional site has a state among $\{\texttt{on}, \texttt{off}, \texttt{trapped}\}$. The spider can bind to an "on"-state site, cannot bind to an “off”-state site, and cannot leave a “trapped”-state site by itself. (2) A functional site may or may not trap a spider. When it traps a spider, the site’s state becomes “trapped”. (3) A functional site may contain a signal of “turning on/off” or “switching to 1 or 0”. The signal held in a functional site is sent out once it is attached by a spider. When a spider attaches to a site holding a signal, the signal “turning on/off” is sent to another site, setting its state “on” or “off”; the signal “switching to 1 or 0” is sent to the spider, changing its value to 1 or 0. When a functional site sends out its signal, it has no signal remaining. Signal transmission is allowed between a site and a spider that is attached to the site, or between two sites that are adjacent to each other. These features could be implemented via DNA strand displacement. We will discuss the AND and OR gate designs in Section 3.2 and the NOT gate design in Section 3.3.

3.2 Designs of the AND and OR Gates

We use three types of functional sites s_t, s_p, and s_u in the designs of the AND gate and OR gate. Site s_t can trap the spider attaching to it, so we place a site s_t at the output location of the gate. The AND gate and OR gate each has two input spiders initially located at the two input locations, which are two junctions as shown in Figure 2. Each input spider selects one of two possible paths when computation begins, where one path leads to the output location without any functional sites and the other path is merged into a crossroad in the middle of the lattice. We place an initially “off”-state site s_p at the heart of the crossroad, which blocks the central path from the crossroad to the output location. We place a site s_u adjacent to site s_p, which will send a ‘turning-on” signal to unblock site s_p when a spider attaches to it, and trap that spider at the same time. The cooperation between sites s_u and s_p guarantees that only when both spiders meet at the crossroad can a spider take the central path to the output location.

Figure 3 shows the layout of the AND gate and OR gate. We explain how the AND gate works under four possible input assignments, and the OR gate follows a similar design. In the AND gate, the two input spiders X and Y are initially placed at two junctions as their input locations. When spiders X and Y are both 0, they both take the path starting with site s_0, which leads to the output location without any functional sites. In this case, whichever spider reaching the output location has value 0, reporting the result of $0 \wedge 0$ is 0. When spider $X = 0$ and spider $Y = 1$, spider Y takes the path starting with site s_1, and gets stuck at the crossroad because site s_p is “off”. Spider X takes the path starting with site

s_0, and will eventually reach the output location, reporting the result of $0 \wedge 1$ is 0. When spider $X = 1$ and spider $Y = 0$, spider X gets to the crossroad via the path starting with site s_1, and gets trapped at the crossroad due to the sites s_t and s_u placed on that path. Spider Y is the only spider that can reach the output location in this case, reporting the result of $1 \wedge 0$ is 0. When both spiders are 1, they meet at the crossroad. Site s_p is turned on by the signal sent from site s_u, so spider Y can take the central path leading to the output location. Since spider X is trapped at the crossroad, only spider Y can reach the output location, reporting the result of $1 \wedge 1$ is 1.

Following a similar design, the layout of the OR gate is shown in Figure 3. When both spiders are 0, they meet at the crossroad. Spider X is trapped on sites s_t and s_u, and spider Y takes the unblocked central path to the output location, reporting the result of $0 \vee 0$ is 0. Under other input assignments, the 0-valued spider takes the path to the crossroad and gets stuck there, only the 1-valued spider can reach the output location, reporting that the result of $1 \vee 0$, $0 \vee 1$, and $1 \vee 1$ is 1.

The movement of the spiders can be modeled as a continuous-time Markov process. We used a kinetic Monte Carlo algorithm to simulate gate computations. For each gate, under different assignments, we investigate the computation time using $10,000$ iterations in each simulation. We assume the transition rate (the rate that a spider limb transits from one site to another) of each spiders is 1. Simulation results for the AND gate and OR gate are shown in Figure 4. In the AND gate or OR gate, under a certain input assignment, the computation time follows a long-tailed distribution because spiders move stochastically. The computation time is the time spent on traversing the path taken by the reporting spider that reaches the output location; it is influenced by factors such as the transition rate or the length of the path. These factors have been discussed in previous work [10, 11], so we do not focus on them in this paper.

3.3 Design of the NOT gate

We use five types of functional sites in the NOT gate design. As is shown in the layout of the NOT gate in Figure 5, site s_t which can trap a spider that attaches to it is placed on the output location. Sites $s_{1\to 0}, s_r^I, s_r^{II}$ and sites $s_{0\to 1}, s_r^I, s_r^{II}$ form two different *switch* mechanisms $SW_{1\to 0}$ and $SW_{0\to 1}$ that are laid on two separate paths. The NOT gate has one input spider which is initially placed at a junction as the input location.

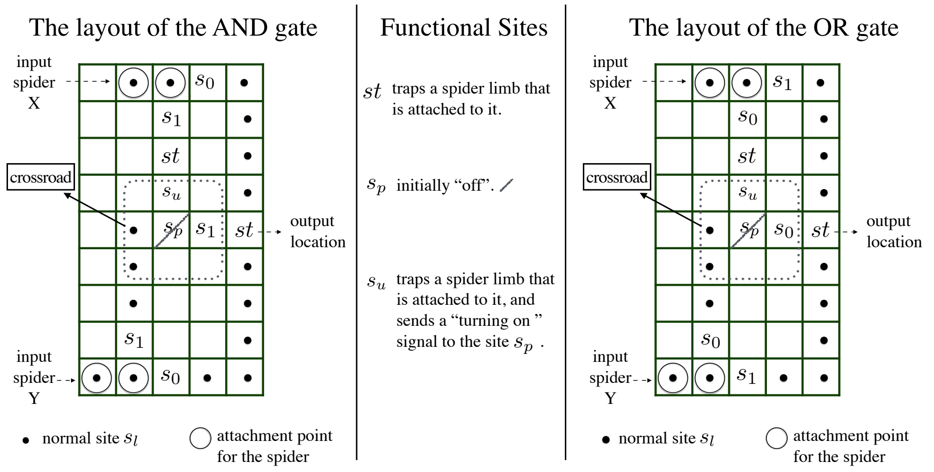
The layout of the AND gate
input spider X
crossroad
output location
input spider Y
normal site s_l
attachment point for the spider
Functional Sites
st traps a spider limb that is attached to it.
s_p initially "off".
s_u traps a spider limb that is attached to it, and sends a "turning on" signal to the site s_p.
The layout of the OR gate
s_0
s_1
st
s_u
s_p

Fig. 3. The layout of the AND gate and OR gate. Three functional sites s_t, s_p, and s_u used in the designs of these two gates are listed in the middle column. Normal site s_1 can only bind to an 1-valued spider and normal site s_0 can only bind to a 0-valued spider. In the AND gate, when both spiders are 1, they meet at the crossroad in the middle. Spider X gets trapped at sites s_t and s_u, site s_u sends a "turning-on" signal to unblock site s_p, allowing spider $Y = 1$ to take the unblocked central path from site s_p to the output location . Under other input assignments, the 1-valued spider gets stuck at the crossroad, so only the 0-valued spider can reach the output location. Therefore, the AND gate yields 1 when both spiders are assigned 1, and yields 0 in all other cases. Similarly, in the OR gate, when both spiders are 0, they meet at the crossroad in the middle and only spider $Y = 0$ can reach the output location. Under other input assignments, the 0-valued spider gets stuck at the crossroad, so only the 1-valued spider can reach the output location. Therefore, the OR gate yields 0 when both spiders are assigned 0, and yields 1 in all other cases.

Two separate paths branch out from the junction: one is taken by the 1-valued spider and contains mechanism $SW_{1\to 0}$ that can change the spider value to 0, the other is taken by the 0-valued spider and contains mechanism $SW_{0\to 1}$ that can change the spider value to 1. When a spider moves through a *switch* mechanism, its value is switched and its backward route is cut off. We explain how mechanism $SW_{1\to 0}$ works with a 1-valued spider as an example; mechanism $SW_{0\to 1}$ works analogously.

Mechanism $SW_{1\to 0}$ is formed by three neighboring functional sites along the horizontal direction: $s_{1\to 0}, s_r^I, s_r^{II}$. We use a staging transition diagram in Figure 5 to describe how mechanism $SW_{1\to 0}$ changes a 1-valued spider to be 0, and cuts off the backward route of the spider. A stage transition shows the change of the spider's location, value or the

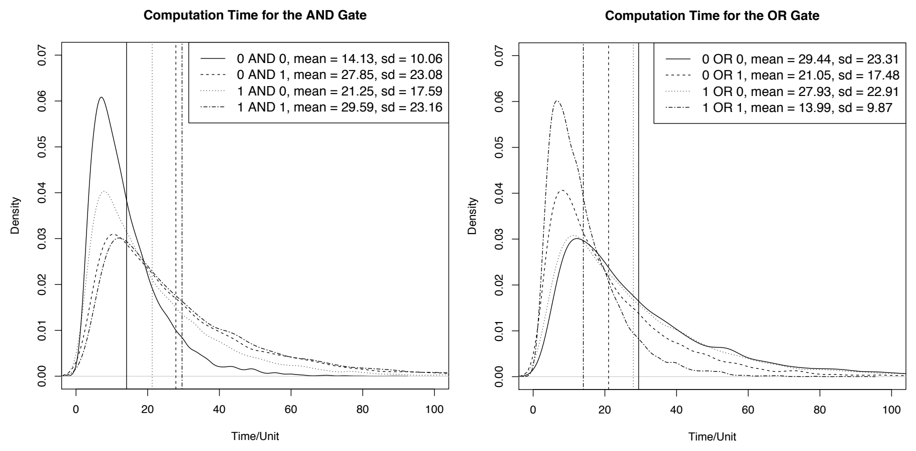

Fig. 4. The computation time distributions for the AND gate and the OR gate under four possible input assignments. Each curve in one gate represents a time distribution under one assignment. The vertical line indicates the mean value of computation time under one assignment in the simulation. The standard deviation for each curve is shown in the legend.

site states. At stage (1), all sites are "on" initially. Site $s_{1\to0}$ can trap a spider, and contains a "switching to 0" signal that will be sent to its left site when a spider attaches to it. Therefore, when a 1-valued spider attaches to $s_{1\to0}$, it is trapped and receives the signal changing its value to 0, causing a transition to stage (2). At stage (2), since the limb trapped at site $s_{1\to0}$ cannot move back, the spider could only move forward by attaching to site s_r^I that traps the spider and sends out a "turning off" signal to its left site. When site $s_{1\to0}$ receives that signal and turns itself "off", we get to stage (3). At stage (3), the limb trapped on s_r^I cannot move back, the spider could only move forward by attaching to site s_r^{II} that sends a "turning off" signal to its left site. When s_r^I receives that signal and turns itself "off", we get to stage (4). At stage (4), the limb on s_r^I can transit to a normal site on the right of s_r^{II}, while the limb on s_r^{II} cannot move back to $s_{1\to0}$ which is "off". The spider could only move forward to get to stage (5). At stage (5), sites s_r^I and s_r^{II} are "off", the spider cannot walk back. When a spider goes through these 5 stages, its value is switched and its backward route is cut off. The mechanism $SW_{0\to1}$ comprising $s_{0\to1}, s_r^I, s_r^{II}$ follows similar staging transitions, the only difference being that a 0-valued spider becomes 1 in the stage transition (1) to (2).

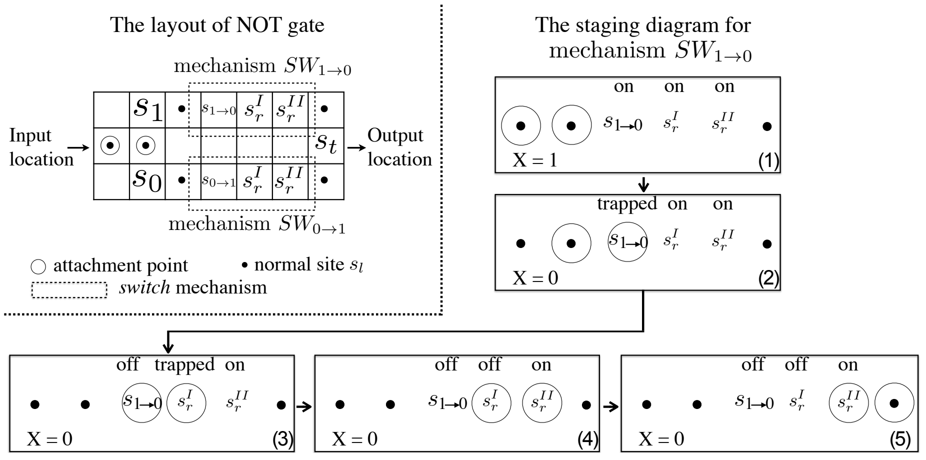

Fig. 5. The layout of gate NOT is shown in the figure. The function of mechanism $SW_{1\to0}$ is to switch a spider's value from 1 to 0 and cuts off its backward route. We show how mechanism $SW_{1\to0}$ works in a staging transition diagram, where the spider value is expressed as X and the state of each functional site is shown above it.

Figure 6 shows the computation time distributions for the NOT gate. The distribution curves for the two input assignments are long-tailed and alike, which is due to the symmetric path design for the 1-valued spider and the 0-valued spider.

3.4 Gate Cascades

To construct a large logic circuit, we need to cascade logic gates of the three kinds defined in Section 3.2 and Section 3.3. A wire w connecting an upstream gate and a downstream gate is composed of continuous normal sites s_l. To ensure that the spider that reaches the output location exits the upstream gate and never goes back to it, we place two additional sites s_r^I and s_r^{II} after site s_t on the output location, forming an *exit* mechanism which cuts off the backward route of a spider that moves through it.

The mechanism *exit* follows similar staging transitions to mechanism $SW_{1\to0}$ shown in Figure 5. It consists of three neighboring functional sites along the horizontal direction: s_t, s_r^I, s_r^{II}. We explained the functionality of site s_r^I and s_r^{II} at the end of Section 3.3. Site s_t is designed to trap the spider. Therefore, a staging transition diagram for mechanism *exit* is similar to the one shown in Figure 5, with the only difference that the spider value is unchanged throughout the five stages. For a downstream gate

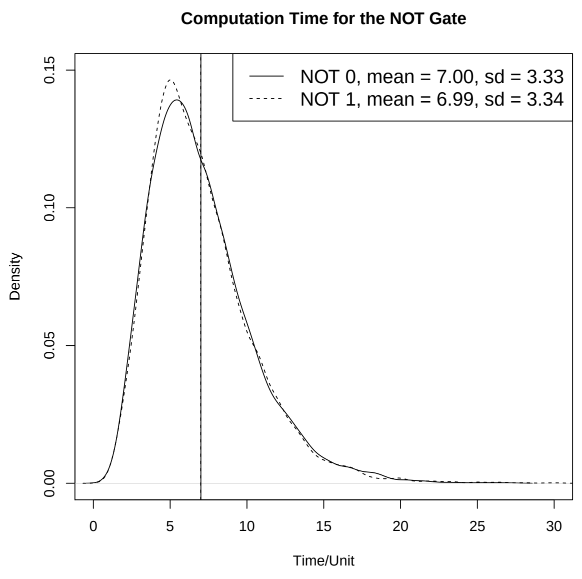

Fig. 6. The computation time distributions for the NOT gate under two possible input assignments.

with two inputs, its two input spiders may arrive at different moments. Computation of the downstream gate begins when either input spider enters the gate, and the asynchronous arrival of input spiders will not influence the computation accuracy of the gate.

Figure 7 illustrates a simple logic circuit implemented by cascading two NOT gates as the inputs to an AND gate. The output location of each NOT gate is connected to an input location of the AND gate via the *exit* mechanism. Spider X and spider Y start to move in the two NOT gates concurrently. When the two spiders move out of the NOT gate, their backward routes are cut off due to the *exit* mechanisms, and they have their values changed to $\neg X$ and $\neg Y$. When either spider enters the AND gate, gate computation begins, yielding the result $\neg X \wedge \neg Y$ eventually. The computation time of this logic circuit is shown in Figure 8. In all simulation runs, the output spider produced the correct output value.

3.5 Complexity Analysis

In a single gate, the computation time t_{gate} is the traversal time of the spider that reaches the output location. Since the spider moves on the track

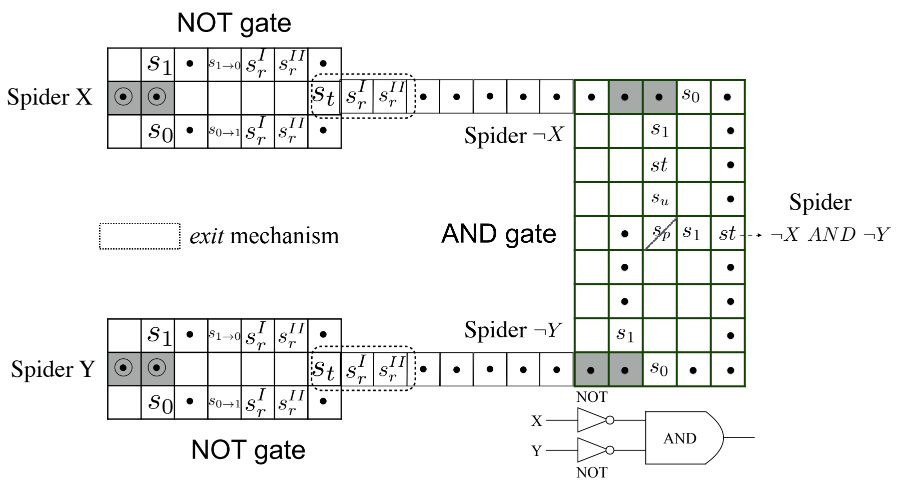

Fig. 7. A logic circuit: $(\neg X \wedge \neg Y)$. The input locations of each gate are highlighted in grey. Spiders X and Y exit the NOT gate, becoming spider $\neg X$ and $\neg Y$ after passing through the *exit* mechanisms. The AND gate computation begins whenever a spider enters the AND gate. The spider reaching the output location of the AND gate represents the computation result $\neg X \wedge \neg Y$.

stochastically, the computation time t_{gate} is a random variable following a long-tailed distribution, as shown in Figure 4 and Figure 6.

When a spider leaves a gate or enters a gate, its backward route is cut off due to the functionality of the *exit* mechanism, so we can use the computation time of a single gate t_{gate} to estimate the computation time t of a circuit. For any n-variable boolean function, we can transform it into 3-CNF, which is a conjunction of m clauses, each a disjunction of at most three literals. Since our design allows parallel evaluation, for a clause $m_i = (l_1^i \vee l_2^i \vee l_3^i)$, the computation time of m_i is

$$t_{m_i} \leq 2 \times (t_{\text{OR}} + t_{\text{NOT}}) = O(1).$$

Since each clause needs time t_{m_i}, to evaluate m clauses in parallel, we conduct $\log m$ AND gate computations that cost $t_{\text{AND}} \times \log m$, and in total use time

$$t = t_{\text{AND}} \times \log m + t_{m_i} = O(\log m).$$

For any boolean function in 3-CNF with m clauses, we use at most $3m$ spiders to represent the literals. For each clause, we need at most three NOT gates and two OR gates if all the literals are the negation of a variable, which is a constant number. For m clauses, we need $m - 1$ AND

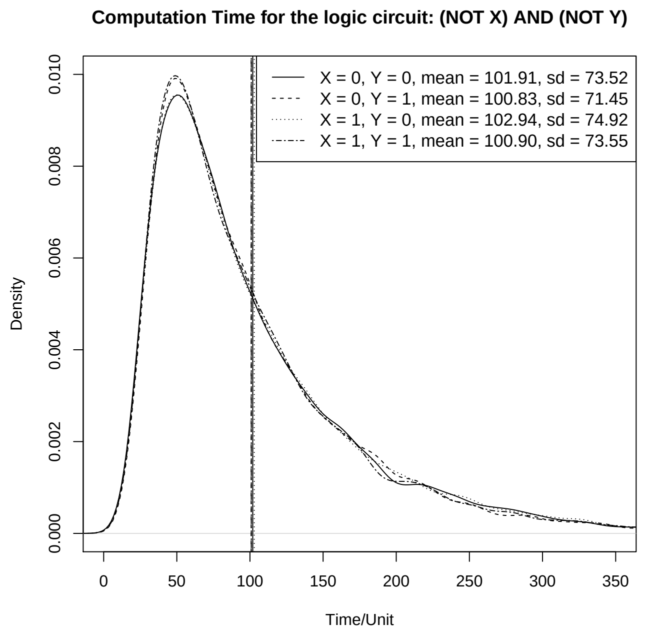

Fig. 8. The computation time distribution for the logic circuit ($\neg X \wedge \neg Y$) under four possible input assignments.

gates. Therefore, the total space complexity is $O(m)$. Hence, our circuit designs are scalable because circuit size in our design scales linearly with formula size, and evaluation time is logarithmic in the formula size.

4 Formal Definition of the Model

The active molecular spider system is modeled as a continuous-time Markov process where the state transitions depend on the interactions between the molecular spiders and the sites on the track. We first define the site types and transition rules of alterable sites, and then give a formal definition of the model.

4.1 Site Types and Transition Rules

Sites are categorized into *normal sites* and *functional sites*. A normal site $s \in S_{norm} = \{s_l, s_0, s_1\}$ has no state. Site s_l binds to the spider's leg. Sites s_0 and s_1 bind to the spider's arm if it has type 0 or 1, respectively.

A functional site $s \in S_{fun}$ has a state of "on", "off", or "trapped". The site state transition diagram is:

$$\begin{tikzcd} on \arrow[d] & \\ trapped \arrow[r] & off \arrow[ul] \end{tikzcd}$$

A spider limb can only attach to an "on"-state site. An "off"-state site is non-alterable. The limb trapped on a "trapped"-state site cannot leave the site by itself. Whether a site can trap a spider is indicated by $TR \in \{0,1\}$: a site with $TR = 1$ will trap a spider when a limb attaches to it. A functional site may change the spider's value, or the state of another site, by sending out a *signal* to the spider or another site. We define

$$signal = (val, d) \text{ or } null, \text{where } d \in \mathbb{Z}^2 \text{ and } val \in \{\text{on, off, trapped}, 1, 0\}. \quad (1)$$

Suppose a functional site is located at (x, y). If it holds a $signal = (val, d = (d_x, d_y))$ then it sends the signal to the location $(x + d_x, y + d_y)$, setting the state of the site located there, or the spider's value, to val. When $d = (0, 0)$, the val field of the signal is either 1 or 0, which is sent to the spider, setting the spider's value to 1 or 0.

Therefore, we can define a functional site $s \in S_{fun}$ as

$$s = (state, TR, signal). \quad (2)$$

Table 1. Definition of different functional sites used in the circuit construction and the *transition rules* applied to them. Suppose the location of the site is (x, y), define $(x', y') = (x + d_x, y + d_y)$.

Transition Rules		
functional site	updated site	other changes
$s_t = (\text{on}, 1, null)$	$s'_t = (\text{trapped}, 1, null)$	
$s_{1\to 0} = (\text{on}, 1, (0, (0, 0)))$	$s'_{1\to 0} = (\text{trapped}, 1, null)$	$A = 0$
$s_{0\to 1} = (\text{on}, 1, (1, (0, 0)))$	$s'_{0\to 1} = (\text{trapped}, 1, null)$	$A = 1$
$s^I_r = (\text{on}, 1, (\text{off}, d))$	$s^{I'}_r = (\text{trapped}, 1, null)$	site at (x', y') becomes off
$s^{II}_r = (\text{on}, 0, (\text{off}, d))$	$s^{II'}_r = (\text{on}, 0, null) = s_l$	site at (x', y') becomes off
$s_u = (\text{on}, 0, (\text{on}, d))$	$s'_u = (\text{on}, 0, null) = s_l$	site at (x', y') becomes on
$s_p = (\text{off}, 0, null)$	$s'_p = (\text{on}, 0, null) = s_l$ when a "turning-on" signal is received	

The signal held in a site is sent out once a spider limb attaches to the site. When a signal is sent out, the site has no signal remaining, which we express as $s = (state, TR, null)$. A functional site $s = (on, null)$ is equivalent to a normal site, which is non-alterable. Once a signal is received by a site or a spider, the site state or the spider's value is changed according to the signal.

In the logic circuit construction, we use two functional sites s_u and s_p in the AND gate and OR gate, and we design a set of functional sites that form different mechanisms in the NOT gate and the gate cascades. Table 1 gives the definitions of these functional sites and the *transition rules* applied to them. A functional site s transits to site s' in the second column, either by receiving a signal or being attached by a spider limb. If s holds a signal, it causes other changes in the last column. In table 1, the updated site s' in the second column is either a normal site or a trapped site. According to the site state transition diagram, a trapped site can only transit to a "off"-state site that is non-alterable by itself. Since no signals are designed to turn on these "off"-state sites transited from the trapped sites, these "off"-state sites are non-alterable finally. Therefore, all the functional sites in Table 1 are alterable initially and become non-alterable finally. The functional sites used in our design are

$$\{s_t, s_{1\to 0}, s_{0\to 1}, s^I_r, s^{II}_r, s_u, s_p\},$$

where each site s among them includes its site transitions under the *transition rules* described in Table 1. The set of site types is $S = S_{norm} \cup S_{fun}$.

A *mechanism* is a set of neighboring mechanism sites along the same direction. We design three different mechanisms used in the logic circuit construction. The *switch* mechanism $SW_{1\to 0}$ ($SW_0 \to 1$) contains sites $s_{1\to 0}(s_{0\to 1}), s_r^I, s_r^{II}$, where site s_r^I, s_r^{II} contains the signal of $(\text{off}, (-1,0))$ which can block its left site. When a spider moves over the *switch* mechanism, its value is flipped, and its backward route is cut off. The *exit* mechanism contains sites s_t, s_r^I, s_r^{II}. When a spider moves over this mechanism, its backward route is cut off.

When a spider limb leaves a site, this limb can reach 4 sites geometrically (shown in figure 1). Since sites have different types, wether a site is available for a limb of a spider depends on the spider value and the site types. Given a spider with value A and a site, algorithm *check* summarizes how to tell if the site is available.

Algorithm 1 Algorithm *check* tells if a given site is available.

```
• if the site is occupied by another spider, it is not available.
• else:
    1. if the site is a normal site:
        (a) if the site is $s_l$, it is available;
        (b) if the site is $s_1$ and $A = 1$, it is available;
        (c) if the site is $s_0$ and $A = 0$, it is available;
    2. else if the site is a functional site:
        (a) if the site is $s_{1\to 0}$ and $A = 1$, it is available;
        (b) if the site is $s_{0\to 1}$ and $A = 0$, it is available;
        (c) if the site is "on"-state, it is available;
    3. else, the site is not available.
```

Using Algorithm 1, we examine every site among the 4 sites shown in figure 1, putting those available into a set $\mathcal{AV}$.

4.2 Model Definition

The *active* multi-spider system with normal sites and alterable sites can be modeled as a continuous-time Markov process. We define the state of the model as

$$X = (S_1, S_2, \ldots, S_n, E), \tag{3}$$

where $S_i = (P_i, A_i)$ $(1 \le i \le n)$ describes the state of the i-th spider. Set $P_i = (p_a^i, p_b^i)$ contains attachment points for the i-th spider, and $A_i \in \{0, 1\}$ represents the Boolean value of the spider. The lattice configuration $E : \mathbb{Z}^2 \to S$ shows the layout of different sites, where S is the set of site types. Normal sites can be regarded as having state "on", $TR = 0$ and no signal, so we can redefine the lattice configuration as

$$E : \mathbb{Z}^2 \to \{\text{on, off, trapped}\} \times \{1, 0\} \times \mathbb{S},$$

where $\mathbb{S}$ represents the set of signals.

Given a model state $X = (S_1, S_2, \ldots, S_n, E)$ at time t, if a limb leaves an attachment point $p \in P_i \in S_i$, we use the algorithm *check* to obtain a set of available sites $\mathcal{AV}$. At time $t + \delta$, this limb transits to $p' \in \mathcal{AV}$, changing the set of attachment points to $P_i' = P_i - \{p\} \cup \{p'\}$. We use the *transition rules* to update A_i, so we have $S_i' = (P_i', A_i')$. The transition rules also updates E, thus the new state is

$$X' = (S_1, S_2, \ldots, S_{i-1}, S_i', S_{i+1}, \ldots, S_n, E').$$

5 Conclusions and Discussions

Using an *active* multi-spider model with spider cooperation and localized signal transmission, we have implemented the basic logic gates (AND, OR, NOT). We have shown how to implement gate cascades, in which each upstream gate G_u is connected to a downstream gate G_d using the *exit* mechanism. We use $O(1)$ types of spiders and sites. To evaluate an n-variable Boolean function that is in 3-CNF with m clauses, the evaluation time is $O(\log m)$ and the size of the circuit is $O(m)$. Therefore, our design supports scalable computation and ensures spatial locality. Molecular circuits with spatial locality overcome the challenges of computation speed-up and sequence reuse in molecular computing in a well-mixed environment, but there are still other issues. Compared with previous work [2,3,5], our design better addresses the following issues:

Geometrical layout. Molecular circuits with spatial locality arrange different computing components on a 2D plane where the distance between different components should be set carefully to avoid interference across components. Reducing the number of gates used in a circuit can ease the geometrical layout problem. Our design implements a NOT gate to avoid dual-rail logic conversion used in previous work [2, 5], which simplifies the circuit and its layout. Compared with the circuit [3] in a form of BDD where the layout of different branching paths requires appropriate angles and lengths, our design only considers connections between gates because each gate has a fixed layout.

Data encoding. In previous work, variable representation is encoded into the circuit [2, 3, 5], so each variable corresponds to a distinct sequence. This complicates sequence design if the circuit has a large number of variables. Our design separates variable representation from circuit design, only using two types of spiders placed at different input locations to represent all variables.

Circuit reusability. Tethered circuits [2,5] use irreversible local signal transmission to implement logic computation and value propagation, so the circuit is not reusable. The circuit [3] adds external strands to unblock a path for an evaluating walker. This procedure irreversibly changes the circuit configuration, thus the circuit is not reusable. In our design, irreversible local signal transmission is used to control the spiders' behavior at a few locations in the circuit, which only occupy a small portion of computation. Since non-alterable sites form the majority of the circuit, most parts of the circuit are reusable.

We lack an experimental implementation of our designs, thus here we use a simulator that simulates the circuit at the site level, assuming spiders have equal transition rates to all sites. We are working on an implementation where normal sites are short DNA strands so that molecular spiders can attach to or detach from the normal sites freely, and functional sites transmit signals to neighboring sites via strand displacement. For example, we can encode a signal in the loop (inactive part) of a hairpin structure. Once a spider attaches to the hairpin structure, the loop is opened so that the exposed domain can react with other neighboring sites, transmitting the signal encoded in the opened loop to other neighboring sites. In the future, we will complete a plausible implementation and focus on a simulator that can better reflect how different sites react with spiders according to that implementation.

Since spiders move bidirectionally on the track, we can use this feature to solve some interesting problems. For example, it may be possible to construct a feedback loop that can be used to solve a SAT problem automatically where the spider that does not satisfy the formula can go back to switch its value. In the current model, molecular spiders can probe, walk, and change their own states and the state of the environment. These behaviors of the molecular spiders can be extended for complex intracellular tasks, e.g., we can use the molecular spiders to replace natural motors. In the future, we will explore applications of our design, as well as the possibility of implementing it in the laboratory.

Acknowledgments. This material is based upon work supported by the National Science Foundation under grants 1318833 and 1422840.

References

1. T. Antal and P. L. Krapivsky. Molecular spiders with memory. *Physical Review E*, 76(2):021121, 2007.

2. H. Chandran, N. Gopalkrishnan, A. Phillips, and J. Reif. Localized hybridization circuits. In *DNA Computing and Molecular Programming*, pages 64–83. Springer, 2011.
3. F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A. Turberfield. DNA walker circuits: Computational potential, design, and verification. In D. Soloveichik and B. Yurke, editors, *DNA Computing and Molecular Programming*, volume 8141 of *Lecture Notes in Computer Science*, pages 31–45. Springer International Publishing, 2013.
4. K. Lund, A. J. Manzo, N. Dabby, N. Michelotti, A. Johnson-Buck, J. Nangreave, S. Taylor, R. Pei, M. N. Stojanovic, N. G. Walter, et al. Molecular robots guided by prescriptive landscapes. *Nature*, 465(7295):206–210, 2010.
5. R. A. Muscat, K. Strauss, L. Ceze, and G. Seelig. DNA-based molecular architecture with spatially localized components. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 177–188. ACM, 2013.
6. J. E. Padilla, W. Liu, and N. C. Seeman. Hierarchical self assembly of patterns from the Robinson tilings: DNA tile design in an enhanced tile assembly model. *Natural Computing*, 11(2):323–338, 2012.
7. R. Pei, S. K. Taylor, D. Stefanovic, S. Rudchenko, T. E. Mitchell, and M. N. Stojanovic. Behavior of polycatalytic assemblies in a substrate-displaying matrix. *Journal of the American Chemical Society*, 128(39):12693–12699, 2006.
8. L. Qian and E. Winfree. Scaling up digital circuit computation with dna strand displacement cascades. *Science*, 332(6034):1196–1201, 2011.
9. L. Samii, G. A. Blab, E. H. C. Bromley, H. Linke, P. M. G. Curmi, M. J. Zuckermann, and N. R. Forde. Time-dependent motor properties of multipedal molecular spiders. *Physical Review E*, 84:031111, Sep 2011.
10. O. Semenov, D. Mohr, and D. Stefanovic. First-passage properties of molecular spiders. *Physical Review E*, 88(1):012724, 2013.
11. O. Semenov, M. J. Olah, and D. Stefanovic. Mechanism of diffusive transport in molecular spider models. *Physical Review E*, 83(2):021117, 2011.
12. O. Semenov, M. J. Olah, and D. Stefanovic. Multiple molecular spiders with a single localized source-the one-dimensional case. In *DNA Computing and Molecular Programming*, pages 204–216. Springer, 2011.
13. S. F. Wickham, J. Bath, Y. Katsuda, M. Endo, K. Hidaka, H. Sugiyama, and A. J. Turberfield. A DNA-based molecular motor that can navigate a network of tracks. *Nature Nanotechnology*, 7(3):169–173, 2012.