Int. Journ. of Unconventional Computing, Vol. 8, pp. 459–507
Reprints available directly from the publisher
Photocopying permitted by license only

©2013 Old City Publishing, Inc.
Published by license under the OCP Science imprint,
a member of the Old City Publishing Group.

Local Parallel Biomolecular Computation*

JOHN H. REIF

Department of Computer Science, Duke University, Durham, NC,
USA and Adjunct, King Abdulaziz University (KAU), Jeddah, Saudi Arabia
E-mail: reif@cs.duke.edu

Received: December 9, 2012. Accepted: December 10, 2012.

Biomolecular Computation (BMC) is computation at the molecular scale, using biotechnology engineering techniques. Most proposed methods for BMC used *distributed (molecular) parallelism (DP)*; where operations are executed in parallel on large numbers of distinct molecules. BMC done exclusively by DP requires that the computation execute sequentially within any given molecule (though done in parallel for multiple molecules). In contrast, *local parallelism (LP)* allows operations to be executed in parallel on each given molecule.

Winfree, *et al.* [120, 124]) proposed an innovative method for LP-BMC, that of computation by *unmediated self-assembly* of $2D$ arrays of DNA molecules, applying known domino tiling techniques (see Buchi [21], Berger [16], Robinson [88], and Lewis and Papadimitriou [50]) in combination with the DNA self-assembly techniques of Seeman *et al.* [105].

We develop improved techniques to more fully exploit the potential power of LP-BMC. we propose a refined *step-wise assembly* method, which provides control of the assembly in distinct steps. Step-wise assembly may increase the likelihood of success of assembly, decrese the number of tiles required, and provide additional control of the assembly process. The *assembly depth* is the number of stages of assembly required and the *assembly size* is the number of tiles required.

We also introduce the *assembly frame*, a rigid nanostructure which binds the input DNA strands in place on its boundaries and constrains the shape of the assembly. Our main results are LP-BMC algorithms for some fundamental problems that form the basis of many parallel computations. For these problems we decrease the assembly size to linear in the input size and and significantly decrease the assembly

*A preliminary version of this paper appeared in Proc. DNA-Based Computers, III: University of Pennsylvania, June 23-26, 1997. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, H. Rubin and D. H. Wood, editors. American Mathematical Society, Providence, RI, vol. 48, 1999, pp. 217–254.

depth. We give LP-BMC algorithms with linear assembly size and logarithmic assembly depth, for the parallel prefix computation problems, which include integer addition, subtraction, multiplication by a constant number, finite state automata simulation, and fingerprinting (hashing) a string. We also give LP-BMC methods for perfect shuffle and pair-wise exchange using a linear size assembly and constant assembly depth. This provides logarithmic assembly depth LP-BMC algorithms for the large class of normal parallel algorithms [49, 112, 113] on shuffle-exchange networks, e.g. DFT, bitonic merge, fixed permutation of data, as well as evaluation of a bounded degree Boolean circuit in time bounded by the product of the circuit depth times a logarithm of the circuit size. Our LP-BMC methods may require somewhat smaller volumes than previous DP-BMC algorithms [75, 81] for these problems. All our LP-BMC assembly techniques can be combined with DP-BMC parallelism to simultaneously solve multiple problems with distinct inputs (e.g. do parallel arithmetic on multiple inputs, or determine satisfying inputs of a circuit), so they are an enhancement of the power of DP-BMC.

1 INTRODUCTION

Biomolecular Computation. We will use the term *nanocomputation* to denote computation executed at the molecular scale. See Reif [82] for an extensive survey of BMC. To be effective, nanocomputation will depend on a technology that allows for manipulation of objects at a molecular scale. One such promising technology is that of microbiology, specifically by use of well established recombinant DNA [108, 109, 118] and RNA operations (appropriately modified to insure the required reliability). A key operation is *hybridization* of single-stranded DNA molecules which are Watson-Crick complementary and oriented in opposite directions, into double-stranded DNA in a helical structure. We will use the term *Biomolecular computation (BMC)* to denote nanocomputation using biotechnology. The first implementation of BMC by Adleman [1] solved a small Hamiltonian path problem by use of recombinant DNA technologies.

The fundamental principles for doing BMC need to be motivated by the underlying biotechnology, but since this is a quickly evolving technology, the principles for doing BMC should be sufficiently general so that they will hold for later forms of this biotechnology. This motivates us to develop parallel algorithms for BMC in the context of well defined models, that represent the key aspects and technology constraints of BMC. It is still not clear what the fundamental techniques for doing BMC may be. In particular, the literature in BMC utilizes only a few distinct classes of methods.

Distributed Parallel Biomolecular Computation. In the following, let a *Distributed Parallel (DP) operation* be a single operation done in parallel on

multiple distinct molecules (for example the use of restriction enzymes to do editing of these molecules, each at the corresponding same labeled site). This parallel execution by DP on multiple molecules to execute computations will be termed here *Distributed Parallel BMC (DP-BMC)*. DP-BMC provides a potential for massive parallel computation due to the vast numbers of molecules in a test tube.

BMCs [108] have utilized certain *base methods* which include:

- generation of a large number of sequences by random assembly; see Adleman [2] and Lipton [63] (interestingly, this is the only method now in use in BMC that could be viewed as a combined DP-BMC and LP-BMC method) and
- DP-BMC methods for string editing operations, executed sequentially within each molecule.

Many known proposed methods for BMC then use a combination of these base methods and DP-BMC. They can be categorized as follows:

- simulation of sequential computations (e.g. execution of bit serial arithmetic or more generally universal Turing Machines), and
- solution of NP search problems, by the use of DP-BMC processing.

Note that the papers [2, 3, 8, 12, 17, 63, 108] propose to solve NP search problems in essentially the same manner, as follows:

1. a *generation phase* for assembling all possible solutions,
2. *verification* of the validity of each possible solution; this is done in parallel by DP-BMC for each possible solution, but is done sequentially for each solution, and
3. *detection* of a valid solution.

In many cases, the potential speed-up benefits from DP-BMC are somewhat limited because each circuit or formula defining the search problem is locally evaluated sequentially, and so such DP-BMC methods for BMC require many steps to do this evaluation locally. This presents a possible disadvantage, since the step duration in BMC is much longer than the step duration for conventional VLSI, but this disadvantage may be overcome by the use of massive parallelism due to DP-BMC.

For example, consider the recent stickers project of Adleman *et al.* [3] (see also an earlier proposal of [17]) for breaking the DES standard cryptosystem via BMC. Their proposed method for breaking DES makes use of DP-BMC to evaluate in parallel the DES circuit for all possible key values.

Their method uses at least as many operations as required by the sequential evaluation of the DES circuit, which consists of a few thousand Boolean gates. In contrast, a conventional VLSI circuit for DES takes time which is the depth of the DES circuit, which can be done in conventional VLSI at a very high step rate, in just a few dozen parallel steps. [3] still argue, quite convincingly, that the massive parallelism due to DP-BMC may allow them a sufficient advantage over conventional computers to allow them to break DES. Nevertheless, this provides a motive for us to attempt to also exploit DP-BMC where possible to improve the advantages of BMC, even in the case where the more obvious major advantage is due to DP.

Circuits can be evaluated in parallel by use of DP-BMC algorithms[1] [75, 81] requiring a large volume; in contrast our LP-BMC methods require a considerably smaller volume.

Local Parallel Biomolecular Computation. Let a *Local Parallel (LP) operation* be a parallel operation done on a given molecule (for example the editing of a single molecule at multiple sites). For large molecules, such as long DNA or RNA strands, LP-BMC is the natural analog to the parallelism as provided by conventional VLSI for parallel circuit evaluation.

Another motivation for developing LP-BMC methods is to allow us to efficiently work with large biological DNA strands. In contrast, most methods proposed for BMC deal with relatively short DNA and RNA strands, due to the sequential processing within individual strands. However, many biological applications, for example DNA sequencing, involve large DNA strands. LP-BMC methods may allow for the efficient processing for such large DNA and RNA strands.

LP-BMC via Unmediated Self-assembly. Domino tiling techniques were developed by Buchi [21], Berger [16], Robinson [88], and Lewis and Papadimitriou [50] in the early 1960s, and allow simulation of a one-tape Turing Machine with $S \geq n$ space (or a $1D$ parallel array of S processors) running in time $T \geq n$ by construction of an assembly of an $S/2 \times T$ array of $A = ST/2 \geq n^2/2$ tiles with assembly depth T. Winfree [120] (also Winfree, *et al.* [124]), proposed to execute these domino tiling techniques in LP-BMC by assembling $2D$ arrays of DNA molecules, using the DNA self-

[1]Reif [81] gave first known polylog DP-BMC algorithm for the parallel evaluation Boolean circuits; if a circuit is constructable in s space with unbounded fan-out, and depth d, then that algorithm had time cost $O(d + s \log s)$ for circuit evaluation in a PAM model for DP-BMC where long complementary matching is allowed (with a factor s further cost for a more restricted RDNA model where only constant length matches are allowed). Recently Ogihara and Ray [75] gave the similar bounds for circuit evaluation assuming implicitly a similar model for DP-BMC where long complementary matching is allowed (but do not consider more refined models such as the RDNA where only constant length matches are allowed). Both these results required large amount of communication between distinct molecules and thus require large volume growing at least as 2^{2s}.

assembly techniques of Seeman *et al.* [105]. Winfree proposed the computation be done by *unmediated self-assembly*, that is the assembly of tiles proceeds without mediation (i.e., external control). While the idea of computation via unmediated self-assembly is very appealing, an early version of this paper expressed skepticism about unmediated self-assembly, it appeared that it was difficult to determine the likelihood for a successful assembly. However, there is now considerable evidence that computation via unmediated self-assembly is feasible. First, computer simulations by Winfree [121] indicated that careful choice of settings of the parameters allow unmediated self-assembly to succeed in kinetic assembly model. Further, the subsequent development of error-resilient tiling systems by Winfree [122], Reif [86] and others, has Then computational tiling assemblies using unmediated self-assembly were first experimentally demonstrated via one dimensional assemblies [59, 67], using designs based on the tiling prefix-computation assemblies described in this paper, in collaberations involving this author's group, Seeman's group at NYU, and Winfree at Caltech. Later work by Winfree's group at Caltech demonstrated two dimensional computational tiling assemblies [90].

The Goals of this Paper. The central task which we investigate in this paper, is to *develop and refine methods for LP-BMC* and to combine these methods with the already known DP-BMC techniques (e.g., as provided naturally by recombinant DNA techniques). Our approach is to utilize certain parallel algorithm design techniques, including parallel prefix computation and parallel shuffle-exchange operations. With the use of some form of global parallel communication between distributed molecules or an associative match operation on sets of molecules, these parallel algorithm design techniques can be applied to BMC (Reif [81]). However, we require in this paper that DP-BMC not utilize global parallel communication between distributed molecules, and in this case application of these parallel algorithm design techniques to LP-BMC is not obvious or straightforward.

It should be emphasized that although our molecular assembly techniques have some experimental basis (due to the extensive DNA self-assembly work of Seeman [105, 124], who has constructed many of the basic components (tiles) of our proposed assemblies, this paper is theoretical in nature and should be evaluated from that perspective.

Our Improvements to LP-BMC:

(1) Step–Wise Assembly. We propose instead a refined assembly method, which we call *step-wise assembly*. This provides control of the assembly of an $S/2 \times T$ array in distinct T steps, to increase the likelihood of success. This simulation is also extended to simulate a $2D$ processor array.

(2) Framing. We also introduce a new method for constraining the assembly, which we call an *assembly frame*, which is a rigid nanostructure whose interior boundary binds the the input DNA strand and constrains the shape of the initial assembly to a given polygonal shape, thus providing for inputs and constraining the geometry of the subsequent tiling assembly.

(3) Compact Assemblies with Small Assembly Depth. Let the *assembly depth* of a tile in the intended tiling be the number of matches between tiles in paths of this breadth first assembly tree. We define the *assembly depth d* of the tiling assembly to be the maximum assembly depth of any tile of the intended assembly; it is the number of stages of assembly required if stepwise assembly is used. The *assembly size* is the number of tiles required.

Using our assembly techniques, we propose LP-BMC methods that have linear assembly size in the input size and have small assembly depthṪhese techniques for LP-BMC can be used for many applications: basic arithmetic and register level operations, data rearrangements and permutations, as well as more sophisticated operations done on large sequences of data. We use LP-BMC to solve the following problems with linear assembly size:

- logarithmic assembly depth *prefix computation:* given a composition operation that is associative and a sequence of n inputs, compute the composition of all prefixes of the inputs (e.g., prefix sums of n numbers; n-bit arithmetic such as addition, and subtraction, multiplication by a constant number; simulation of a finite state automata on an input of length n; fingerprint a length n string),
- constant assembly depth *perfect shuffle and pair-wise exchange permutations*, which allows constant time emulation of the shuffle-exchange network,
- logarithmic assembly depth execution of a large class of *normal parallel algorithms* [49, 112, 113] on shuffle-exchange networks (e.g., DFT and bitonic merge of n-vectors),
- logarithmic assembly depth *general permutations*, by use of known Beneš network techniques [15],
- *evaluation of a bounded degree Boolean circuit* of size n in assembly depth bounded by the circuit depth times $O(\log n)$ (without a large amount of communication between distinct molecules, in contrast with known DP-BMC algorithms [75, 81]).

Using LP-BMC to Enhance DP-BMC. LP parallelism complements the more apparent DP parallelism available in BMC. We feel that BMC needs to develop methodologies that combine the use of LP-BMC and DP-BMC, just as conventional electronic computational systems combine the use of VLSI and more distributed modes of computing. Let a *LPDP* operation be a

LP operation done on multiple distinct molecules, that is via a combination of LP and DP. Let *LPDP-BMC* be the resulting mode of BMC computation combining LP-BMC and DP-BMC. With the use of surface attachments, all the LP-BMC techniques we propose can be executed in parallel by separate molecules in a distributed fashion. Thus they can simultaneously solve multiple problems with distinct inputs. For example, our LP-BMC methods for parallel prefix, addition, and circuit evaluation can be done simultaneously in parallel for all possible inputs using LPDP-BMC, so they can execute parallel arithmetic on multiple inputs, and also determine satisfying inputs of a circuit. Thus our LP-BMC assembly techniques are an enhancement to the power of DP-BMC.

1.1 Organization of this Paper

An introduction to BMC is given in Section 1. Section 2 provides useful DNA notation (Subsection 2.1), preliminary descriptions of previous work on DNA self-assembly techniques (Subsection 2.2) and known Turing Machine simulations using tiling (Subsection 2.3). Section 3 describes Winfree's proposed use of unmediated BMC using DNA self-assembly. Section 4 presents our refined method for BMC via step-wise assembly. We discuss emulation of a $1D$ array using step-wise $2D$ assembly (Subsection 4.1), methods for self-assembly of the tiling assembly (Subsection 4.2) (including nano-construction of of tiles, encoding methods, readout methods after assembly, and rigid framing of the initial assembly), 3-way and 4-way pairing of DNA strands, and extensions to emulations of $2D$ processor arrays (Subsection 4.3). The further Sections provide our main results: the construction of *very compact, linear size assemblies, with progressively small depth*. Section 5 proposes a step-wise local assembly method for prefix computation with linear size and depth and Section 6 describes application of these techniques to integer addition. Section 7 describes step-wise local assembly with linear size and logarithmic depth for parallel prefix computation and integer addition. Section 8 describes methods for emulating shuffle-exchange operations with linear size and constant depth using unmediated assembly, including pair-wise exchange (subsection 8.1), and perfect shuffle operations (subsection 8.2). This allows us to efficiently emulate shuffle-exchange networks (Subsection 8.3), execute normal parallel algorithms (Subsection 8.4), and execute general permutations and parallel evaluation of circuits (Subsection 8.5). Section 9 gives conclusions and acknowledgments.

2 PRELIMINARIES

2.1 DNA notation

We use *ssDNA* to denote single-stranded DNA and use *dsDNA* to denote double-stranded DNA. We denote DNA double crossover molecules as *DX*.

We let $5' - 3'$ denote the normal orientation of a ssDNA strand by its direction from $5'$ to $-3'$.

2.2 Nanofabrication Techniques Using DNA Self-assembly

Feynman [36] proposed nanofabrication of structures of molecular size. Nanotechnology, without use of DNA, is discussed in [26, 69].

Fabrication of nanostructures in DNA using self-assembly was pioneered by Seeman (e.g., see [105]) in the 1990s (also see [23, 72] for other early work in DNA nanotechnology). These ingenious constructions used such basic constructive components as:

- *DNA junctions:* i.e., immobile and partially mobile DNA n-armed branched junctions [102],
- *DNA knots:* e.g., ssDNA knots [32, 73] and Borromean rings [66],
- *DNA crossover molecules:* e.g., DX molecules of Fu and Seeman [38].

Many, but not all, of Seeman's constructions used:

- DX molecules for rigidity (or dsDNA for partial rigidity), and
- construction by hybridization in solution, usually followed by ligation.

Using these self-assembly techniques, Seeman and his students, such as Chen and Wang, nanofabricated in DNA (see [100, 103, 103, 104, 104–107, 126, 127])

- $2D$ *polygons*, including interlinked squares, and
- $3D$ *polyhedra*, including a cube and a truncated octahedron.

The truncated octahedron was made by *solid-support* (e.g., immobilized via surface attachments [110]), to avoid interaction between constructed molecules [126].

Seeman also characterized the structural and topological properties of unusual DNA motifs in solution [103, 104, 107].

The ultimate goal in Seeman's work seems to have been to construct regular arrays to immobilize proteins in regular arrays so as to be able to do crystallography analysis of proteins. However, his work may well be of central importance to the progress of the emerging field of BMC.

2.3 Known Tiling Results

Consider the square regions in the plane defined by horizontal and vertical lines which are integer units apart from the horizontal and vertical axis respectively.

A class of *(domino) tiling problems* were defined by Wang [115] as follows: We are given a finite set of tiles of unit size square tiles each with top

and bottom sides labeled with symbols over a finite alphabet. These labels will be called *pads*. We also specify the initial placement of a specified subset of these tiles, and the borders of the region where tiles must be placed defining the *region of tiling*. The tiling problem is to determine if it is possible to place the tiles, chosen with replacement, so the placed tiles fully cover the specified region of tiling, so that each pair of vertical abutting tiles have identical symbols on their contacting sides. Let the *size* of the tiling assembly be the number of tiles placed.

Wang [115] defined the tiling problem. Buchi [21] proved that given a finite set of tile types, the domino tiling problem is undecidable if the extent of tiling is the positive quadrant of the plane and a single initial tile is required to be at a fixed location. Berger [16] gave a proof that removed the latter condition, thus proving the undecidability of the tiling problem with no initial placement of tiles. Later simplified proofs of Robinson [88], and the text book of Lewis and Papadimitriou [50], (pages 296–300) provide a direct simulation of a single tape deterministic Turing Machine to prove this undecidablility result.

Garey, Johnson, and Papadimitriou [40] (see citation of [39], page 257) proved that the domino tiling problem is NP-complete if the extent of tiling is a rectangle of polynomial size. They used a direct reduction from the directed Hamiltonian path problem (known to be NP-complete). Later, the textbook of Lewis and Papadimitriou [50](pages 345–348) gave a direct proof of this NP-completeness result, providing a simulation of a single tape nondeterministic Turing Machine running in time $T \geq n$ and space $S \leq n$ by assembly of an $S/2 \times T$ array of tiles. Grunbaum, Branko, and Shepard [42] surveyed these and related results on the complexity of tiling, including these direct simulations of Turing Machines.

3 BMC USING UNMEDIATED SELF-ASSEMBLY

Winfree [120] defined a *cellular automaton* as a Turing Machine with a single tape of S cells such that disjoint pairs of adjacent cells are updated in parallel by a transition function depending on those cells and the state of the machine. The disjoint pairs of cells which are updated are at consecutive positions that are either odd, even or even, odd (these two cases alternate each step). He observed that the tiling constructions cited above ([50](pages 345–348), [42]) also give the direct simulation of a cellular automaton running in time $T \geq n$ and space S by assembly of an $S/2 \times T$ array of tiles with T assembly depth.

Winfree [120] then proposed a very intriguing idea: to do these tiling constructions by application of the DNA nanofabrication techniques of

Seeman *et al.* [105], which may be used for the self-assembly of small DNA molecules that can function as square tiles with pads on the sides. The pads are ssDNA (pairs of complementary pads are denoted *sticky ends* in some literature). If two pads are Watson-Crick complementary and $5' - 3'$ oriented in opposite directions, we consider them to be *matching*. At the appropriate conditions (determined by temperature and salinity, etc.), they may hybridize into doubly stranded DNA. The assembly of the tiles is due to this hybridization of pairs of matching pads on the sides of the tiles. We will call this idea *BMC via unmediated DNA self-assembly*, since the computations advance with no intervention by any controllers. The advantages of the unmediated DNA assembly idea of Winfree, *et al.*, are potentially very significant for BMC: the computations advance with no intervention by any controllers, and require no thermal cycling.

Winfree, *et al.* [124] then provided further elaboration of this idea, to solve a variety of computational problems using unmediated DNA self-assembly. For example, they propose the use of these unmediated DNA assembly techniques to directly solve the NP-complete directed Hamiltonian path problem, using a construction similar to that of Garey, Johnson, and Papadimitriou's [40] (see also [39], page 257) NP-completeness proof for tiling of polynomial size extent. Winfree, *et al.* [124], also provided a very valuable experimental test validating the preferential pairing of matching DNA tiles over partially non-matching DNA tiles, but did not at that time experimentally demonstrate a DNA self-assembly for a (non-trivial) computation.

Erik Winfree, *et al.* [123] experimentally constructed the first large (involving thousands of individual tiles) two dimensional arrays of DNA crystals by self-assembly of nearly identical DNA tiles. The tiles consisted of two double-crossovers (DX) which self-assemble into a periodic 2D lattice. They produced atomic force microscope (AFM) images of these tilings (by insertion of a hairpin sequence into one of the tiles they created 25 nm stripes in the lattice). They also verified the assembly by the use of *reporter* ssDNA sequences spanning multiple tiles adjacent in the lattice, where the reporter ssDNA sequences were formed by ligation of individual ssDNA sequences within the adjacent tile. This experiment provided strong evidence of the feasibility of large scale self-assembly, but it was not in itself computational. LaBean, *et al.* [**?**, 48] designed and experimentally tested in the lab a new class of DNA tiles, denoted TAO, which is a rectangular shaped triple crossover molecule with sticky ends on each side that can match with other such tiles and with a reporter ssDNA sequence that runs through the tile from lower left to upper right, facilitating output of the tiling computation. Future major milestones will be to experimentally demonstrate: (i) DNA

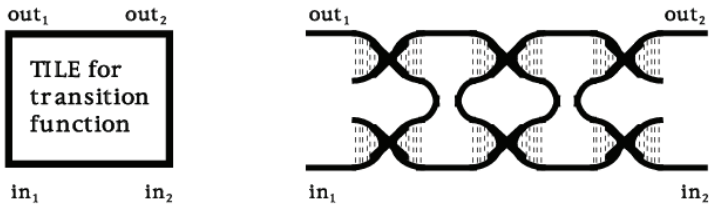

FIGURE 1
The tile transition function and a DNA DX nano-structure for the tile.

self-assembly for a (non-trivial) computation, and (ii) DNA self-assembly of a (possibly non-computational) 3D tiling.

Simulation of a $1D$ Cellular Automaton. We now consider Winfree's [120] direct simulation of a $1D$ cellular automaton running in time $T \geq n$ and space $S \geq n$ by assembly of an $S/2 \times T$ array with assembly depth T. Each tile π has two distinguished pads in_1, in_2 ordered left to right on its bottom side, and two distinguished pads out_1, out_2 ordered left to right on its top side (see Figure 1).

The pads are defined from the transitions of the cellular automaton. We adopt the convention of considering the bottom of the tile to be associated with the current step and the top of the tile associated with subsequent step, via its pads in these positions. In Winfree's [120] intended assembly, for each $t, 0 \leq t < T$ and $i, 1 \leq i \leq S$ where $(i = t) \mod 2$, there will be placed a tile $\pi(i, t)$ representing the transition from time $t - 1$ to time t at the two consecutive tape cells at positions $i, i + 1$. The bottom of the tile is associated with time $t - 1$ and the top of the tile is associated with time t, via its pads in these positions. The construction was intended to result in an assembly so tile $\pi(i, t)$ is placed to abut and match its pads in_1, in_2 at the out_2, out_1 respective pads of the neighboring tiles $\pi(i - 1, t - 1), \pi(i + 1, t - 1)$, which abut just below $\pi(i, t)$. Also, tile $\pi(i, t)$ is placed to abut and match its pads out_1, out_2 at the in_2, in_1 respective pads of the neighboring tiles $\pi(i - 1, t + 1), \pi(i + 1, t + 1)$ which abut just atop $\pi(i, t)$. The size (number of tiles placed) of the assembly is $A = ST/2$.

Is Unmediated DNA Self-Assembly Feasible for Computation? We have noted that the idea of unmediated DNA assembly has theoretical and mathematical foundations provided by the known tiling complexity results which were quite well established in the 1960s and 1970s. However, the idea of unmediated DNA assembly can not necessarily rely only on these, for it needs some sort of analysis or experimental validation of the random assembly process itself.

4 BMC VIA STEP-WISE ASSEMBLY

We consider here an assembly method for computation which we call *step-wise assembly* that involves control of the assembly in distinct steps. Our step-wise assembly method refines the unmediated self-assembly method of Winfree, *et al.* [120, 124]. It may increase the likelihood that the assembly always succeeds. However, it has the drawback of not being fully autonomous. For example, we modify the assembly of Winfree, *et al.* [120], as detailed also in Section 3, to control the assembly of the $S/2 \times T$ tiling array so it is done in breadth first manner from the initial tile placements, in T distinct steps.

As in the previous Section 3, we idealize the DNA molecules to be assembled as polyhedra which we call *tiles*, with pads on the sides that can be matched by Watson-Crick complementation, and the assembly of the tiles is due to hybridization of matching pads on the sides of the tiles.

We specialize the tiles used for each time step t to include *time stamps* on their pads; that is we augment the tile pads with characters distinctly encoding the assembly depth (complemented if need be to insure consistent Watson-Crick complementary matching). The input pads of each such tile are edited to contain a DNA string whose Watson-Crick complement encodes the step number t and the output pads of the tile are edited to contain also a unique DNA string encoding $t + 1$ (this is used to insure Watson-Crick complementary matching of the tile layer for time step t with the tile layer for time step $t + 1$)

The number of tile types is increased by a multiplicative factor of d, and each resulting tile has an associated depth. We presume that the initial tiles are initially placed as specified, without any initial blockage. To insure that distinct assemblies do not interfere with each other, we assume each initial assembly is solid-supported using standard surface biotechnology (e.g., immobilized via surface attachments [110]).

We will consider the entire assembly to be done with a maximal matching tile in each given location, and assume that there are no partial or total mismatches allowed between the pads of abutting placed tiles.

For our assembly algorithms, for simplicity we assume a *worst case assembly* model: we assume that the further assembly is done by choosing a worst case unit region which adjoins at least one region so far tiled and where further tiling is possible (the region must be so far untiled, and not a blockage), and then choose (the worst case) maximal matching tile to place there, where that tile is chosen among all tiles where there are no pad mismatches among abutting tiles. We repeat this worst case choice of regions to tile, until every such region that can be tiled, is tiled in this worst case manner. (Note that our assumption that there are no pad mismatches is intended to

approximate the actual case where the likelihood of pad mismatch is nonzero, but very low, by appropriate choice of key parameters such as temperature and match lengths.)

Our idea is to provide control for the assembly in discrete steps $t = 1, 2, \ldots$. We add all the tiles (providing multiple copies of each type of tile) of depth t on step t and after sufficient time for the completion of this step's further DNA assembly, wash away the unattached tiles of step t. We will insure no blockage, as previously described, can ever occur, in our worst case assembly model. As in the previous assembly methods, we require no thermal cycling.

4.1 Emulation of a $1D$ Array Using Step-wise $2D$ Assembly

To illustrate the idea of step-wise assembly, we now refine Winfree's [120] direct simulation of a cellular automaton running in time $T \geq n$ and space $S \geq n$ by assembly of an $S/2 \times T$ array of $A = ST/2$ tiles. We again assume the reader is familiar with his construction.

We augment the pads of each of the tiles defined in Winfree's construction by including *time stamps*: that is for each $t, 0 \leq t \leq T-1$, and for each tile, the bottom pads in_1, in_2 are edited to contain a unique DNA time stamp encoding of t (complemented to insure Watson-Crick complementary matching) and the top pads out_1, out_2 are edited to contain also a unique DNA time stamp encoding $t+1$ (non-complemented to insure Watson-Crick complementary matching). Thus, in our intended assembly, the neighboring tiles $\pi(i-1, t-1), \pi(i+1, t-1)$ have both of their upper side pads out_1, out_2 composed with time stamp encoding t, and the neighboring tiles $\pi(i-1, t), \pi(i+1, t)$ have both of their bottom side pads in_1, in_2 composed with time stamp encoding t.

The assembly is controlled as follows: the tiles representing the initial configuration of the machine at time $t = 0$ can be assume to be initially placed in correct location, as $\pi(1, 0), \ldots, \pi(S, 0)$. For each $t = 1, \ldots, T$ we must add to the assembly only the step t tiles which are time-stamped with $t-1$ in pads in_1, in_2 and t in pads out_1, out_2. These time-stamped tiles can and must be uniquely placed as $\{\pi(i, t)|, 1 \leq i \leq S$ where $(i = t) \mod 2\}$, thus allowing the assembly to advance from time $t-1$ to time t on all the tape positions $1 \leq i \leq S$ where $(i = t) \mod 2\}$. The resulting step t addition to the assembly is in the shape of a $2D$ rectangular tiling of length S and unit height (see Figure 2).The time-stamped paddings of each of these tiles do not interfere with any other tiles at this time: the assembly correctly places all tiles associated with each time step $t > 0$. Thus, the resulting tiling assembly is unique.

Note that we can somewhat simplify time stamping by using time stamps modulo 2. This requires us to use solid-support methodology [126] for the

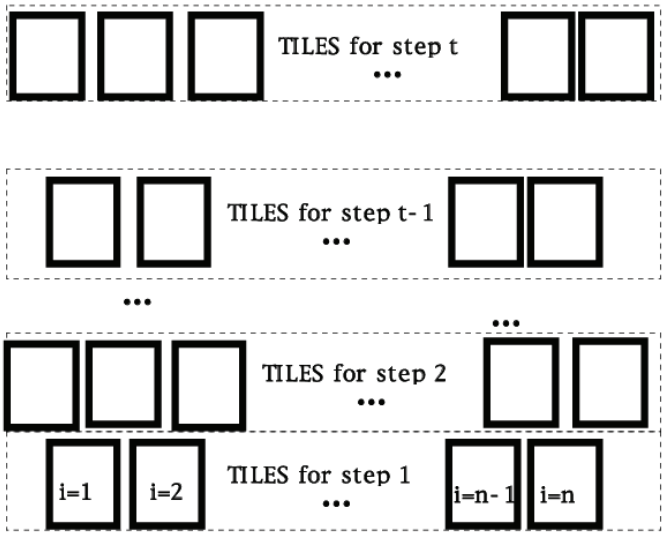

FIGURE 2
Step-wise assembly for step t.

assemblies and to wash away all the tiles of previous steps $t' < t$ before proceeding to step t.

4.2 Nanofabrication of the Tiling Assembly

We now breifly discuss how the tiling assembly is to be done via DNA self-assembly methods for nanofabrication.

Nanofabrication of Tiles. The tiles need be made quite rigid, so the shape conforms with the intended multi-tile assembly. The tiles can be contructed by the nanofabrication techniques of Seeman, *et al.* as surveyed in [103–105] (see also Subsection 2.2). Interaction between tiles during their construction can be avoided using solid-support methodology described in [126]. Winfree, *et al.* [120, 124] made use of DX molecules [38] for nanofabrication of rigid tiles, and it may also be possible to make use of DNA polycrossover molecules. It should be cautioned that considerable care must be made to insuring the geometry is correct so that (i) the tiles are rigid as intended, (ii) the shape conforms with the intended multi-tile assembly, (iii) the pads of matching tiles are correctly aligned, and (iv) the helical twist of the doubly stranded DNA is respected. Since the nanofabrication techniques of Seeman, *et al.* are at this time more of an art than a science (see [103]), considerable experimentation is required.

Encoding Methods for Pads. Throughout this paper, we will assume distinct encoding functions such as E from tuples of small integers to short distinct ssDNA words. We assume the ssDNA words are not degenerate (i.e., with repeated subsequences or reverse subsequences to avoid mismatching), and are chosen using the DNA word design techniques as described in [1, 11, 19, 41, 63] and [27, 28, 41, 52, 55, 71].

These known ssDNA word design techniques have the dual goals of (i) minimization of secondary structure (e.g., unwanted folding) and (ii) maximization of binding specificity and discrimination efficiency. We let $\overline{E}(-)$ denote the Watson-Crick complementation of these encodings.

Readout Methods After Assembly. The final configuration of the simulation machine is given by the linear sequence of pads on the bottom sides of the *final tiles:* $\{\pi(i, T)|, 1 \leq i \leq S \text{ where } (i = T) \mod 2\}$ of step T. This output might be constructed as a ssDNA in $O(1)$ BMC steps by a variety of well known techniques:

1. By adding short segments of ssDNA, each containing the composition of a pair of ssDNA which are Watson-Crick complements of a consecutive pair of pads on the top side of the final tiles, and then ligating these short segments together to form the required output ssDNA.
2. By adding dangling ssDNA segments attached to the top ends of the final tiles, ligating consecutive pairs of these together to form a single ssDNA, and disconnecting (by appropriate use of restriction enzymes) the side edges of the final tiles.

It should be cautioned again that in these constructions, considerable care must be made (and experimentation), to insure the geometry is correct so that the readout ssDNA is correctly aligned with the appropriate pads of the tiles, and the helical twist of the dsDNA is respected.

Rigid Framing of the Initial Assembly. The assembly frame is one of the innovations of this paper and is also employed in many of our other assembly constructions given in this paper. An assembly frame is a rigid DNA nanostructure. The purpose of the assembly frame is (i) to constrain the placement of the input ssDNA on the boundaries of the tiling assembly, and (ii) to initiate and further constrain the geometry of the subsequent assembly. The input ssDNA strand binds to the assembly frame at prescribed places along the strand, conforming to the boundaries of the assembly frame. The initial tiling assembly also binds to these input ssDNA strands (at other prescribed places). In this way the assembly frame constrains the initial tiling assembly, and thus constrains the geometry of the subsequent tiling assembly.

For example, in our simulation of a $1D$ automaton, to insure the initial tiles are placed as specified, i.e., as a $S/2 \times 1$ rectangle, we can employ a rigid DNA structure, which we call an *assembly frame*. In this case, this can be easily done by adding additional pads on the left and right sides of each of the initial tiles for first time step $t = 1$. We then define these pads so that the left pad of each tile is the same as the right pad of the tile intended on its right, and so that the right pad of each tile is the same as the left pad of the tile intended on its left.

In general, a frame may constrain the assembly to a prescribed polygonal path, e.g., a straight line, or constrain the assembly within a prescribed polygon, e.g., a rectangle. (See Figures 6 and 23), giving a ssDNA encoding the input n-vector, and see Figures 7 and 24, giving a possible DNA nanostructure for the rectangular frame.) The construction of an assembly frame in this case is less straightforward, but never-the-less can be done by known recombinant DNA techniques and/or DNA nanofabrication techniques of Seeman *et al.* [105], as discussed in subsection 2.2:

- We can make use of dsDNA, which is much more rigid than ssDNA for moderate lengths.
- We can make use of the DNA junctions developed by Seeman, *et al.* [33, 102] and the DX molecules of Fu and Seeman [38] which are quite rigid, to form a DNA superstructure to hold the DNA strand in the required initial position.
- We can insert into the input, which we generally assume is ssDNA, some unique pads at regular distances, and then build a DNA superstructure (built of DX molecules) containing a polygon of the required initial shape with matching pads (provided by Watson-Crick complementation) to hold the input ssDNA in place.
- Each initial assembly frame can be solid-supported using standard surface biotechnology, to insure distinct assemblies to not interfere with each other.

3-way and 4-way Pairing of DNA Strands. The construction of an assembly frame can be facilitated in certain cases by 3-way pairing of DNA strands. There are number of known methodologies for 3-way pairing of DNA strands.

- The DNA triple helix can be constructed by the Dervan system [30].
- Another option is the use of a motif known as DX+J [61], which is almost as stiff as DX, and provides the option of having DNA come out the side of a tile.

- Seeman [98] further suggests to ligate from the perpendicular arm a second 3-arm junction, which would provide a location for a complementary ssDNA to bind, with 4-ary recognition.

4.3 Extensions to Emulation of $2D$ Processor Arrays

Our step-wise assembly techniques can also be extended to $3D$ DNA assembly to simulate a $2D$ processor array where each processor is an FSA with the same finite state control. Let a $2D$ *cellular automaton* be a Turing Machine with a single $2D$ square tape of shape $S' \times S'$ such that disjoint pairs of adjacent cells are updated in parallel by a transition function depending on those cells and the state of the machine. We describe a T step $3D$ DNA array assembly of total size $ST/4$, for simulating T steps of a processor array with $S = (S')^2$ cells, refining a similar (but unmediated) $3D$ assembly method briefly discussed in [120] and also [124]. In the intended assembly, for each $t, 1 \leq t \leq T$ and $i, 1 \leq i, j \leq S'$ where $(i = t) \mod 2$ and $(j = t) \mod 2$ there will be a cubic tile $\pi(i, j, t)$ representing the transition from time $t-1$ to time t at the four adjacent tape cells at positions $(i, j), (i+1, j), (i+1, j+1), (i, j+1)$ (we will adopt this rotational order for positioning of pads as well). We again adopt the convention of considering the bottom of the tile to be associated with time $t-1$ and the top of the tile associated with time t, via its pads in these positions. This requires the construction of $3D$ DNA cubes using the nanotechnology developed by Seeman *et al.* [103]. Each cubic tile $\pi(i, j, t)$ has (i) four distinguished pads in_1, in_2, in_3, in_4 rotationally positioned at the four corners on its bottom side augmented with time stamps encoding $t-1$ (complemented to insure Watson-Crick complementary matching) and also (ii) four distinguished pads $out_1, out_2, out_3, out_4$ similarly rotationally positioned at the four corners on its bottom side augmented with time stamps encoding t (non-complemented to insure Watson-Crick complementary matching). This results in an assembly that has tile $\pi(i, j, t)$ abut and match its pads in_1, in_2, in_3, in_4 at the $out_3, out_4, out_1, out_2$ respective pads of the neighboring tile $\pi(i-1, j-1, t-1), \pi(i+1, j-1, t-1), \pi(i+1, j+1, t-1), \pi(i-1, j+1, t-1)$, which abut just below $\pi(i, j, t)$. Also, the assembly that has tile $\pi(i, j, t)$ abut and match pads $out_1, out_2, out_3, out_4$ at the in_3, in_4, in_1, in_2 respective pads of the at neighboring tiles $\pi(i-1, j-1, t+1), \pi(i+1, j-1, t+1), \pi(i+1, j+1, t+1), \pi(i-1, j+1, t+1)$, which abut just atop $\pi(i, j, t)$. The size of the assembly is $A = ST/4$.

It should be cautioned that such $3D$ assemblies may entail considerably more difficulties than $2D$ assemblies. In particular: (i) the cubic tiles need to be designed to be rigid (a multiple DX is suggested by [120], but may not be rigid), and (ii) the geometry of the assembly must be carefully designed to facilitate diffusion and access of polyhedral tiles into the untiled locations

adjacent to the $3D$ assembly (this may be improved by our use of step-wise assembly).

5 A LINEAR SIZE AND DEPTH PREFIX COMPUTATION ASSEMBLY

One drawback of the step-wise assembly method, if applied using the standard regular square tiling simulations of T step computations, is that it results in a large assembly of $A = ST$ tiles. In the rest of this paper we give our main results, which are tilings that are much more compact, and have considerably smaller total size which is linear in the input. We apply these tilings to solve some restricted, but fundamental, problems that arise in parallel computation.

Monoid Sum and Prefix Computation. Let $(D, \cdot)$ be a *monoid*, that is D is a set with an identity element λ over the operation $\cdot$ and the operation $\cdot$ is associative. Suppose we are given as input an n-vector $(a_1, \ldots, a_n)$ with elements from some domain D. The *monoid sum problem* is given $(a_1, \ldots, a_n)$, to compute $a_1 \cdot \cdots \cdot a_n$. The *prefix computation problem* is given $(a_1, \ldots, a_n)$, to compute each prefix: $b_i = a_1 \cdot \cdots \cdot a_i$, for $i = 1, \ldots, n$. The monoid sum and prefix computation problems occur in many applications, for example:

1. *Prefix sums:* in this case the domain D are assumed to be small b-bit numbers, the operation $\cdot$ represents integer addition, and n is the number of integers to be summed.
2. *Integer arithmetic*, including addition, subtraction, and multiplication times a constant. In the case of n bit binary addition, the domain is $D = \{\text{carry, no-carry, carry-propagate}\}$ and the operation $\cdot$ represents carry-sums from a bit position to the next higher bit position. Once the propagated carries are determined at each bit position by the prefix computation, the output bits giving the bits of the addition can be immediately determined.
3. *Finite State Automaton (FSA) simulation:* in this case the domain D is the set of states of the FSA paired with the input symbols, the operation $\cdot$ represents the finite state transitions, and n is the length of the input string. Once the resulting states at each time step are determined by the prefix computation, the output of the FSA can be immediately determined.
4. *fingerprinting strings:* given a length n string, the operation $\cdot$ represents is defined to be an associative hash function, and the monoid sum gives a hash encoding of the input string.

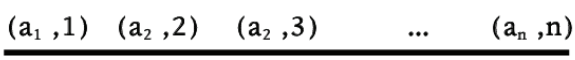

FIGURE 3
An ssDNA strand encoding input n-vector fixed in a straight line.

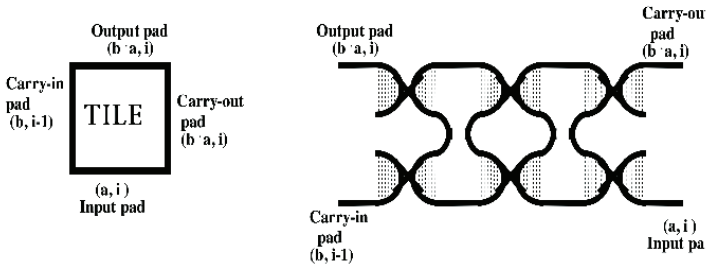

FIGURE 4
Square tile for sequential prefix computation and a DNA DX nano-structure for the tile.

All these problems have obvious sequential algorithms with linear work for many conventional models of computation such as circuits and random access machines.

Linear Assembly for Prefix Computation. To solve the prefix computation problem $(a_1, \ldots, a_n)$ by LP-BMC, we propose a linear size assembly. We assume distinct encoding functions E, E' from pairs in $D \times \{1, \ldots, n\}$ to short distinct ssDNA words, as briefly discussed in subsection 4.2. We let $\bar{E}(-)$, $\bar{E}'(-)$ denote the Watson-Crick complementation of these encodings. We assume the input ssDNA encodes the vector $(a_1, \ldots, a_n)$ as a sequence of unit length ssDNA words of the form $E(a_1, 1), \ldots, E(a_n, n)$. This input ssDNA is positioned into a straight line of length n (see Figure 3) by the use of rigid framing techniques described in subsection 4.2, e.g., the use of a DNA DX superstructure with pads.

Each tile in this case is a unit square as given in Figure 4. Each tile will have pads, as defined below[2]. For each $a, b \in D$, $a' = b \cdot a \in D$ and $t \in \{1, \ldots, n\}$ (where if $t = 1$ then $b = \lambda$), there is a *step* t tile $\tau(a, b, t)$ with:

- the *input* pad $\bar{E}(a, t)$ on the bottom side (with the intent to match with the tth input),

[2]To reduce notation in this and all further Figures of this paper, we simply label each tile pad within the Figures without use of the encoding notation E, and drop complementation notation.

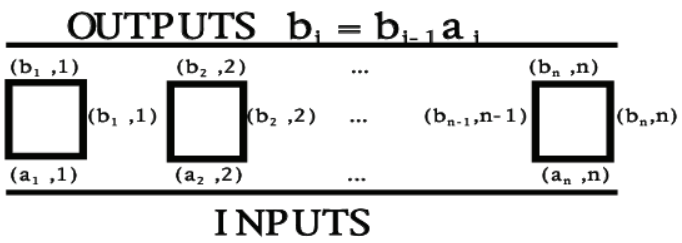

FIGURE 5
2D assembly for sequential prefix computation.

- the *output* pad $\bar{E}''(a', t)$ on the top side (with the intent to match with the tth output),
- the *carry-in* pad $\bar{E}'(b)$, on the left side (with the intent, for $t > 1$, to match with the $(t-1)$-th prefix given by the carry-out pad on the left side of the $t-1$ tile), and
- the *carry-out* pad $E'(a')$ on the right side (with the intent to match with the tth prefix given by the carry-in pad on the left side of the $t+1$ tile).

We use step-wise assembly, adding all step t tiles at time $t = 1, \ldots, n$. Suppose the prefix computation problem $(a_1, \ldots, a_n)$ has output $(b_1, \ldots, b_n)$, where $b_i = a_1 \cdot \cdots \cdot a_i$. For each $t = 1, \ldots, n$ the tile $\tau(a_t, b_t, t)$ is the unique tile that can be placed on the tth step, and that must be placed so its input pad matches the tth element of the input, and for $t > 2$, this tile also abuts the previous tile $\tau(a_{t-1}, b_{t-1}, t-1)$ matching the respective carry-in and carry-out pads. Thus, the tiling is unique and there can be no blockages.

The resulting $2D$ assembly (see Figure 5) is in the shape of rectangular tiling of length n and unit height, containing tiles $\tau(a_1, b_1, 1)$, $\tau(a_2, b_2, 2), \ldots, \tau(a_n, b_n, n)$, in this order, with the encoded input matched to the input pads on the bottom sides of these tiles and the output pads on the top sides of the tiles giving the encoded output sequence $(\bar{E}(b_1, 1), \ldots, \bar{E}(b_n, n))$.

6 APPLICATION TO BIT-SERIAL INTEGER ADDITION

Lipton *et al.* [18] developed circuit evaluation algorithms using BMC which allow for binary addition at the cost of a number of steps linear in the size of the circuit. Bancroft and Guarnieri [44] demonstrated the first experimental execution of a DNA-based arithmetic calculation on a single pair of bits. Their methods, if extended to n-bit arithmetic, require at least n

thermal cycles and a considerable number of recombinant DNA operations per bit.

Here, we give a step-wise assembly method requiring no repeated thermal cycling. The *integer addition problem* can be considered to have as input two length n vectors $(a_1, a_2, \ldots, a_{n-1}, a_n)$ and $(a'_1, a_2, \ldots, a'_{n-1}, a'_n)$ of bits representing the binary representation of two numbers; the output is a length $n+1$ vector $(s_1, s_2, \ldots, s_n, s_{n+1})$ of bits representing the sum of these two numbers.

We apply our proposed step-wise method for prefix computation to synthesize a step-wise method for n-bit binary addition. Our method will take n steps and will construct an $O(n)$ size assembly (the number of tiles in the assembly can be further reduced to $O(n/\log n)$ by a somewhat more complex construction where we use a k-ary encoding on the input numbers, for $k = \log n$, and define tiles to do the appropriate k-ary arithmetic). We assume the same distinct encoding functions E, E', E'' and assume the input is a ssDNA encoding the vectors $(a_1, a_2, \ldots, a_{n-1}, a_n)$ and $(a'_1, a_2, \ldots, a'_{n-1}, a'_n)$ as a sequence of unit length ssDNA words of the form $E(a_1, 1), \ldots E(a_n, n), E(a'_1, n+1), \ldots, E(a'_n, 2n)$. It will be useful to define the $n+1$ bits of the input $a_{n+1} = 0, a'_{n+1} = 0$. For technical reasons, we need to reverse the order of the encoding of the bits of the second input number. Again by known recombinant DNA techniques (as described in our Section 8 on assemblies for shuffle permutations), we can reverse the latter part of the input ssDNA containing the latter n elements $E(a'_1, n+1) \ldots E(a'_n, 2n)$, yielding a ssDNA with a sequence of unit length ssDNA words of the form $E(a_1, 1) \ldots E(a_n, n) E(0, n+1)\beta$, $\overline{E}(0, 2n+1)^R \overline{E}(a'_n, 2n)^R, \ldots \overline{E}(a'_1, n+1)^R$.

Also, a unit length ssDNA dummy (distinct from all the other strings used in DNA encodings) can easily be inserted into this ssDNA just after the n-th element. This ssDNA is positioned into the shape of a rectangle (see Figure 6) of length $n+1$ and unit height by the use of rigid framing techniques described in subsection 4.2 (see Figure 7).

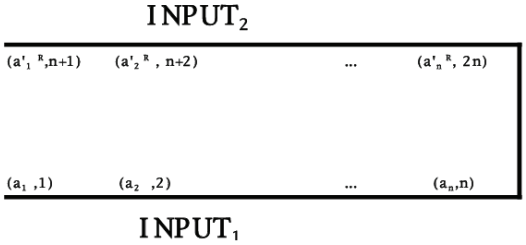

FIGURE 6
Input ssDNA encoding two binary numbers.

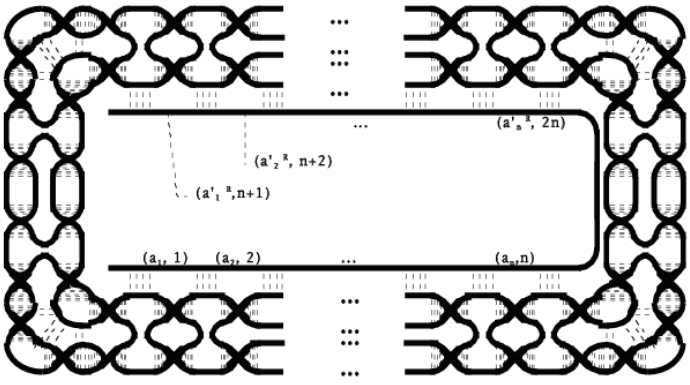

FIGURE 7
A DNA DX nano-structure for the rectangular frame.

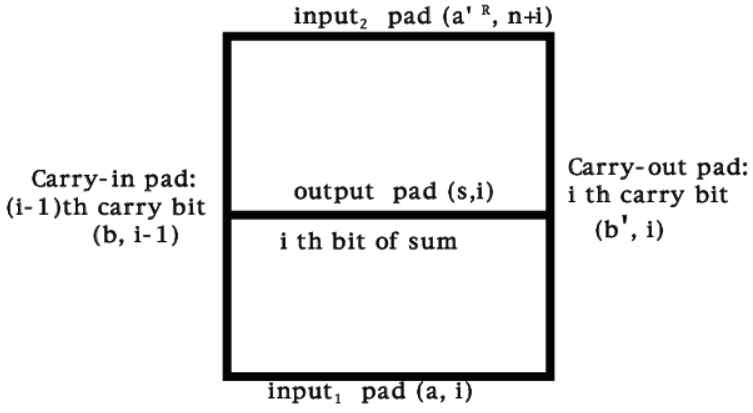

FIGURE 8
Square tile with middle segment, used for bit-serial addition.

Each tile in this case is a unit square with a unit length internal middle segment as given in Figure 8.

For each $a, a', b \in \{0, 1\}$ and $i \in \{1, \ldots, n+1\}$ (where if $i = 1$ then $b = 0$), there is a tile $\tau^*(a, a', b, i)$ with:

- a unit square with
  - *input*$_1$ pad $\bar{E}(a, i)$ on the bottom side, (with the intent of giving the i-th bit of the first input number),
  - *input*$_2$ pad $E^R(a', n+i)$ on the top side, (with the intent of giving the i-th bit of the second input number),

  - *carry-in* pad $\bar{E}'(b, i-1)$ on the left side, (with the intent of giving the Watson-Crick complement of the $(i-1)$-th carry bit b), and
  - *carry-out* pad $E'(b', i)$ on the right side, for $b' = (a \vee a') \wedge (a \vee b) \wedge (b \vee a')$ (with the intent of giving the i-th carry bit b'),
- an internal middle segment, running between the left and right sides, with an *output* pad $\bar{E}''(s, i)$, where $s = a \oplus a' \oplus b$ (with the intent of giving the i-th bit of the intended output).

Thus, $\tau^*(a, a', b, i)$ determines the carry-sum at the i-th position from both the inputs at the i-th position and the carry from the $i-1$ bit position, and then propagates the carry-sum at the i-th position to the $i-1$ bit position.

We will use a step-wise mediated assembly, adding all tiles of the form $\tau^*(a, a', b, i)$ on distinct steps $i = 1, \ldots, n+1$. For each $i \in \{1, \ldots, n\}$, the tile $\tau^*(a_i, a'_i, i)$ is the unique tile that can be placed so

- the bottom side abuts (input_1 pad matches) the i-th element $E(a_i, i)$ of the input giving the i-th bit of the first input number,
- the top side abuts (input_2 pad matches) the element $\bar{E}(a'_i, n+i)$ giving the i-th bit of the second input number,
- the left side abuts (the carry-in pad matches) $E'(b_{i-1}, i-1)$, giving the $(i-1)$-th carry bit b_{i-1} (which is 0 if $i = 1$),
- the right side abuts (the carry-out pad matches) $E'(b_i, i)$, providing the i-th carry bit $b_i = (a_i \vee a'_i) \wedge (a_i \vee b_i) \wedge (b_i \vee a'_i)$, and
- the middle segment has output pad $\bar{E}''(s_i, i)$, where $s_i = a_i \oplus a'_i \oplus b_i$, giving the i-th bit of the intended output.

The location of this tile does not interfere with the placement of any other tile.

The resulting $2D$ assembly (see Figure 9) is unique and contains tiles $\tau^*(a_1, a'_1, 1)$, $\tau^*(a_2, a'_2, 2), \ldots,$ $\tau^*(a_n, a'_n, n)$, $\tau^*(a_{n+1}, a'_{n+1}, n+1)$ in this order, with the inputs matched to the top and bottom side input pads input_1, input_2 of these square tiles, with the carry pads of consecutive tiles matched, and with the output pads of the middle segment of the tiles giving the required encoded output sequence $\bar{E}''(s_1, 1), \bar{E}''(s_2, 2), \ldots, \bar{E}''(s_{n-1}, n-1), \bar{E}''(s_n, n)$, where $s_i = a_i \oplus a'_i \oplus b_i$ is the i-th bit of the sum of the input numbers. Again, this output can be constructed as a ssDNA in $O(1)$ BMC steps by a variety of known techniques, as described at the end of Subsection 8.2.

(Alternatively, the output can again be facilitated by redefining each tile to be a $3D$ polyhedron resembling a pup tent as given in Figure 10, with

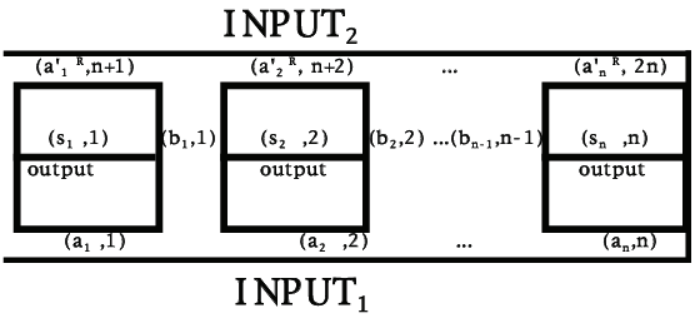

FIGURE 9
2D assembly for bit-serial addition.

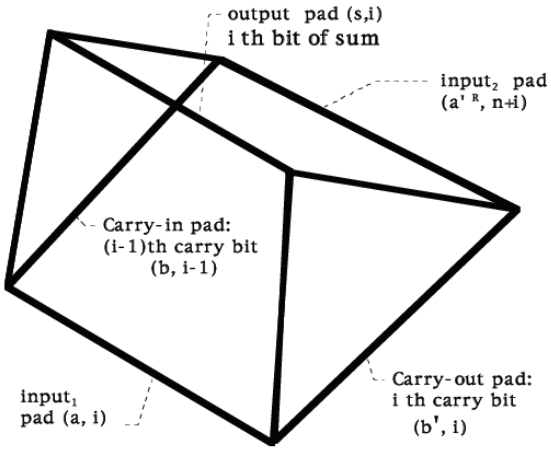

FIGURE 10
Alternative 3D polyhedral tile for bit-serial addition.

the same square base as the square tiles described above, and with sloping rectangular sides meeting at a common segment with an output pad.)

7 LOGARITHMIC DEPTH ASSEMBLY FOR PREFIX COMPUTATION

A drawback of our proposed step-wise assembly method given in Section 5, if applied using the standard regular square tiling simulations of T step computations, is that it involves T assembly steps, which is the depth of such a regular tiling.

We now consider tilings that are not regular, and thus can have considerably smaller depth. We apply these tiling to some restricted, but fundamental,

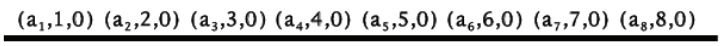

FIGURE 11
Input ssDNA encoding an n-vector.

problems that arise in parallel computation. The prefix computation problem has known optimal parallel algorithm (for circuits and parallel random access machines), with $O(\log n)$ time and n work (the work bound is the product of the number of processors times the parallel time) due to Ladner and Fischer [47].

Fortune and Wyllie [37] developed a technique known as *parallel pointer jumping* which contracts consecutive pairs of an input list $\mathcal{L}^{(0)} = (a_1, \ldots, a_n)$, into single elements. Let us assume n is a power of 2. Let $(D, \cdot)$ be the monoid associated with the problem. For example the sublist (a_i, a_{i+1}), for an odd i, is contracted by pointer jumping into $(a_i \cdot a_{i+1})$. When executed in parallel on every consecutive odd-even pair of an even length list $(a_1, a_2, \ldots, a_{n-1}, a_n)$, parallel pointer jumping contracts this list into the list $\mathcal{L}^{(1)} = (a_1 \cdot a_2, a_3 \cdot a_4, \ldots, a_{n-1} \cdot a_n)$ of length $n/2$. The result of t stages of parallel pointer jumping is the list $\mathcal{L}^{(t)} = (a_1^{(t)}, a_2^{(t)}, \ldots, a_{n^{(t)}}^{(t)})$, where $n^{(t)} = n/2^{t-1}$, and for each odd $i \in \{1, \ldots, n^{(t)}\}$, we have $a_{(i+1)/2}^{(t+1)} = a_i^{(t)} \cdot a_{i+1}^{(t)}$. Repeating this parallel pointer jumping for $T = \log n$ stages contracts a length $n = 2^T$ list to a single element.

To solve the monoid sum problem $(a_1, \ldots, a_n)$, (computing output $a_1 \cdot \cdots \cdot a_n$) where n is a power of 2, we propose a linear size assembly by LP-BMC in $\log n$ steps using self-assembling tiles to emulate parallel pointer jumping.

We slightly redefine the distinct encoding functions E, E' from pairs in $D \times \{1, \ldots, n\} \times \{0, \ldots, \log n\}$ to short distinct binary ssDNA words. We assume the input ssDNA encodes the vector $\mathcal{L}^{(0)} = (a_1, \ldots, a_n)$ as a sequence of unit length ssDNA words of the form $E(a_1, 1, t) \ldots E(a_n, n, t)$.

This ssDNA (see Figure 11) is initially positioned as a straight line (again, our assembly may be facilitated by the use of rigid framing techniques described in subsection 4.2).

The tiles are rectangular in shape of size $2^t \times 1$, for stages $t = 1, \ldots, \log n$, as given in Figure 12. The width of the rectangular tiles of step t is 2^t, and the height is 1.

For each $a, a' \in D$ and $t \in \{1, \ldots, \log n\}$, and odd $i \in \{1, \ldots, n^{(t)}\}$ there is a *step t* tile $\tau(a, a', i, t)$ with:

- on the lower side of the tile, they have two consecutive pads: the $input_{odd}$ pad $\bar{E}(a, i, t-1)$ followed by the $input_{even}$ pad $\bar{E}(a, i$

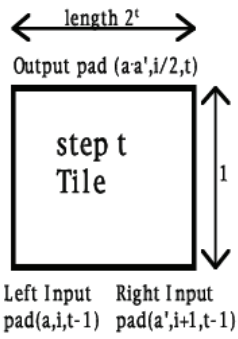

FIGURE 12
A Rectangular tile for parallel monoid sum computation.

$+1, t-1$) (with the intent to match these pads with values a, a' at consecutive positions $i, i+1$ of $\mathcal{L}^{(t-1)}$ computed in a previous stage $t-1$), and

- *output* pad $E(a'', (i+1)/2, t)$ on the upper side of the rectanglar tile, where $a'' = a \cdot a' \in D$ (with the intent to match this pad with the composition of the values at positions $i, i+1$ and provide this resulting value a'' to the $(i+1)/2$-th position of $\mathcal{L}^{(t)}$ on the subsequent step).

We have already noted that rigid square tiles have been nanofabricated in DNA, and by composing these, we can construct rigid rectangular square tiles.

We use step-wise assembly, adding all step t tiles at times $t = 1, \ldots, \log n$. The monoid sum problem $\mathcal{L}^{(0)} = (a_1, \ldots, a_n)$ has intended output $b_n = a_1 \cdot \cdots \cdot a_n$. Let $a_{i,j}$ denote $a_i \cdot a_{i+1} \cdots \cdot a_j$. The resulting assembly (see Figures 13, 14, 15) is a polyhedral tiling of size $O(n)$ and depth $O(\log n)$.

An inductive argument shows that in the t-th step of assembly, the unmatched bases of the rectanglar tiles involved in the assembly at step t form a unique (note that the chain is uniquely determined since the

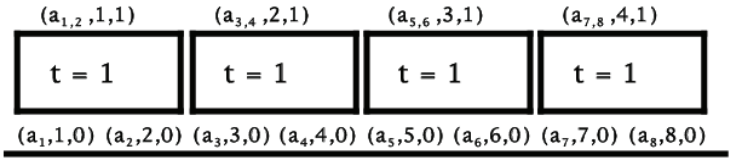

FIGURE 13
Step $t = 1$ of step-wise assembly for parallel monoid sum computes $a_{i,i+1} = a_i \cdot a_i + 1$ for odd i.

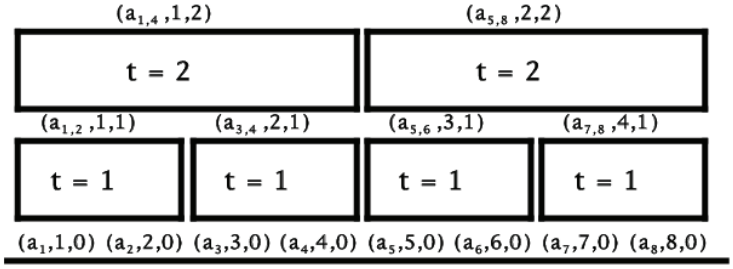

FIGURE 14
Step $t = 2$ of step-wise assembly for parallel monoid sum computes $b_4 = a_{1,4} = a_1 \cdot a_2 \cdot a_3 \cdot a_4$ and $a_{5,8} = a_5 \cdot a_6 \cdot a_7 \cdot a_8$.

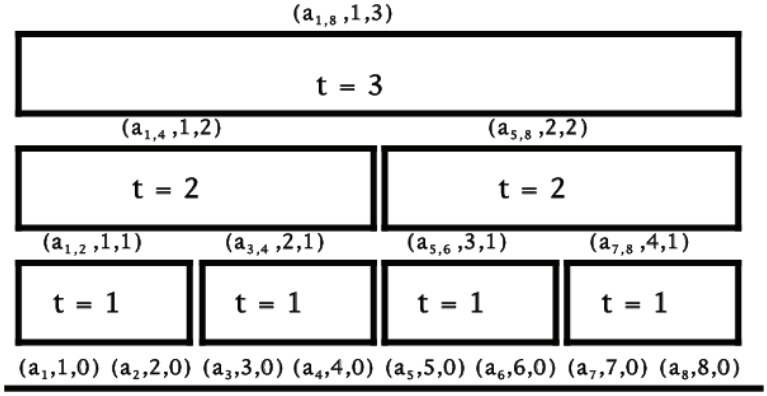

FIGURE 15
Final step $t = 3$ of assembly for parallel monoid sum computes $b_8 = a_{1,8} = a_1 \cdot a_2 \ldots a_8$.

pairing due to pointer jumping is between only odd and even consecutive elements) length $n^{(t)} \leq n/2^t$ chain of output pads encoding the exactly the vector $\mathcal{L}^{(t)} = (a_1^{(t)}, a_2^{(t)}, \ldots, a_{n^{(t)}}^{(t)})$ derived from from the input vector $\mathcal{L}^{(0)} = (a_1, a_2, \ldots, a_{n-1}, a_n)$ after t stages of parallel pointer jumping. This clearly holds on the 1st stage, and remains true after each stage.

The step t rectanglar tiles will form a sequence:

$$\tau(a_1^{(t-1)}, a_2^{(t-1)}, 1, t), \tau(a_3^{(t-1)}, a_4^{(t-1)}, 3, t), \ldots, \tau(a_{n^{(t-1)}}^{(t-1)}, a_{n^{(t-1)}}^{(t-1)}, n^{(t)}, t).$$

The i-th rectangle $\tau(a_i^{(t-1)}, a_{i+1}^{(t-1)}, i, t)$, of this sequence has

- input$_{odd}$ pad $\bar{E}(a_i^{(t-1)}, i, t-1)$ which uniquely matches the i-th element of $\mathcal{L}^{(t-1)}$,

- input$_{even}$ pad $\bar{E}(a_{i+1}^{(t-1)}, i+1, t-1)$ which uniquely matches the $(i+1)$-th element of $\mathcal{L}^{(t-1)}$, and
- output pad $E(a_{i,(i+1)/2}^{(t)}, (i+1)/2, t)$, where $a_i^{(t-1)} \cdot a_{i+1}^{(t-1)} = a_{(i+1)/2}^{(t)}$ which provides the $i(i+1)/2$-th element of $\mathcal{L}^{(t)}$.

The location of each of these tiles do not interfere with the placement of any other tile at this step t, so the step-wise assembly insures no blockages.

At each stage, the number of elements of the vector $\mathcal{L}^{(t)}$ decreases by a factor of $1/2$, so $n^{(t)} \leq n^{(t-1)}/2$. Since $n^{(\log n)} = 1$, on the final stage $T = \log n$ there is only a single step T tile which is placed in the assembly and its output pad at its upper side is unmatched, with encoded output $E(a_1 \cdot a_2 \cdot \cdots \cdot a_n, 1)$ giving the solution $b_n = a_1 \cdot a_2 \cdot \cdots \cdot a_n$ of the monoid sum problem. We now extend this $O(\log n)$ assembly depth solution of the monoid sum problem to solve prefix computation in $O(\log n)$ assembly depth. (Note that this assembly can be simplified somewhat by letting the index i of each tile $\tau(a_1, a_2, i, t)$ be taken $\mod 2$, if the input and output values do not require indexing.)

3D Tiling for Prefix Computation. To solve the prefix computation problem $(a_1, \ldots, a_n)$ in $O(\log n)$ steps we propose an additional linear size assembly by LP-BMC. We will use self-assembling $3D$ polyhedra tiles to do reverse propagation of intermediate values computed by the rectanglular tiles in the earlier monoid sum assembly process. Recall the prefix computation problem $(a_1, \ldots, a_n)$ has intended output $(b_1, \ldots, b_n)$, where $b_i = a_1 \cdot \cdots \cdot a_i$. Also let $b_0 = \lambda$.

Each $3D$ polyhedron forming a tile in this case is given in Figure 16, consisting of two horizontal rectangles (each with the same shape of the

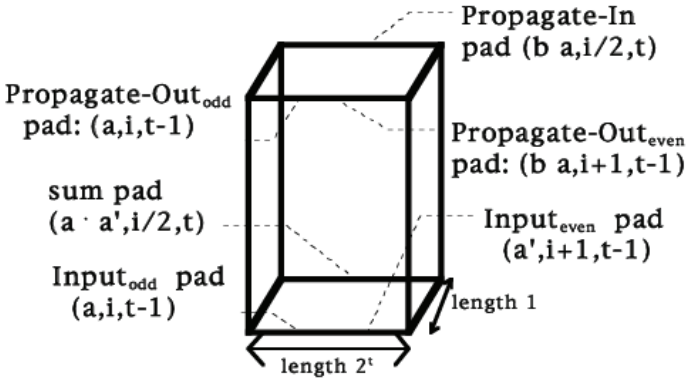

FIGURE 16
A 3D polyhedron tile for parallel prefix computation.

previously defined $2^t \times 1$ rectangular tiles for parallel monoid sum: with a base of length 2^t and sides of length 1) connected by unit length line segments between each of their corresponding vertices.

To allow matching of the polyhedra tiles defined here with the sides of the rectangular tiles for parallel monoid sum, we can simply install additional pads on the sides of those original rectangular tiles. Alternatively, we can assume a known methodology for 3-way pairing of DNA strands, as discussed in subsection 4.2, obtained by restricting the encoding of DNA strands to two bases. We let $\overline{E}(-)$, $\widehat{E}(-)$ denote the (possibly 3-way) Watson-Crick complementation of these encodings.

In this phase of the assembly, it is convenient to have the step numbers decrease from $\log n$ to 1. For each $a, a', b \in D$, and $t \in \{1, \ldots, \log n\}$ (where if $t = \log n$ then $b = \lambda$), and odd $i \in \{1, \ldots, n^{(t)}\}$ there is a *step t* polyhedron tile $\tau^+(a, a', b, i, t)$ with:

- on the lower rectangle (which is intended to match with an already placed rectanglar tile $\tau(a, a', i, t)$) there are:
  - *monoid sum input*$_{odd}$, *input*$_{even}$ pads $\widehat{E}(a, i, t-1)$, $\widehat{E}(a, i+1, t-1)$ on the lower side of the lower rectangle, (with the intent to match onto the previously placed rectanglar tile $\tau(a, a', i, t)$ with values a, a' at consecutive locations $i, i+1$ of $\mathcal{L}^{(t-1)}$ computed in a previous stage $t-1$), and
  - *monoid sum output* pad $E(a'', (i+1)/2, t)$ on the upper side of the lower rectangle, where $a'' = a \cdot a' \in D$ (with the intent to match with the composition of the values at locations $i, i+1$ given by the upper side pad of the previously placed rectanglar tile $\tau(a, a', i, t)$).
- on the upper rectangle (which is intended to propagate the partially computed prefix sums backward) there are:
  - *propagate-in* pad $\overline{E}'(b, (i+1)/2, t)$ on the upper side of the upper rectangle,
  - on the lower side of the upper rectangle, two consecutive pads: *propagate-out*$_{odd}$ pad $\widehat{E}'(b, i, t-1)$ (with the intent to match with b and propagate it as the prefix-sum to the i-th tile of step $t-1$), followed by *propagate-out*$_{even}$ pad $\widehat{E}'(b \cdot a, i+1, t-1)$ (with the intent to match with $b \cdot a$ and propagate it as the prefix-sum to the to the $(i+1)$-th tile of step $t-1$).

After completion of the step-wise assembly of the rectangular tiles for parallel monoid sum, as described above, we next use a step-wise assembly, adding all step t polyhedron tiles in reverse order $t = \log n, \log n - 1, \ldots, 1$. The prefix computation problem $(a_1, \ldots, a_n)$ has intended output $a_1 \cdot \cdots \cdot a_n$.

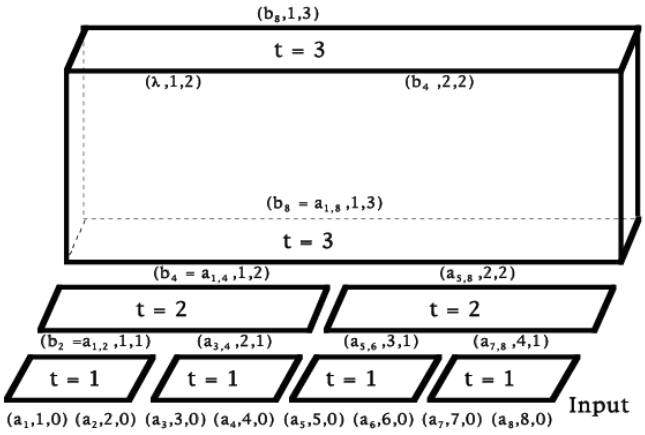

FIGURE 17
Stage $t = 3$ of 3D parallel prefix assembly: computation of b_4.

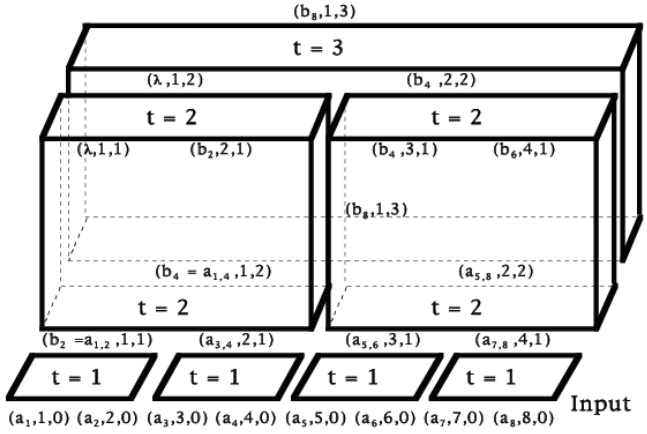

FIGURE 18
Stage $t = 2$ of 3D parallel prefix assembly: computation of b_2, b_4 and b_6.

The resulting $3D$ assembly (see Figures 17,18, and 19) is a polyhedral tiling of size $O(n)$ and depth $O(\log n)$.

To aid us in the analysis of our assembly, observe that if the result of t stages of parallel pointer jumping is the vector $\mathcal{L}^{(t)} = (a_1^{(t)}, a_2^{(t)}, \ldots, a_{n^{(t)}}^{(t)})$, then the solution of the prefix problem for $\mathcal{L}^{(t)}$ is given by $(b_1^{(t)}, b_2^{(t)}, \ldots, b_{n^{(t)}}^{(t)})$, with $b_0^{(t)} = \lambda$, where for each odd $i \in \{1, \ldots, n^{(t)}\}$, and $t = 2, \ldots, \log n$ we have $b_i^{(t-1)} = b_{(i-1)/2}^{(t)}$ and $b_{i+1}^{(t-1)} = a_i^{(t-1)} \cdot b_{(i-1)/2}^{(t)}$. Hence each $b_i = b_{i,1}$ can be computed by these recurrence equations; in fact we will determine them by matching pads of the assembly of tiles.

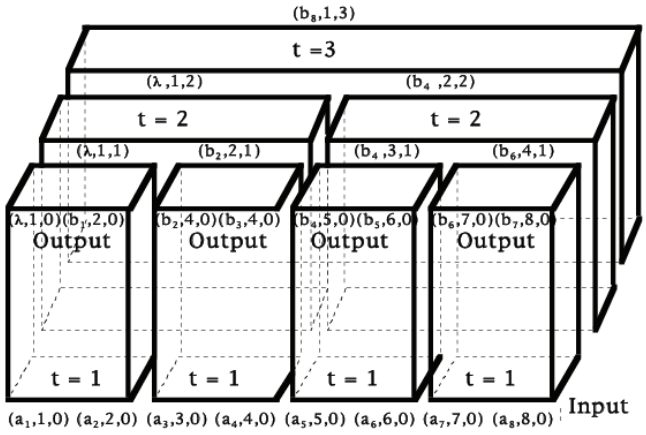

FIGURE 19
Final stage $t = 1$ of 3D parallel prefix assembly: computation of $b_1, \ldots, b_7$.

The step t polyhedral tiles will form a sequence:

$$\tau^+(a_1^{(t-1)}, a_2^{(t-1)}, b_0^{(t)}, 1, t), \tau^+(a_3^{(t-1)}, a_4^{(t-1)}, b_1^{(t)}, 1, t), \ldots,$$
$$\tau^+(a_{n^{(t-1)}}^{(t-1)}, a_{n^{(t-1)}, b_{n^{(t)}-1}^{(t)}}^{(t-1)}, n^{(t)}, t).$$

By induction we have that the i-th tile $\tau^+(a_i^{(t-1)}, a_{i+1}^{(t-1)}, b_{(i-1)/2}^{(t)}, i, t)$ has

- a lower rectangle with:
  - monoid sum input$_{odd}$, input$_{even}$ pads $\widehat{E}(a_i^{(t-1)}, i, t-1)$,$\widehat{E}(a_{i+1}^{(t-1)}, i+1, t-1)$ which match the (odd, even) i,$(i+1)$-th elements of $\mathcal{L}^{(t-1)}$, and
  - monoid sum output pad $E(a_{i,(i+1)/2}^{(t)}, (i+1)/2, t)$, since $a_i^{(t-1)} \cdot a_{i+1}^{(t-1)} = a_{(i+1)/2}^{(t)}$ which provides the $(i+1)/2$-th element of $\mathcal{L}^{(t)}$.
- and an upper rectangle with:
  - propagate-in pad $\overline{E}'(b_{(i-1)/2}^{(t)}, (i+1)/2, t)$ on the upper side, and
  - on the lower side: propagate-out$_{odd}$ pad $\widehat{E}'(b_{(i-1)/2}^{(t)}, i, t-1) = \widehat{E}'(b_{i-1}^{(t-1)}, i, t-1)$, since $b_{(i-1)/2}^{(t)} = b_{i-1}^{(t-1)}$, followed by propagate-out$_{even}$ pad $\widehat{E}'(b_{(i-1)/2}^{(t-1)} \cdot a_i^{(t-1)}, i, t-1) = \widehat{E}'(b_i^{(t-1)}, i, t-1)$ on the left side, on the top right side, since $b_{(i-1)/2}^{(t-1)} \cdot a_i^{(t-1)} = b_i^{(t-1)}$.

This clearly holds on the 1st stage, and remains true after each stage.

The output to the prefix computation problem is provided by the upper rectangle of the stage 1 polyhedral tiles, that give, for odd i, on their (left side) propagate-out$_{odd}$ pad $\widehat{E}'(b_{i-1}^{(t-1)}, i, t-1)$ encoding value $b_{i-1} = b_{i-1,1}$

and on their (left side) propagate-out$_{even}$ pad $\widehat{E}'(b_i^{(t-1)}, i, t-1)$ encoding value $b_i = b_{i,1}$. So, the polyhedral assembly provides $(b_1, \ldots, b_{n-1})$. Recall that we already have the monoid sum value b_n given by the base pad of the last (step $\log n$) previously placed rectanglular tiles. Thus, the assemblies together provide the entire solution $(b_1, \ldots, b_n)$ of the prefix computation problem. Again, this output can be constructed as a ssDNA in $O(1)$ BMC steps by a variety of known techniques given in subsection 4.2.

8 EMULATING SHUFFLE-EXCHANGE NETWORKS

8.1 Pair-wise Exchange using Unmediated Assembly

Given an even length n vector $(a_1, a_2, \ldots, a_{n-1}, a_n)$ of elements from any domain D, the *pair-wise exchange* problem is to form a vector $(a_2, a_1, \ldots, a_n, a_{n-1})$, that is exchange every odd-even pair. (This is an essential operation for emulating Shuffle-Exchange Networks.)

We can use nearly the same assembly construction as used for sequential prefix in Section 5 to solve this problem, but in this case we do not require control of the assembly.

We assume distinct DNA encoding functions E, E' and again assume the input ssDNA encodes the vector $(a_1, \ldots, a_n)$ as a sequence of unit length ssDNA words of the form $E(a_1, 1), \ldots, E(a_n, n)$. This input ssDNA is positioned into a straight line of length n (see again Figure 3), by the use of rigid framing techniques described in subsection 4.2.

Each tile is a 2×2 square as given in Figure 20. For each $a, a' \in D$ and $i \in \{1, \ldots, n\}$ where i is odd, there is a tile $\tau'(a, a', i)$ with:

- the *input* pad $\bar{E}(a, i)$, $\bar{E}(a', i+1)$ on the bottom side (with the intent to match the i,$(i+1)$-th pair of inputs),

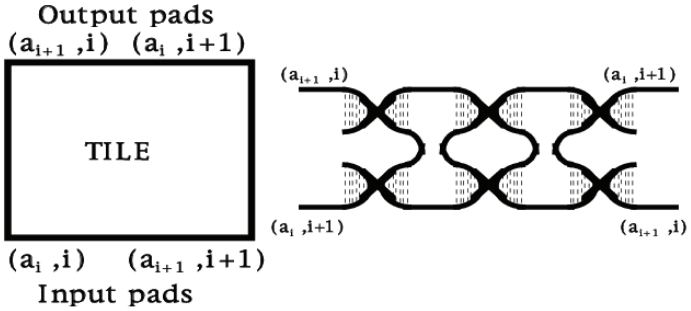

FIGURE 20
A square tile for pair-wise exchange and a DNA DX nano-structure for the tile.

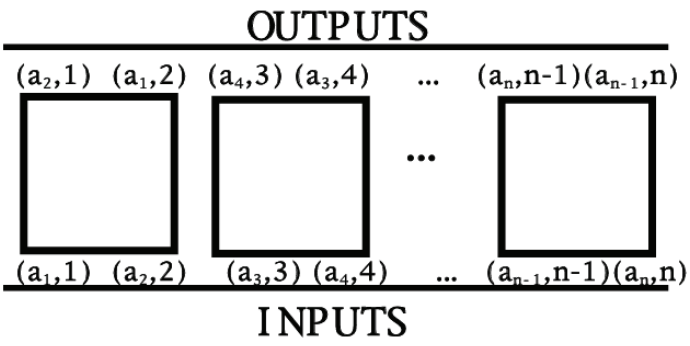

FIGURE 21
The 2-D assembly for pair-wise exchange.

- *output* pads $E'(a', i)$, $E'(a, i+1)$ on the top side (with the intent to output the i, $(i+1)$-th pair of outputs in reverse order).

We use unmediated assembly, adding all tiles at the same time. For each $i \in \{1, \ldots, n\}$ where i is odd, the tile $\tau'(a_i, a_{i+1}, i)$ is the unique tile that can be placed so its bottom side abuts and its pad matches the i, $(i+1)$-th pair of elements $E(a_i, i)$, $E(a_{i+1}, i+1)$ of the input. Its location and pad positions do not interfere with any other tiles at this time, and the tiling is unique.

The resulting $2D$ assembly is unique and is in the shape of a rectangular tiling (see Figure 21) of length n and height 2 containing tiles

$$\tau'(a_1, a_2, 1), \tau'(a_3, a_4, 3), \ldots, \tau'(a_{n-1}, a_n, n-1)$$

in this order, with the encoded input matched to the input pads on the bottom sides of these tiles and the output pads on the bottom sides of the tiles giving the required encoded output sequence

$\bar{E}(a_2, 1), \bar{E}(a_1, 2), \quad \bar{E}(a_4, 3), \bar{E}(a_3, 4), \quad \ldots, \bar{E}(a_n, n-1), \quad \bar{E}(a_{n-1}, n)$.

Again, this output can be constructed as a ssDNA in $O(1)$ BMC steps by a variety of well known techniques; see Subsection 4.2. (Note that this tiling assembly construction can be simplified somewhat by taking the time stamps mod 2, but the integer time stamps will in general be needed if we apply repeated pair-wise exchange and perfect shuffle operations.)

8.2 Perfect Shuffle Operations

Given an even length n vector $(a_1, a_2, \ldots, a_{n-1}, a_n)$ of elements from any domain D, the *perfect shuffle problem* is to form a vector $(a_1, a_{n/2+1}, a_2, a_{n/2+2}, \ldots, a_{n/2}, a_n)$, i.e., form a vector that shuffles the first

$n/2$ elements with the last $n/2$ elements. (This is another essential operation for emulating Shuffle-Exchange Networks.)

To solve the Perfect Shuffle problem we propose a linear size assembly, which will be unmediated and will not require multiple steps in the assembly.

We assume distinct DNA encoding functions E, E' and again assume the input ssDNA encodes the vector $(a_1, \ldots, a_n)$ as a sequence of ssDNA words of the form

$$E(a_1, 1), \ldots, E(a_n, n).$$

Without loss of generality, let position 1 be the $5'$ end of the input ssDNA, position n be the $3'$ end.

For technical reasons, we need to reverse the order of the encoding in the latter half of the input. By known recombinant DNA techniques we can reverse the latter part of the ssDNA containing the latter $n/2$ elements $E(a_{n/2+1}, n/2+1) \ldots E(a_n, n)$, by using the Watson-Crick complement, yielding a ssDNA with a sequence of unit length ssDNA words of the form

$$E(a_1, 1) \ldots E(a_{n/2}, n/2)\bar{E}(a_n, n)^R \ldots \bar{E}(a_{n/2+1}, n/2+1)^R,$$

where the superscript R denotes string reversal [3].

By known techniques (via circularization and appropriate use of restriction enzymes), ssDNA is then edited so as to increase its length to $n+2$, by insertion of a dummy (distinct from all the other strings used in DNA encodings) ssDNA just after the $(n/2)$-th element. The resulting ssDNA can be positioned in $2D$ into the shape of a rectangle (the dummy ssDNA was inserted to form the right side of this rectangle) with a missing left side (see Figure 22) of width $n/2$ and height 2 by the use of rigid framing techniques described in subsection 4.2 (see Figure 23).

[3] We can do the reversal (with refinements suggested by [121]) as follows:

- Ligate a hairpin region H to the 3' end.
- Use polymerase to extend the hairpin, and then denature; this yields $E(a_1, 1), \ldots E(a_n, n)\ H\ \bar{E}(a_n, n)^R \ldots\ \bar{E}(a_1, 1)^R$.
- Cut between $\bar{E}(a_{n/2+1}, n/2+1)^R\ \bar{E}(a_{n/2}, n/2)^R$, using an excess of complementary oligos to form the dsDNA target site for the restriction enzyme; this yields $E(a_1, 1), \ldots E(a_n, n)\ H\ \bar{E}(a_n, n)^R \ldots\ \bar{E}(a_{n/2+1}, n/2+1)^R$.
- Circularize and then cut between $E(a_{n/2}, n/2)\ E(a_{n/2+1}, n/2+1)$ and between $H\ \bar{E}(a_n, n)^R$, so that the longer DNA is $\bar{E}(a_n, n)^R \ldots\ \bar{E}(a_{n/2+1}, n/2+1)^R\ E(a_1, 1), \ldots E(a_{n/2}, n/2)$.
- Circularizing again, and cutting between $E(a_{n/2+1}, n/2+1)^R\ \ E(a_1, 1)$, we get $E(a_1, 1) \ldots E(a_{n/2}, n/2)\ \bar{E}(a_n, n)^R \ldots\ \bar{E}(a_{n/2+1}, n/2+1)^R$, as required.

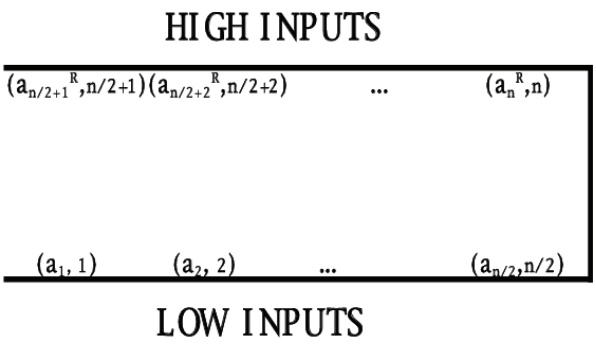

FIGURE 22
An ssDNA encoding input n-vector.

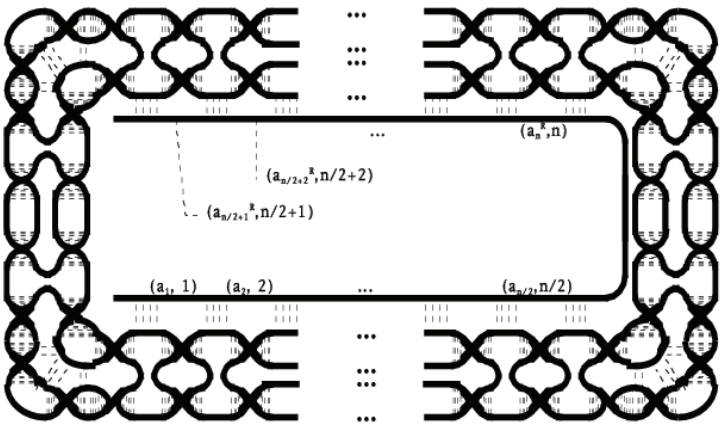

FIGURE 23
A DNA nano-structure for the rectangular frame.

Each tile in this case is a rectangle of width 1 and height 2 with an internal middle segment. The internal middle segment consists of two consecutive straight segments each of length 1 forming a (raised upsidedown) V, and bridging between the left and right sides of the tile, as given in Figure 24.

For each $a, a' \in D$ and $i \in \{1, \ldots, n/2\}$, there is a tile $\tau''(a, a', i)$ with:

- a 1×2 rectangle with
  - the $input_{low}$ pad $\bar{E}(a, i)$ on the bottom side (with the intent to match the i-th input a_i) and

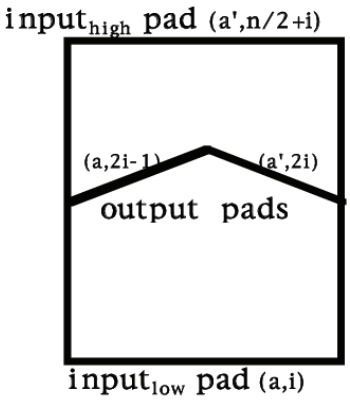

FIGURE 24
A square tile for perfect shuffle, with middle segment used for output.

  - the $input_{high}$ pad $E(a', n/2 + i)^R$ on the top side (with the intent to match the $(n/2 + i)$-th input $a_{n/2+i}$), and
- an internal middle section, bridging between the left and right sides, with an *output* pad
  $\overline{E}'(a, i)\overline{E}'(a, n/2 + i)$ (with the intent to provide the outputs).

We will use unmediated assembly, adding all tiles at the same time. For each $i \in \{1, \ldots, n/2\}$, the tile $\tau''(a_i, a_{n/2+i}, i)$ is the unique tile that can be placed so the bottom side abuts (and its $input_{low}$ pad matches) the i-th element $E(a_i, i)$ of the input and the top side abuts (and its $input_{high}$ pad matches) the $(n/2 + i)$-th element $E(a_{n/2+i}, n/2 + i)$ of the input. The location of this tile does not interfere with the placement of any other tile.

The resulting $2D$ assembly (see Figure 25) is unique and is a rectangle of length n and height 2, containing tiles $\tau''(a_1, a_{n/2+1}, 1)$,

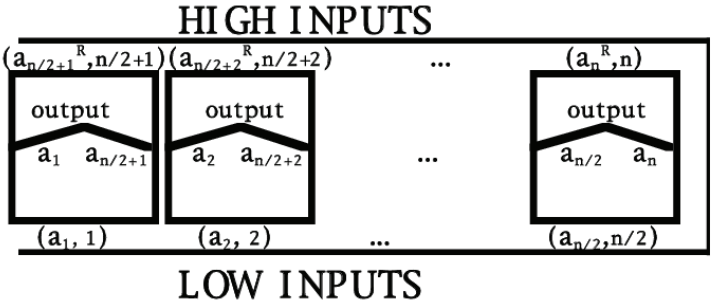

FIGURE 25
2D assembly for perfect shuffle. The middle segments give output.

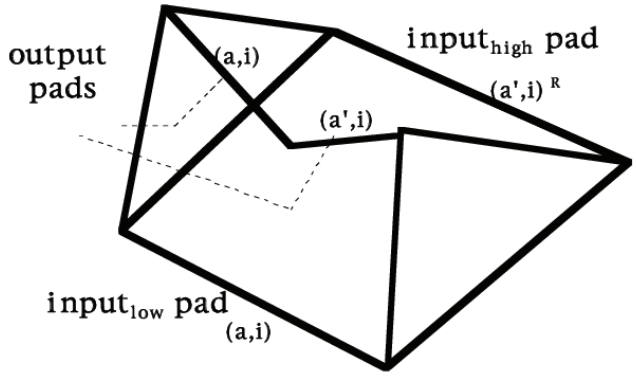

FIGURE 26
Alternative 3D polyhedral tile for perfect shuffle.

$\tau''(a_2, a_{n/2+2}, 2), \ldots, \tau''(a_{n/2}, a_n, n/2)$ in this order, with the inputs matched to pads on the bottom and top sides of these tiles and the output pads of the middle segments of the tiles giving the required encoded output sequence $\bar{E}'(a_1, 1)$, $\bar{E}'(a_{n/2}, 2), \bar{E}'(a_2, 3)$, $\bar{E}'(a_{n/2+1}, 4), \ldots, \bar{E}'(a_{n/2}, n-1)$, $\bar{E}'(a_n, n)$. This output can be constructed as a ssDNA in $O(1)$ BMC steps by a variety of known techniques; see subsection 4.2. For example, we may insert some matching ssDNA to the ends of the middle segments, ligate the consecutive middle segments to form a single ssDNA, then cut out, by restriction enzymes, all but this middle segment ssDNA containing the output pads. (Note that again this tiling assembly construction can be simplified somewhat by taking the time stamps mod 2, but the integer time stamps will in general be needed if we repeatedly apply pair-wise exchange operations in combination with these perfect shuffle operations.)

To further facilitate readout, each tile can be alternatively defined to be $3D$ polyhedron in the shape given in Figure 26, resembling pup tent, with the same rectangular base as the 1×2 tiles described above, and a top segment with an output pad consisting of two consecutive straight segments each of length 1 forming a V, and bridging between the left and right sides of the pup tent. These $3D$ polyhedron tiles might be fabricated the use of three-axis multiple DX molecules i.e., a DX with one additional double helix axis stacked on top.

8.3 Shuffle-Exchange Networks

A *shuffle-exchange network* (Stone [112]) is a graph with node set $\{1, \ldots, n\}$ and with a set of edges that effect both

1. a perfect shuffle permutation on the nodes, and
2. a pair-wise exchange operation on the nodes.

See Ullman [113] and Leighton [49] for discussion of the the use of shuffle-exchange network for emulations of other networks such as the butterfly and CCC networks and see Schawbe [96] for a proof of the computational equivalence of hypercube-derived networks, such as the butterfly network.

In Subsections 8.1, 8.2 we have given constant assembly depth unmediated LP-BMC algorithms for executing perfect shuffle and pair-wise exchange on length n vectors encoded into DNA. By simple edits (insertion of short dummy sequences to and reversal of subsequences) to the DNA as described in these subsections, the output encoding (say from a shuffle-exchange step) is made compatible with the input encoding required for the following step (say a pairwise exchange). By the definition of a shuffle-exchange network, it follows that these LP-BMC algorithms can be used to emulate a shuffle-exchange network, in constant assembly depth. Furthermore, we can combine each shuffle-exchange with a computational step on consecutive odd-even pairs of data, using one step of the step-wise tiling assembly technique described in Section 4.

8.4 Normal Parallel Algorithms

Normal parallel algorithms are a well-known class of parallel algorithms; for details see Ullman [113] and Leighton [49] (note they are also sometimes called *ascend-descend* algorithms). A normal parallel algorithms can be executed (see details in [49, 112, 113]) on an n-vector of data using n processors executing in parallel in $O(\log n)$ stages. Each stage consists of the following parallel operations:

- a shuffle-exchange permutation and
- parallel execution of a computational step (costing each processor $O(1)$ work) on each of the $n/2$ consecutive odd-even pairs of data.

We can do each of these stages using LP-BMC within constant assembly depth, as follows:

- The techniques of Subsection 8 executes the shuffle-exchange permutation by LP-BMC in constant assembly depth.
- The parallel execution of an $O(1)$ work computational step on $n/2$ consecutive odd-even pairs of data, can be done by LP-BMC in constant assembly depth by emulation of one time step of an n processor $1D$ array, using a step-wise tiling assembly as given in Subsection 4.1. In this assembly, which will have constant depth, we use again square tiles, each with two input pads which match to consecutive odd-even pairs of data, and two output pads providing these computed values to the next stage (we can apply the constant time readout methods of Subsection 4.2, to provide these computed values to the next stage).

Well known example applications of normal parallel algorithms are

- the $O(\log n)$ time, n processor bitonic merge algorithm of Batcher [9], which merges two length n vectors using a normal parallel algorithm and
- the $O(\log^2 n)$ time, n processor [9] bitonic sort algorithm, which sorts a vector of length n using $O(\log n)$ normal algorithm passes.

Assuming that the elements to be operated on are integers of b bits, we can use 2^{2b} tiles to effect a key comparison in one step of tiling. Thus, the bitonic merge and sort algorithms can be executed by our LP-BMC methods with these same assembly depth bounds and using $O(n)$ size assemblies.

8.5 Executing General Permutations and Parallel Evaluation of Circuits

A known parallel algorithm, known as the Beneš network [15], allows the execution of an (arbitrary) fixed permutation of n data elements by use of a normal parallel algorithm (Waksman [114]). This requires $O(\log n)$ LP-BMC steps by our previous results in Subsection 8.4 for normal parallel algorithms.

This also implies LP-BMC can do parallel evaluation of a fixed, bounded degree Boolean circuit of size n and of depth d. The LP-BMC assembly will have size $O(n)$ and depth $O(d \log n)$. To do this, we evaluate the circuit in stages $t = 1, \ldots, d$ where in stage t we:

- apply, in $O(\log n)$ LP-BMC steps, the Beneš technique (precomputed for the permutations of each level of the fixed circuit) to move the data required for inputs to each node of the circuit of level t,
- use the techniques of Subsection 4.1 to do by LP-BMC in constant assembly depth, the parallel Boolean operations required to evaluate each level t node of the circuit (again, this is done by emulation of an n processor $1D$ array, using a constant number of steps of our step-wise $2D$ assembly given in Subsection 4.1), and
- after assembly, we apply the constant assembly depth methods of Subsection 4.2, to provide these node evaluation values to the next level of the circuit.

9 CONCLUSION

We summarize the advantages of our step-wise technique for LP-BMC:

- it requires no thermal cycling,
- it results in no blockages, and

- it provides greater flexibility and control over the chemical conditions under which assembly occurs, as compared to unmediated self-assembly, and
- whereas the kinetic models of Winfree [121] suggest that unmediated self-assembly should proceed as slowly as possible, in step-wise self assembly, we may be able to speed up considerably by washing over high concentrations of tiles.

There are a number of insights provided by this paper that are new: (1) different geometrical sizes and shapes of tiles can have a very significant effect upon computational speed, (2) DNA folding over on itself using rigid frames can allow communication between memory bits in long strands of DNA, thus allowing fast circuit simulation, and (3) stepwise methods in which the assembly is partially deconstructed after completion (e.g. the rigid frame and tiles are removed, keeping only the output strand), allowing for considerably greater flexibility in the computation.

Our main constructive results provided assemblies which are compact ($O(n)$ size) with small $O(\log n)$ assembly depth for a number of fundamental problems, including prefix computation, integer addition, list merging, fixed permutations, and many other normal parallel algorithms. This small depth and size may decrease probability of errors in the assembly process.

Subsequent work in self assembly is described in [58, 59, 67, 123, 124] and surveyed in [45, 83, 84]

ACKNOWLEDGMENTS.

N. C. Seeman gave us many very valuable and insightful suggestions on our assemblies and self-assembly based nanofabrication techniques in general, particularly with respect to the geometry and self-assembly of our DNA tiles and assembly frame, and also informed us of the useful biochemistry, e.g. methods for 3-way pairing of DNA strands. E. Winfree gave helpful suggestions on many aspects of this paper, for which we are very grateful, including our lower bound results for unmediated assembly, the use of Boltzman energy functions, time stamping modulo 2, methods for fabrication of $3D$ polyhedron tiles via multiple DX molecules, a refinement of our reversal method for segments of DNA strands, and further enumerating the potential advantages of our step-wise techniques. Also, A. Gehani, T. LaBean, U. Majumder, S. Sahu, and P. Yin provided many valuable suggestions for improvements to this paper, for which we are also very grateful. K. Redding provided substantial aid in preparation of the paper. Thanks to my father A. Reif, a biochemist who in my early years provided access to his lab, thus providing me with an introduction to basic biological lab techniques.

REFERENCES

[1] L. Adleman, *Molecular Computation of Solution to Combinatorial Problems*, Science, **266**, 1021–1024, (1994).

[2] L. Adleman, *On Constructing a Molecular Computer*, Proceedings of the First Workshop on DNA Based Computers, 1995. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 27, Edited by: Richard J. Lipton and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1996). ISBN-10: 0-8218-0973-3 ISBN-13: 978-0-8218-0973-0.

[3] L. M. Adleman, P. W. K. Rothemund, S. Roweis, E. Winfree, *On Applying Molecular Computation To The Data Encryption Standard*, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, June 10-12, 1996, pp 217-235. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-0 Published as *On Applying Molecular Computation to the Data Encryption Standard*, Journal of Computational Biology, 6 (1). pp. 53–63 (1999). ISSN 1066-5277.

[4] A. P. Alivisatos, K. P. Johnsson, X. Peng, T. E. Wilson, C. J. Loweth, M. P. Bruchez Jr., P. G. Schultz, *Organization of 'nanocrystal molecules' using DNA*, Nature, bf 382, 609–611, August 1996.

[5] J.-T. Amenyo, *Mesoscopic computer engineering: Automating DNA-based molecular computing via traditional practices of parallel computer architecture design*, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, June 10-12, 1996, pp 217-235. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-0

[6] M. Amos, A. Gibbons, and D. Hodgson, *Error-resistant Implementation of DNA Computations*, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, June 10-12, 1996, pp. 87–101. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-0

[7] M. Arita and M. Hagiya, *Joining and Rotating Data with Molecules*, ICEC'97 Special Session on DNA Based Computation, IEEE International Conference on Evolutionary Computation, Indiana, (Apr 13-16 1997).

[8] E. Bach, A. Condon, E. Glaser, and C. Tanguay, *Improved Models and Algorithms for DNA Computation*, Proc. 11th Annual IEEE Conference on Computational Complexity, Submitted (by invitation) to: J. Computer and System Sciences, pp. 243–248, May 1996. ISBN: 0-7803-3949-5

[9] K. Batcher *Sorting Networks and their applications*, Spring Joint Computer Conference, **32**, 307–314, AFIPS Press, Montvale, N. J. (1968).

[10] E. B. Baum, *How to build an associative memory vastly larger than the brain*, Science, 268(5210), pp. 583–585 (April 28, 1995.

[11] E. B. Baum, *DNA Sequences Useful for Computation*, pp. 122–127, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, June 10-12, 1996, pp. 87–101. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-0

[12] D. Beaver, *Factoring: The DNA Solution*, Proceedings of Advances in Cryptology - ASIACRYPT '94, 4th International Conference on the Theory and Applications of Cryptology, Lecure Notes in Computer Science, Springer Verlag, Vol. 917, Wollongong, Australia, pp. 419–423, (November 28 - December 1, 1994). DOI: 10.1007/BFb0000453

[13] D. Beaver, *A Universal Molecular Computer*, revised as *Molecular Computing*, Proceedings of the First Workshop on DNA Based Computers, 1995. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 27, Edited by: Richard J. Lipton and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1996). ISBN-10: 0-8218-0973-3 ISBN-13: 978-0-8218-0973-0.

[14] D. Beaver, *Computing with DNA*, J. Comp. Biol., **2**, 1–7, (1995).

[15] V. Beneš, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, New York, NY (1965).

[16] R. Berger *The Undecidability of the Domino Problem*, Memoirs of the American Mathematical Society, **66**, (1966).

[17] D. Boneh, C. Dunworth, R. Lipton, *Breaking DES Using a Molecular Computer*, Proceedings of the First Workshop on DNA Based Computers, 1995. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 27, Edited by: Richard J. Lipton and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1996). ISBN-10: 0-8218-0973-3 ISBN-13: 978-0-8218-0973-0.

[18] D. Boneh, C. Dunworth, R. Lipton, J. Sgall, *On the Computational Power of DNA*, Princeton CS Tech-Report number CS-TR-499-95, (1995). Also published in Discrete Applied Mathematics, Special Issue on Computational Molecular Biology, Vol. 71 (1996), pp. 79–94.

[19] D. Boneh, R. Lipton, *Making DNA Computers Error Resistant*, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, June 10-12, 1996, pp 102–110. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-0

[20] C. Brooks, M. Karplus, M. Pettitt, *Proteins, A Theoretical Perspective of Dynamics, Structure & Thermodynamics*, John Wiley & Sons,

[21] J. R. Buchi, *Turing Machines and the Entscheidungsproblem*, Mathematiche Annalen, **148**, 201–213, (1962).

[22] W. Cai, A. Condon, R. M. Corn, Z. Fei, T. Frutos, E. Glaser, Z. Guo, M. G. Lagally, Q. Liu, L. M. Smith, and A. Thiel, *The Power of Surface-Based Computation*, Proc. First International Conference on Computational Molecular Biology (RECOMB97), January 1997.

[23] W. Cai, E. Rudkevich, Z. Fei, A. Condon, R. Corn, L. M. Smith, M. G. Lagally, *Influence of Surface Morphology in Surface-Based DNA Computing*, Submitted to the 43rd AVS National Symposium, Abstract No. BI+MM-MoM10, (1996).

[24] J.-H. Chen, M. E. A. Churchill, T. D. Tullius, N. R. Kallenbach, N. C. Seeman, *Construction and Analysis of Monomobile DNA Junctions*, Biochemistry, **27**, (1988).

[25] J. Chen, C. A. Rauch, J. H. White, P. T. Englund, N. R. Cozzarelli, em The Topology of the Kinetoplast DNA Network, rm Cell, **80**, 61–69, January 1995.

[26] B. Crandall and J. Lewis (eds.), *Nanotechnology*, MIT Press, (1992).

[27] R. Deaton, R. C. Murphy, M. Garzon, D. R. Franceschetti, and S. E. Stevens, Jr., *Good encodings for DNA-based solutions to combinatorial problems*, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, June 10-12, 1996, Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer

Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-0

[28] R. Deaton, R. C., Murphy, J. A. Rose, M. Garzon, D. R. Franceschetti, and S. E. Stevens, Jr., *A DNA Based Implementation of an Evolutionary Search for Good Encodings for DNA Computation*, ICEC'97 Special Session on DNA Based Computation, Indiana, April 1997.

[29] A. L. Delcher, L. Hood, R. M. Karp, *Report on the DNA/Biomolecular Computing Workshop*, June 1996.

[30] M. D. Distefano, P. B. Dervan *Ligand promoted dimerization of oligonucleotides binding cooperatively to DNA*, J.Am.Chem.Soc., **114**, 11006-11007, (1992).

[31] R. Drmanac, S. Drmanac, Z. Strezoska, T. Paunesku, I. Labat, M. Zeremski, J. Snoddy, W. K. Funkhouser, B. Koop, L. Hood, R. Crkenjakov, *DNA Sequence Determination by Hybridization: A Strategy for Efficient Large-Scale Sequencing*, Science, **260**, 1649–1652, (1993).

[32] S. M. Du and N. C. Seeman, *The Synthesis of a DNA Knot Containing both Positive and Negative Nodes*, J. Am. Chem. Soc., **114**, 9652–9655, (1992).

[33] S. M. Du, S. Zhang and N. C. Seeman, *DNA Junctions, Antijunctions and Mesojunctions*, Biochem., **31**, 10955–10963, (1992).

[34] Eisenberg and Crothers, *Physical Chemistry with applications to the life sciences*.

[35] M. Engler and C. Richardson, *The Enzyme*, (P. Boyer, ed.) Academic Press, 3–29, (1982).

[36] R. Feynman, Minaturization (D. Gilbert, ed.), Reinhold, 282–296, (1961).

[37] S. Fortune and J. Wyllie, *Parallelism in random access machines*, Proc. 10th Annual ACM S.T.O.C., San Diego, CA, 114–118, (1978).

[38] T.-J. Fu and N.C. Seeman, *DNA Double Crossover Structures*, Biochemistry, **32**, 3211–3220, (1993).

[39] M. R. Garey and D. S. Johnson *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H Freeman and Company, page 257, (1979).

[40] M. R. Garey, D. S. Johnson, and C. H. Papadimitriou, unpublished manuscript, (1977).

[41] J. M. Gray, T. G. Frutos, A. M. Berman, A. E. Condon, M. G. Lagally, L. M. Smith, R. M. Corn, *Reducing Errors in DNA Computing by Appropriate Word Design*, University of Wisconsin, Department of Chemistry, October 9, 1996.

[42] S. Grunbaum, Branko, and G. C. Shepard *Tilings and Patterns*, H Freeman and Company, **Chapter 11**, (1987).

[43] F. Guarnieri, and C. Bancroft, *Use of a Horizontal Chain Reaction for DNA-Based Addition*, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, June 10-12, 1996, Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-0

[44] Guarnieri, F., Fliss, M., and C. Bancroft, *Making DNA add*, Add. Science, **273**, 220–223, (1996).

[45] Hieu Bui, Harish Chandran, Sudhanshu Garg, Nikhil Gopalkrishnan, Nikhil Gopalkrishnan, Reem Mokhtar, Tianqi Song and John H Reif, DNA Computing, Chapter in Computing Handbook, Volume I: Computer Science and Software Engineering, Section 3: Architecture and Organization, Edited by Teofilo F. Gonzalez, Taylor & Francis Group, (2013).

[46] J. JáJá, *An Introduction to Parallel Algorithms*, Addison Wesley, (1992).

[47] R. E. Ladner, and M. J. Fischer, *Parallel Prefix Computation*, JACM, **27(4)**:831–838, (1980).

[48] Thomas H. LaBean, Hao Yan, Jens Kopatsch, Furong Liu, Erik Winfree, John H. Reif and Nadrian C. Seeman, *The construction, analysis, ligation and self-assembly of DNA triple crossover complexes*, Journal of American Chemistry Society(JACS) 122, pp. 1848–1860 (2000).

[49] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufmann Press, San Mateo, CA, Chapter 3, (1992).

[50] H. R. Lewis and C. H. Papadimitriou *Elements of the Theory of Computation*, Prentice-Hall, pages 296–300 and 345–348 (1981).

[51] N. Jonoska, S. A. Karl, *A Molecular Computation of the Road Coloring Problem*, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, June 10-12, 1996. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-0

[52] N. Jonoska, S. A. Karl, *Ligation Experiments in Computing with DNA*, ICEC'97 Special Session on DNA Based Computation, Indiana, April 1997.

[53] N. Jonoska and S. A. Karl *Creating 3-Dimensional Graph Structures with DNA*, 3nd Annual Meeting on DNA Based Computers, University of Pens., (June 1997).

[54] L. F. Landweber and R. Lipton *DNA 2 DNA Computations: A Potential 'Killer App'?*, 3nd Annual Meeting on DNA Based Computers, University of Pens., (June 1997).

[55] P. Kaplan, G. Cecchi, and A. Libchaber, *DNA based molecular computation: Template-template interactions in PCR*, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, June 10-12, 1996, Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-0

[56] S. A. Kurtz, S. R. Mahaney, J. S. Royer, J. Simon, *Active Transport in Biological Computing*, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, June 10-12, 1996, Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-0

[57] S. A. Kurtz, S. R. Mahaney, J. S. Royer, J. Simon, *Biological Computing*, Complexity theory retrospective II pp. 179–195, Springer-Verlag, New York, NY (1997). ISBN:0-387-94973-9

[58] T. H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J. H. Reif and N. C. Seeman, The construction, analysis, ligation and self-assembly of DNA triple crossover complexes, J. Am. Chem. Soc. 122, 1848-1860 (2000). www.cs.duke.edu/∼reif/paper/DNAtiling/tilings/JACS.pdf

[59] T. H. LaBean, E. Winfree, J. H. Reif, Experimental Progress in Computation by Self-Assembly of DNA Tilings, Proceeding of DNA Based Computers V: Cambridge, MA, June 14-16, 1999. Published in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 54, edited by Erik Winfree and D.K. Gifford, American Mathematical Society, Providence, RI, pp. 123–140, (2000). http://www.cs.duke.edu/∼-thl/tilings/labean.ps

[60] M. G. Lagoudakis, T. H. LaBean, 2D DNA Self-Assembly for Satisfiability, 5th International Meeting on DNA Based Computers(DNA5), MIT, Cambridge, MA, (June, 1999). DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol.44, American Mathematical Society, ed. E. Winfree, (1999).

[61] X.J. Li, X.P. Yang, J. Qi, and N.C. Seeman, *Antiparallel DNA Double Crossover Molecules as Components for Nanoconstruction*, J. Am. Chem. Soc., **118**, 6131–6140, (1996).

[62] R. Lipton, *Speeding Up Computations via Molecular Biology*, Proceedings of the First Workshop on DNA Based Computers, 1995. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 27, Edited by: Richard J. Lipton and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1996). ISBN-10: 0-8218-0973-3 ISBN-13: 978-0-8218-0973-0.

[63] R. J. Lipton, *DNA Solution of Hard Computational Problems*, Science, **268**, 542–545, (1995).

[64] R. J. Lipton. *DNA Computations Can Have Global Memory*, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, June 10-12, 1996, Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-0

[65] Q. Liu, Z. Guo, A. E. Condon, R. M. Corn, M. G. Lagally, and L. M. Smith, *A Surface-Based Approach to DNA Computation,*, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, June 10-12, 1996, 206–216. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-0. Published in J Comput Biol., 5(2): pp. 255–67, (1998). PMID: 9672831.

[66] C. Mao and N. C. Seeman, *Construction of Borromean Rings from DNA*, Nature, **386**(6621), 137–138, (March, 1997).

[67] C. Mao, T. H. LaBean, J. H. Reif, and N. C. Seeman, Logical Computation Using Algorithmic Self-Assembly of DNA Triple-Crossover Molecules, Nature, vol. 407, pp. 493–495 (Sept. 28 2000); C. Erratum: Nature 408, 750–750 (2000). www.cs.duke.edu/∼reif/paper/SELFASSEMBLE/AlgorithmicAssembly.pdf

[68] J. McCammon, S. Harvey, *Dynamics of Proteins and Nucleic Acids*, Cambridge University Press, (1987).

[69] R. Merkle, Nanotechnology **4** 21, (1993).

[70] G. Miller, J. H. Reif, *Parallel Tree Contraction and its Application*, 26th Annual IEEE Symposium on Foundations of Computer Science, Portland, OR, 478–489, October 1985. Also published as *Parallel Tree Contraction Part I: Fundamentals*, Advances in Computing Research, **5**, 47–72, (1989), and *Parallel Tree Contraction Part II: Further Applications*, SIAM Journal on Computing, **20:6**, 1128–1147, (1991).

[71] K. U. Mir, *A Restricted Genetic Alphabet for DNA Computing*, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, June 10-12, 1996, Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-0

[72] C. A. Mirkin, R. L. Letsinger, R. C. Mucic, J. J. Storhoff, *A DNA-based Method for Rationally Assembling Nanoparticles Into Macroscopic Materials*, Nature, **382**, 607–611, August 1996.

[73] J. E. Mueller, S. M. Du and N. C. Seeman, *The Design and Synthesis of a Knot from Single-Stranded DNA*, J. Am. Chem. Soc., **113**, 6306–6308, (1991).

[74] R. C. Murphy, R. Deaton, D. R. Franceschetti, S. E. Stevens, *A New Algorithm for DNA Based Computation*, ICEC'97 Special Session on DNA Based Computation, Indiana, April 1997.

[75] M. Ogihara, A. Ray, *Simulating Boolean circuits on a DNA computer*, 1st Annual International Conference On Computational Molecular Biology (RECOMB97), Santa Fe, New Mexico, January 1997.

[76] R. Old and S. Primrose, *Principles of Gene Manipulation, An Introduction to Genetic Engineering*, Blackwell Scientific Publications, Fifth Edition, (1994).

[77] G. E. Plum and D. S. Pilch, *Nucleic Acid Hybridization: Triplex Stability and Energetics.*, Annu. Rev. Biophys. Biomol. Struct. , **24:**, 319–350, (February 1997).

[78] R. Pool, *Dr. Tinkertoy*, Discover, **18:2**, 50–57, (February 1997).

[79] J. Qi, X. J. Li, X. P. Yang, and N. C. Seeman, *Ligation of triangles built from bulged 3-arm DNA branched junctions*, J. Am. Chem. Soc., **v118:26**, 6121–6130, (July, 1996).

[80] J. H. Reif, (ed.), *Synthesis of Parallel Algorithms*, Morgan Kaufmann, (1993).

[81] J. H. Reif,, *Parallel Molecular Computation: Models and Simulations*, Seventh Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA95), ACM, Santa Barbara, 213–223, June 1995. Published in **Algorithmica**, special issue on Computational Biology, 25:142-175, 1998. A PDF version of this paper is at http://www.cs.duke .edu/∼reif/paper/paper.html.

[82] J. H. Reif,, *Paradigms for Biomolecular Computation*, First International Conference on Unconventional Models of Computation, Auckland, New Zealand, January 1998. Published in *Unconventional Models of Computation*, edited by C.S. Calude, J. Casti, and M.J. Dinneen, Springer Publishers, January 1998, pp 72-93. A postscript version of this paper is at http://www.cs.duke.edu/∼reif/paper/paradigm.ps.

[83] J. H. Reif, H. Chandran, N. Gopalkrishnan, and T. LaBean, Self-assembled DNA Nanostructures and DNA Devices. Invited Chapter 14, Nanofabrication Handbook (Edited by Stefano Cabrini and Satoshi Kawata), pages 299-328, CRC Press, Taylor and Francis Group, New York, NY, ISBN13:9781420090529, ISBN10: 1420090526 (2012).

[84] J. H. Reif, T. H. LaBean, and N. C. Seeman, Challenges and Applications for Self-Assembled DNA Nanostructures, Proc. Sixth International Workshop on DNA-Based Computers, Leiden, The Netherlands, June 13-17, 2000. Published in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Edited by A. Condon and G. Rozenberg. Lecture Notes in Computer Science, Springer-Verlag, Berlin Heidelberg, vol. 2054, 2001, pp. 173–198. http://www.cs.duke.edu/∼reif/paper/ SELFASSEMBLE/selfassemble.ps

[85] J. H. Reif and T. H. LaBean,Computationally Inspired Biotechnologies: Improved DNA Synthesis and Associative Search Using Error-Correcting Codes and Vector-Quantization, Sixth International Meeting on DNA Based Computers (DNA6), Leiden, The Netherlands (June 2000). DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Edited by A. Condon and G. Rozenberg. Lecture Notes in Computer Science, Springer-Verlag, Berlin Heidelberg, vol. 2054, pp. 145–172 (2001). www.cs.duke.edu/∼reif/paper/Error-Restore/Error-Restore.ps

[86] J. H. Reif, Sudheer Sahu, Peng Yin, Compact Error-Resilient Computational DNA Tiling Assemblies, Tenth International Meeting on DNA Based Computers (DNA10), Milano, Italy, June 7-10, 2004. Lecture Notes in Computer Science (Edited by C Ferretti, G. Mauri and C. Zandron), Vol. 3384, Springer-Verlag, New York, (2005), pp. 293–307. Published as an invited chapter in text "Nanotechnology: Science and Computation", Springer Verlag series in Natural Computing (edited by J. Chen; N. Jonoska and G. Rozenberg), Springer-Verlag Berlin, Germany, pp. 79–104, (2006).

[87] R. J. Roberts, *Restriction and modification enzymes and their recognition sequences*, Gene, **4**, 183–194, (1978).

[88] R. M. Robinson *Undecidablility and Nonperiodicity for Tilings of the Plane*, Inventiones Mathematicae, **12**, 177–209, (1971).

[89] P. W. K. Rothemund, *A DNA and restriction enzyme implementation of Turing Machines*, Proceedings of the First Workshop on DNA Based Computers, 1995. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 27, Edited by: Richard J. Lipton and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1996). ISBN-10: 0-8218-0973-3 ISBN-13: 978-0-8218-0973-0.

[90] P. W. K. Rothemund, N. Papadakis, E. Winfree Algorithmic self-assembly of DNA Sierpinski triangles, PLoS Biology, Vil. 2(12): e424 (2004).

[91] D. Rooβ and K.W. Wagner, *On the power of Bio-Computers*, unpublished manuscript.

[92] S. Roweis, E. Winfree, R. Burgoyne, N. V. Chelyapov, M. F. Goodman, P. W. K. Rothemund, L. M. Adleman, *A Sticker Based Architecture for DNA Computation*, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, June 10-12, 1996, 1–29. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-06. Published in Journal of Computational Biology, 5(4): 615–629, 1998.

[93] G. Rozenberg, A. Salomaa, ICEC'97 Special Session on DNA Based Computation, Indiana, April 1997.

[94] H. Rubin, *Looking for the DNA killer app.*, Nature, **3**, 656–658, (1996).

[95] J. Sambrook, E. Fritsch, T. Maniatis, *Molecular Cloning*, Cold Spring Harbor Lab, NY, (1989).

[96] E. Schawbe *On the computational equivalence of hypercube-drived networks*, Proceeding of the 2nd Annual Symposium on Parallel Algorithms and Architectures, 388–397 (1990).

[97] N. C. Seeman, *Nucleic Acid Junctions and Lattices*, J. Theor. Biol., **99**, 237–247, (1982).

[98] N. C. Seeman, *personal communication to J.H. Reif*, (1997).

[99] N. C. Seeman, *Macromolecular Design, Nucleic Acid Junctions and Crystal Formation*, Journal of Biomolecular Structure and Dynamics, **3**, 1–34, (1985).

[100] N. C. Seeman and J. Chen, *Synthesis from DNA of a molecule with the connectivity of a cube*, Nature, **350**, 631–633, (1991).

[101] N. C. Seeman, J. Chen, S. M. Du, John E. Mueller, Yuwen Zhang, Tsu-Ju Fu, Yinli Wang, Hui Wang, Siwei Zhang, *Synthetic DNA knots and catenanes*, New Jour. of Chemistry, **17**, 739–755, (1993).

[102] N. C. Seeman, J.-H. Chen, N. R. Kallenbach, *Gel electrophoretic analysis of DNA branched junctions*, Electrophoresis, **10**, 345–354, (1989).

[103] N. C. Seeman, J. Qi, X. Li, X. Yang, N. B. Leontis, B. Liu, Y. Zhang, S. M. Du, and J. Chen, *The control of DNA structure: From topological modules to geometrical modules*, Modular Chemistry, J. Michl, ed., Kluwer, To appear, (1996).

[104] N. C. Seeman, H. Wang, B. Liu, J. Qi, X. Li, X. Yang, F. Liu, W. Sun, Z. Shen, R. Sha, C. Mao, Y. Wang, S. Zhang, T.-J. Fu, S. Du, J. E. Mueller, Y. Zhang, and J. Chen, *The Perils of Polynucleotides: The Experimental Gap Between the Design and Assembly of Unusual DNA Structures*, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton University, June 10-12, 1996, pp. 215–233. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-0

[105] N. C. Seeman, Y. Zhang, and J. Chen, *DNA nanoconstructions*, J. Vac. Sci. Technol., **12:4**, 1895–1905, (1994).

[106] N. C. Seeman, Y. Zhang, S. M. Du, and J. Chen, *Construction of DNA polyhedra and knots through symmetry minimization*, Supramolecular Stereochemistry, J. S. Siegel, ed., 27–32, (1995).

[107] N. C. Seeman, Y. Zhang, S. Du, H. Wang, J. E. Mueller, and J. Chen, *The control of DNA structure and topology: An overview*, Mat. Res. Soc. Symp. Proc., **356**, 57–66, (1994).

[108] J. Setubal and J. Meidanis *Introduction to Computational Molecular Biology*, PWS Pub. Co., Chapt 9, (1997).

[109] R. Sinden, *DNA Structure and Function*, Academic Press, (1994).

[110] L. M. Smith, *Automated Synthesis and Sequence Analysis of Biological Macromolecules*, Anal. Chem., **60**, 381A–390A, (1988).

[111] W. Smith, A. Schweitzer, *DNA Computers in Vitro and Vivo*, Proceedings of the First Workshop on DNA Based Computers, 1995. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 27, Edited by: Richard J. Lipton and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1996). ISBN-10: 0-8218-0973-3 ISBN-13: 978-0-8218-0973-0.

[112] H. S. Stone *Parallel Processing with the perfect shuffle*, IEEE Trans. on Computers, **C-20**:2, 153–161 (1971).

[113] J. Ullman, *Computational Aspects of VLSI*, Computer Science Press, (1984), Chapter 6.

[114] A. Waksman *A Permutation Network*, JACM, **15(1)**,159–163 (1968).

[115] H. Wang, *Proving Theorems by Pattern Recognition*, Bell System Technical Journal, **40**, 1–141, (1961).

[116] H. Wang, R. J. Di Gate, and N. C. Seeman, *An RNA Topoisomerase*, Proc. Nat. Acad. Sci. (USA), **93**, 9477–9482, (1996).

[117] J. Watson, M. Gilman, J. Witkowski, M. Zoller, *Recombinant DNA (2nd ed.)*, Scientific American Books, W.H. Freeman and Co., (1992).

[118] J. Watson, N. Hoplins, J. Roberts, *et al.*, *Molecular Biology of the Gene*, Benjamin/Cummings, Menlo Park, CA, (1987).

[119] E. Winfree, *Complexity of Restricted and Unrestricted Models of Molecular Computation*, Proceedings of the First Workshop on DNA Based Computers, 1995. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 27, Edited by: Richard J. Lipton and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1996). ISBN-10: 0-8218-0973-3 ISBN-13: 978-0-8218-0973-0.

[120] E. Winfree, *On the computational power of DNA annealing and ligation*, Proceedings of the First Workshop on DNA Based Computers, 1995. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 27, Edited by: Richard J. Lipton and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1996). ISBN-10: 0-8218-0973-3 ISBN-13: 978-0-8218-0973-0.

[121] E. Winfree, communication to J. H. Reif (June, 1997).

[122] Winfree, E. and R. Bekbolatov, Proofreading tile sets: Error correction for algorithmic self-assembly, In DNA Based Computers 9, volume 2943 of LNCS, pp. 126–144, (2004).

[123] E. Winfree, F. Liu, L. A. Wenzler, N. C. Seeman, *Design and Self-Assembly of Two Dimensional DNA Crystals*, Nature 394: 539–544, 1998. (1998).

[124] E. Winfree, X. Yang, N. C. Seeman, *Universal Computation via Self-assembly of DNA: Some Theory and Experiments*, Proceedings of the Second Annual Meeting on DNA

Based Computers, Princeton University, June 10-12, 1996, pp. 191–213. Published in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Volume 44, Edited by Laura Faye Landweber and Eric B. Baum, Published by Am. Math. Soc., Providence, RI, (1999). ISBN-10: 0-8218-0756-0

[125] H. Yan, T. H. LaBean, L. Feng, and J. H. Reif, *Directed Nucleation Assembly of Barcode Patterned DNA Lattices*, Proceedings of the National Academy of Science(PNAS), Volume 100, No. 14, pp. 8103–8108, July 8, (2003). PubMed PMID: 12821776.

[126] Y. Zhang and N. C. Seeman, *A Solid-Support Methodology for the Construction of Geometrical Objects from DNA*, J. Am. Chem. Soc., **114**, 2656–2663, (1992).

[127] Y. Zhang and N.C. Seeman, *The Construction of a DNA Truncated Octahedron*, J. Am. Chem. Soc., **116**, 1661–1669, (1994).