# Local Parallel Biomolecular Computation

John H. Reif
Department of Computer Science
Duke University [‡]

**Abstract**

*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 [W96, WYS96]) 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 [B62], Berger [B66], Robinson [R71], and Lewis and Papadimitriou [LP81]) in combination with the DNA self-assembly techniques of Seeman et al [SZC94]. The likelihood for successful unmediated self-assembly of computations has not been determined (we discuss a simple model of assembly where there may be blockages in self-assembly, but more sophisticated models may have a higher likelihood of success).

We develop improved techniques to more fully exploit the potential power of LP-BMC. To increase the likelihood of success of assembly, we propose instead a refined *step-wise assembly* method, which provides control of the assembly in distinct steps. It does not require large volumes or convergence time, even in the worst case assembly model that we assume for our own assembly algorithms. 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 number of time steps. We give LP-BMC algorithms with linear assembly size and logarithmic time, 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 time. This provides logarithmic time LP-BMC algorithms for the large class of normal parallel algorithms [S71, U84, L92] 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 require much smaller volumes than previous DP-BMC algorithms [R95,R98b, OR97] 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.

[‡]Surface address: Department of Computer Science, Duke University, Box 90129, Durham, NC 27708-0129. E-mail: reif@cs.duke.edu. This work was first supported by Grants NSF/DARPA CCR-9725021, CCR-96-33567, NSF IRI- 9619647 and EIA-0086015, ARO contract DAAH-04-96-1-0448, and ONR contract N00014-99-1-0406; it was subsequently supported by National Science Foundation Grants CCF–0432038, CCF–0432047, ITR 0326157, EIA–0218376, EIA–0218359, and EIA–0086015. This paper is at http://www.cs.duke.edu/∼reif/paper/DNAassembly/Assembly.pdf . 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.

# 1 Introduction

**Biomolecular Computation.** We will use the term *nanocomputation* to denote computation executed at the molecular scale. See Reif [R98a] 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 [WHR87, S94, SM97] 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 [A94] 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 [SM97] have utilized certain *base methods* which include:

- generation of a large number of sequences by random assembly; see Adleman [A95] and Lipton [L95] (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 [Be94, A95, L95, BDL95, SM97, BCGT96, ARRW96] 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 [ARRW96] (see also an earlier proposal of [BDL95]) 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. [ARRW96] 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* [R95,R98b, OR97] 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 [B62], Berger [B66], Robinson [R71], and Lewis and Papadimitriou [LP81] 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. Winfree [W96] (also Winfree, et al [WYS96]), proposed to execute these domino tiling techniques in LP-BMC by assembling $2D$ arrays of DNA molecules, using the DNA self-assembly techniques of Seeman et al [SZC94]. 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, it is not clear what the likelihood is for a successful assembly. This is a critical issue, for if the likelihood of successful assembly is very low, possible remedies by use of multiple parallel assembly attempts or repetitions would require either unfeasible large volumes or take a correspondingly very large time. We consider a simple model for random assembly, which assumes assembly growth by tile placement at random untiled locations, using only tiles which have maximal matching with previously placed tiles. We show that for certain settings of the parameters in our random assembly models, computations via unmediated self-assembly can fail with extremely high likelihood $1 - 2^{-cA} \geq 1 - 2^{-cn^2/2}$. Since that random assembly model does not include the possibility of disassembly, our analysis is not conclusive, but does illustrate the possible pitfalls of unmediated self-assembly. We also consider a more sophisticated model that includes disassembly as well as assembly. Although recent computer simulations [W97] indicate that careful choice of settings of the parameters may allow unmediated self-assembly to succeed in such a

*Reif [R95,R98b] 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 Reif's 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 [OR97] 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 $2^{2^s}$.

model, unmediated self-assembly remains unproved and experimentally untested.

**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 [R95,R98b]). 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 [SZC94, WYS96], 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. It does not require large volumes or convergence time, even in the worst case assembly model that we assume for our own assembly algorithms. 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 Depth.** Using these assembly techniques, we propose LP-BMC methods that decrease the size of the assembly to linear in the input size and have small assembly depth (time steps for assembly). These 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 time *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 time *perfect shuffle and pair-wise exchange permutations*, which allows constant time emulation of the shuffle-exchange network,
- logarithmic time execution of a large class of *normal parallel algorithms* [S71, U84, L92] on shuffle-exchange networks (e.g., DFT and bitonic merge of $n$-vectors),
- logarithmic time *general permutations*, by use of known Beneš network techniques [B65],
- *evaluation of a bounded degree Boolean circuit* of size $n$ in time 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 [R95,R98b, OR97]).

**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. (In the Appendix (Section 10) we consider some random assembly models and give bounds on the success likelihood of unmediated 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 [F61] proposed nanofabrication of structures of molecular size. Nanotechnology, without use of DNA, is discussed in [CL92, M93].

Fabrication of nanostructures in DNA using self-assembly was pioneered by Seeman (e.g., see [SZC94]) in the 1990s (also see [CRFCC96, MLMS96] for other 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 [SCK89],

- *DNA knots:* e.g., ssDNA knots [MDS91, DS92] and Borromean rings[MS97],
- *DNA crossover molecules:* e.g., DX molecules of Fu and Seeman[FS93].

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 [ZS92, ZS94, SWLQ96, SQLYL96, SZDC95, SZC94, SC91, SZDWM94] and [SQLYL96, SWLQ96])

- $2D$ *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 [S88]), to avoid interaction between constructed molecules [ZS92].

Seeman also characterized the structural and topological properties of unusual DNA motifs in solution [SZDWM94, SQLYL96, SWLQ96].

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 [W61] 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 [W61] defined the tiling problem. Buchi [B62] 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 [B66] 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 [R71], and the text book of Lewis and Papadimitriou [LP81], (pages 296–300) provide a direct simulation of a single tape deterministic Turing Machine to prove this undecidablility result.

Garey, Johnson, and Papadimitriou [GJP77] (see citation of [GJ79], 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 [LP81]( 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 [GBS87] surveyed these and related results on the complexity of tiling, including these direct simulations of Turing Machines.

## 3 BMC using Unmediated Self-Assembly

Winfree [W96] 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 ([LP81]( pages 345–348), [GBS87]) 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.

Winfree [W96] then proposed a very intriguing idea: to do these tiling constructions by application of the DNA nanofabrication techniques of Seeman et al [SZC94], 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 [WYS96] 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 [GJP77] (see also [GJ79], page 257) NP-completeness proof for tiling of polynomial size extent. Winfree, et al [WYS96], 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 [WLW98] 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 [LYR98] 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 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 [W96] 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. Each tile $\pi$ 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 [W96] intended assembly, for each $t, 0 \leq t < T$

and $i, 1 \leq i \leq S$ where $(i = t) \bmod 2$, 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.

**Possible Blockages during Unmediated Self-Assembly.** A *blockage* is an untiled location where no tile can be placed there without a *pad mismatch* (a conflict between a pad of the tile and the pad of an abutting tile at that location). We assume there are no partial or total mismatches allowed between the pads of abutting placed tiles. In the assembly process, a blockage will eventually result in a situation where further tiling is not possible (without some disassembly) due to inconsistent pads on nearby tiles.

**Homogeneous Assemblies without Blockages During Unmediated Self-Assembly.** We will consider some cases where blockages may not occur. An assembly will be called a *homogeneous* if all the tiles of the assembly have identical pads, and will otherwise be called *heterogeneous*. Unmediated DNA self-assembly may often be used successfully for homogeneous assembly, since subassemblies can compose without blockages due to pad mismatches (however, they may not be able to compose due to geometry mismatches). Such a special case is similar to the homogeneous crystallization process found in physics. This is an important application in its own right from the perspective of biochemists and for the general goal of nanofabrication of large structures. In the case of a homogeneous crystal assembly with 1, or a small number of tiles, Seeman [S85] contended that there is an entropic driving force operative to solve small inconsistencies between the ideal contacts and ideal periodicity. An assembly frame, as proposed in this paper, may be used to seed an initial homogeneous crystal assembly, and to constrain subsequent unmediated homogeneous assembly.

**Assumptions that Rule out Assembly Blockages During Unmediated Self-Assembly.** The papers of Winfree [W96] and Winfree, et al [WYS96] do not consider the possibility of a blockage, as defined above. It is implicitly assumed that blockages do not occur in the assembly process. Instead, only assemblies without blockages are considered as end products of the assembly process. Let a tile be *a $k$-pad-matching tile* for a given location adjacent to the assembly if it may be placed into the assembly in this location with some $k$ pads which correctly match other pads in abutting previously placed tiles of the assembly. Winfree [W96] assumes the assembly process is restricted to add into the assembly only maximal matching tiles which are $k$-pad-matching, for $k > 1$, and never to add 1-pad-matching tiles. The unmediated DNA assembly of Winfree [W96] succeeds without blockage if the assembly process is so restricted.

**Assembly Blockages That Might Occur During Unmediated Heterogeneous Self-Assembly.** Let a tile $\tau$ be *a $m$-BP-matching tile* for a given location adjacent to the assembly if $\tau$ may be placed into the assembly in this location with a total of $m$ base pairs in pads which correctly match, via Watson-Crick complementation, abutting previously placed tiles of the assembly. Let a tile be a *maximal matching tile* in that given location, if it is a $m$-BP-matching tile in that given location, and there is no other tile which is a $m'$-BP-matching tile in that given location, for any $m' > m$. Note that the definition of a maximal matching tile in that given location depends on the current state of the assembly, and a maximal matching tile which is a $m$-BP-matching tile in the current assembly may be a $m'$-BP-matching tile in a later state of the assembly, for some $m' \geq m$. When we use unmediated DNA assembly techniques for general computation simulations, the resulting tiles will

not generally have identical pads, so the assemblies are heterogeneous. Thus a maximal matching tile $\tau$ may not be unique for a given location $L$; for example if an adjoining location $L'$ (where the tile has a pad which is expected to be matched) is currently untiled, there may be another tile $\tau'$ that is also currently a maximal matching tile in that given location $L$ (but placing $\tau$ or $\tau'$ at $L$ may be incorrect, since they might later have a pad mismatch in any tiling of that adjoining location $L'$). We shall show that such situations may lead to blockages, which may potentially be endemic to unmediated local assemblies doing computation simulations.

The above example indicates [†] that at a given state of the assembly, there can be many locations where a tile may be placed so that all maximal matching tiles in that location are only 1-pad-matching tiles, and these can quickly become 2-pad-matching tiles at a subsequent stage in the assembly.

**Blockage Probabilities in Random Assembly Models.** In Appendix Section 10 we consider various random assembly models for the kinetics of assembly via hybridization at a fixed temperature. For a simple random assembly model, we shall demonstrate in Appendix Section 10 for certain setting of assembly parameters, 1-pad-matching tiles might be initially placed at such locations, and thus blockages in assembly can occur. For a more sophisticated random assembly-disassembly model also considered in Section 10, the issue is unresolved, though there are some recent computer simulations of Winfree [W97] which seem to avoid blockages in a assembly-disassembly model for careful choice of settings of the parameters.

# 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 [W96, WYS96] to increase the likelihood that the assembly always succeeds (and does not require such large volumes or convergence time per step). However, it has the drawback of not being fully autonomous. For example, we modify the assembly of Winfree, et al [W96], 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. Let the *depth* of a tile in the intended tiling be the number of matches between tiles in paths of this breadth first assembly tree. Let the *assembly depth* $d$ be the maximum depth of the intended assembly. 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

[†]Note that only 1-pad-matching tiles compete to be placed at such locations. Also note that the placement of such a 1-pad-matching tile $\tau$ (see Figure 2) in the assembly may be shortly later reinforced by subsequent placement of tiles at adjoining locations that increase the number of pad matches of $\tau$ above 1 in the resulting assembly (see Figure 3). Thus, a 1-pad-matching tile can quickly become 2-pad-matching tile at a subsequent stage in the assembly. This can easily happen in unmediated DNA assembly even if the temperature has been set to favor 2-pad-matching over 1-pad-matching. Thus, it does not seem reasonable to us to arbitrarily restrict the assembly process so as to adds tiles into the assembly only at locations where there are only $k$-pad-matching tiles, for $k > 1$.

solid-supported using standard surface biotechnology (e.g., immobilized via surface attachments [S88]).

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 [W96] 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 $(i = t) \bmod 2\}$, thus allowing the assembly to advance from time $t-1$ to time $t$ on all the tape positions $1 \leq i \leq S$ where $(i = t) \bmod 2\}$. 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 4). The time-stamped paddings of each of these tiles do not interfere with any other tiles at this time, so there can be no blockages: 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 [ZS92] for the 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 [SZC94, SWLQ96, SQLYL96] (see also Subsection 2.2). Interaction between tiles during their construction can be avoided using solid-support methodology described in [ZS92]. Winfree, et al [W96, WYS96] made use of DX molecules [FS93] 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 [SQLYL96]), 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 [A94, L94, BL95, B96, GFBCL96] and [M96, KCL96, GFBCL96, DMGFS96, JK97a, DMRGF97].

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 $\bar{E}(-)$ 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:* $\{\pi(i,T)|, 1 \leq i \leq S$ where $(i = T) \bmod 2\}$ 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 key 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 8 and 25, giving a ssDNA

encoding the input n-vector. See Figures 9 and 26, 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 [SZC94], 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 [SCK89, DZS92] and the DX molecules of Fu and Seeman[FS93] 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 [DD92].
- Another option is the use of a motif known as DX+J [LYQS96], which is almost as stiff as DX, and provides the option of having DNA come out the side of a tile.
- Seeman [S97] 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 [W96] and also [WYS96]. In the intended assembly, for each $t, 1 \leq t \leq T$ and $i, 1 \leq i, j \leq S'$ where $(i = t) \bmod 2$ and $(j = t) \bmod 2$ 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 [SQLYL96]. 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 [W96], 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 $\lambda$ 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 $D = \{$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.

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 5) 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 6). Each tile will have pads, as defined below[‡]. 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 $t$th input),
- the *output* pad $\bar{E}''(a', t)$ on the top side (with the intent to match with the $t$th 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 $t$th 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 $t$th step, and that must be placed so its input pad matches the $t$th 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 7) 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. [BDLS95] 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 [GFB96] 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 $E(a_1, 1) \ldots E(a_n, n) E(0, n+1)\beta$, $\bar{E}(0, 2n+1)^R \bar{E}(a'_n, 2n)^R, \ldots \bar{E}(a'_1, n+1)^R$. Also, a unit

[‡]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.

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 8) of length $n+1$ and unit height by the use of rigid framing techniques described in subsection 4.2 (see Figure 9).

Each tile in this case is a unit square with a unit length internal middle segment as given in Figure 10.

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 11) 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 12, with 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, 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 [LF80].

Fortune and Wyllie [FW78] 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 13) 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 14. 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 $\bar{E}(a, i+1, t-1)$ (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 $a_i \cdot a_{i+1} \cdot \cdots \cdot a_j$. The resulting assembly (see Figures 15, 16, and 17) 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 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 $\bmod 2$, 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 18, consisting of two horizontal rectangles (each with the same shape of the 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 $\bar{E}(-), \widehat{E}(-)$ 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 $\bar{E}'(b,(i+1)/2,t)$ 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 $a_1\cdot\cdots\cdot a_n$. The resulting $3D$ assembly (see Figures 19, 20, and 21) 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.

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 $\bar{E}'(b_{(i-1)/2}^{(t)},(i+1)/2,t)$ 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 5), by the use of rigid framing techniques described in subsection 4.2.

Each tile is a $2 \times 2$ square as given in Figure 22. 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),
- *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, so there can be no blockages, and the tiling is unique.

The resulting $2D$ assembly is unique and is in the shape of a rectangular tiling (see Figure 23) 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

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

By known recombinant DNA 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 24) of width $n/2$ and height 2 by the use of rigid framing techniques described in subsection 4.2 (see Figure 25).

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 26.

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

- a $1 \times 2$ 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
  - 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 $\bar{E}'(a, i)\bar{E}'(a, n/2+i)$ (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 27) is unique and is a rectangle of length $n$ and height 2, containing tiles $\tau''(a_1, a_{n/2+1}, 1), \tau''(a_2, a_{n/2+2}, 2), \ldots, \tau''(a_{n/2}, a_n, n/2)$ 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;

[§] We can do the reversal (with refinements suggested by [W97]) 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.

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 $\bmod 2$, 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 28, resembling pup tent, with the same rectangular base as the $1 \times 2$ 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 [S71]) 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 [U84] and Leighton [L92] 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 [S90] 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 time 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 time. 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 [U84] and Leighton [L92] (note they are also sometimes called *ascend-descend* algorithms). A normal parallel algorithms can be executed (see details in [S71, U84, L92]) 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 time, as follows:

- The techniques of Subsection 8 executes the shuffle-exchange permutation by LP-BMC in constant time.

- 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 time 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 [B68], which merges two length $n$ vectors using a normal parallel algorithm and
- the $O(\log^2 n)$ time, $n$ processor [B68] 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 time 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 [B65], allows the execution of an (arbitrary) fixed permutation of $n$ data elements by use of a normal parallel algorithm (Waksman [W68]). 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 time, 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 time 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 [W97] 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 $O(\log n)$ 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 [WLW98, WYS96, LYK00, LWR99, MMLR00] and surveyed in [RLS00].

**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

[A94] L. Adleman, *Molecular Computation of Solution to Combinatorial Problems*, Science, **266**, 1021–1024, (1994).

[A95] L. Adleman, *On Constructing a Molecular Computer*, U.S.C. Dept of CS Draft (1995). Available via anonymous ftp from $ftp.usc.edu/pub/csinfo/papers/adleman/molecular\_computer.ps$

[ARRW96] L. M. Adleman, P. W.K. Rothemund, S. Roweis, E. Winfree, *On Applying Molecular Computation To The Data Encryption Standard*, 2nd Annual Meeting on DNA Based Computers, Princeton University, June 1996.

[AJPWL96] 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.

[A96] 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, June 1996.

[AGH96] M. Amos, A. Gibbons, and D. Hodgson, *Error-resistant Implementation of DNA Computations*, 2nd Annual Meeting on DNA Based Computers, Princeton University, June 1996.

[AH97] M. Arita and M. Hagiya, *Joining and Rotating Data with Molecules*, ICEC'97 Special Session on DNA Based Computation, Indiana, April 1997.

[BCGT96] 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, May 1996.

[B68] K. Batcher *Sorting Networks and their applications*, Spring Joint Computer Conference, **32**, 307–314, AFIPS Press, Montvale, N. J. (1968).

[B95] E. B. Baum, *How to build an associative memory vastly larger than the brain*, Science, April 28, 1995.

[B96] E. B. Baum, *DNA Sequences Useful for Computation*, 2nd Annual Meeting on DNA Based Computers, Princeton University, June 1996.

[Be94] D. Beaver, *Factoring: The DNA Solution*, Advances in Cryptology, Asia Crypt94 Proceedings, Lecure Notes in Computer Science, (1994), Springer Verlag,

[BeA95] D. Beaver, *A Universal Molecular Computer*, revised as *Molecular Computing*, Penn State University Technical Memo CSE-95-001, Pond Lab, (1995).

[BeB95] D. Beaver, *Computing with DNA*, J. Comp. Biol., **2**, 1–7, (1995).

[B65] V. Beneš, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, New York, NY (1965).

[B66] R. Berger *The Undecidability of the Domino Problem*, Memoirs of the American Mathematical Society, **66**, (1966).

[BDL95] D. Boneh, C. Dunworth, R. Lipton, *Breaking DES Using a Molecular Computer*, Princeton CS Tech-Report number CS-TR-489-95, (1995).

[BDLS95] 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 Math, (Dec 96)

[BL95] D. Boneh, R. Lipton, *Making DNA Computers Error Resistant*, Princeton CS Tech-Report CS-TR-491-95, Also in 2nd Annual Meeting on DNA Based Computers, Princeton University, June 1996.

[BKP90] C. Brooks, M. Karplus, M. Pettitt, *Proteins, A Theoretical Perspective of Dynamics, Structure & Thermodynamics*, John Wiley & Sons,

[B62] J.R Buchi, *Turing Machines and the Entscheidungsproblem*, Mathematiche Annalen, **148**, 201–213, (1962).

[CCCFF97] 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.

[CRFCC96] 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).

[CCTKS88] 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).

[CRWEC95] 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.

[CL92] B. Crandall and J. Lewis (eds.), *Nanotechnology*, MIT Press, (1992).

[DMGFS96] 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, June 1996.

[DMRGF97] 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.

[DHK96] A. L. Delcher, L. Hood, R. M. Karp, *Report on the DNA/Biomolecular Computing Workshop*, June 1996.

[DD92] M. D. Distefano, P. B. Dervan *Ligand promoted dimerization of oligonucleotides binding cooperatively to DNA*, J.Am.Chem.Soc., **114**, 11006-11007, (1992).

[DDSPL93] , Drmanac, R, Drmanac, S., Strezoska, Z., Paunesku, T., Labat, I., Zeremski, M., Snoddy, J., Funkhouser, W. K., Koop, B., Hood, L., Crkenjakov, R., *DNA Sequence Determination by Hybridization: A Strategy for Efficient Large-Scale Sequencing*, Science, **260**, 1649–1652, (1993).

[DS92] 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).

[DZS92] S.M. Du, S. Zhang and N.C. Seeman, *DNA Junctions, Antijunctions and Mesojunctions*, Biochem., **31**, 10955–10963, (1992).

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

[ER82] M. Engler and C. Richardson, *The Enzyme*, (P. Boyer, ed.) Academic Press, 3–29, (1982).

[F61] R. Feynman, Minaturization (D. Gilbert, ed.), Reinhold, 282–296, (1961).

[FW78] S. Fortune and J. Wyllie, *Parallelism in random access machines*, Proc. 10th Annual ACM S.T.O.C., San Diego, CA, 114–118, (1978).

[FS93] T.-J. Fu and N.C. Seeman, *DNA Double Crossover Structures*, Biochemistry, **32**, 3211–3220, (1993).

[GJ79] 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).

[GJP77] M. R. Garey, D. S. Johnson, and C. H. Papadimitriou, unpublished manuscript, (1977).

[GFBCL96] 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.

[GBS87] S. Grunbaum, Branko, and G.C. Shepard *Tilings and Patterns*, H Freeman and Company, **Chapter 11**, (1987).

[GB96] 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., June 10-12, 1996, American Mathematical Society, Providence, RI (in press), (1996).

[GFB96] Guarnieri, F., Fliss, M., and C. Bancroft, *Making DNA add*, Add. Science, **273**, 220–223, (1996).

[J92] J. JáJá, *An Introduction to Parallel Algorithms*, Addison Wesley, (1992).

[LF80] R.E. Ladner, and M.J. Fischer, *Parallel Prefix Computation*, JACM, **27(4)**:831–838, (1980).

[LYR98] Thomas H. LaBean, Hao Yan, John H. Reif and Nadrian Seeman, *Construction and Analysis of a DNA Triple Crossover Molecule*, (November. 1998).

[L92] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufmann Press, San Mateo, CA, Chapter 3, (1992).

[LP81] H. R. Lewis and C.H. Papadimitriou *Elements of the Theory of Computation*, Prentice-Hall, pages 296–300 and 345–348 (1981).

[JK96] N. Jonoska, S. A. Karl, *A Molecular Computation of the Road Coloring Problem*, 2nd Annual Meeting on DNA Based Computers, Princeton University, June 1996.

[JK97a] N. Jonoska, S. A. Karl, *Ligation Experiments in Computing with DNA*, ICEC'97 Special Session on DNA Based Computation, Indiana, April 1997.

[JK97b] 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).

[LL97] 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).

[KCL96] P. Kaplan, G. Cecchi, and A. Libchaber, *DNA based molecular computation: Template-template interactions in PCR*, The Second Annual Workshop on DNA Based Computers, American Mathematical Society, To appear, (1996).

[KMRS96a] S. A. Kurtz, S. R. Mahaney, J. S. Royer, J. Simon, *Active Transport in Biological Computing*, 2nd Annual Meeting on DNA Based Computers, Princeton University, June 1996.

[KMRS96b] S. A. Kurtz, S. R. Mahaney, J. S. Royer, J. Simon, *Biological Computing*, to appear, (1996).

[LYK00] LaBean, T. H., Yan, H., Kopatsch, J., Liu, F., Winfree, E., Reif, J.H. and Seeman, N.C., 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

[LWR99] LaBean, T. H., E. Winfree, J. H. Reif, Experimental Progress in Computation by Self-Assembly of DNA Tilings, 5th International Meeting on DNA Based Computers(DNA5), MIT, Cambridge, MA, (June, 1999). To appear in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, ed. E. Winfree, to appear American Mathematical Society, 2000. http://www.cs.duke.edu/∼-thl/tilings/labean.ps

[LL99] Lagoudakis, M. G., 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).

[LYQS96] 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).

[L94] R. Lipton, *Speeding Up Computations via Molecular Biology*, Princeton University Draft, (1994).

[L95] R. J. Lipton, *DNA Solution of Hard Computational Problems*, Science, **268**, 542–545, (1995).

[L96] , R. J. Lipton. *DNA Computations Can Have Global Memory*, unpublished manuscript, April 1996.

[LBL 96] R. J. Lipton, D. Boneh, and L. Landweber, *Analog DNA Based Computation*, in preparation. (1996).

[LGCCL96] Q. Liu, Z. Guo, A.E. Condon, R.M. Corn, M.G. Lagally,, and L.M. Smith, *A Surface-Based Approach to DNA Computation*,, Proc. Second Annual Princeton Meeting on DNA-Based Computing, June 1996.

[MS97] C. Mao and N.C. Seeman, *Construction of Borromean Rings from DNA*, Nature, **386**(6621), 137–138, (March,1997).

[MMLR00] Mao, C., T.H. LaBean, J. H. Reif, and N.C. Seeman, An Algorithmic Self-Assembly, Nature, Sept 28, (2000). www.cs.duke.edu/∼reif/paper/SELFASSEMBLE/AlgorithmicAssembly.pdf

[MH87] J. McCammon, S. Harvey, *Dynamics of Proteins and Nucleic Acids*, Cambridge University Press, (1987).

[M93] R. Merkle, Nanotechnology **4** 21, (1993).

[MR85] 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).

[M96] K. U. Mir, *A Restricted Genetic Alphabet for DNA Computing*, 2nd Annual Meeting on DNA Based Computers, Princeton University, June 1996.

[MLMS96] 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.

[MDS91] 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).

[MDFS97] 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.

[OR97] 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.

[OP94] R. Old and S. Primrose, *Principles of Gene Manipulation, An Introduction to Genetic Engineering*, Blackwell Scientific Publications, Fifth Edition, (1994).

[PP97] G. E. Plum and D. S. Pilch, *Nucleic Acid Hybridization: Triplex Stability and Energetics.*, Annu. Rev. Biophys. Biomol. Struct. , **24:**, 319–350, (February 1997).

[Po97] R. Pool, *Dr. Tinkertoy*, Discover, **18:2**, 50–57, (February 1997).

[QYS96] 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).

[R93] J. Reif (ed.), *Synthesis of Parallel Algorithms*, Morgan Kaufmann, (1993).

[R95,R98b] J. 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.

[R98a] J. 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 $http://www.cs.duke.edu/\sim reif/paper/paradigm.ps$.

[RLS00] J.H. Reif, T. H. LaBean, and Seeman, N.C.,Challenges and Applications for Self-Assembled DNA Nanostructures, Invited paper, Sixth International Meeting on DNA Based Computers (DNA6), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Leiden, The Netherlands, (June, 2000) ed. A. Condon. To be published by Springer-Verlag as a volume in Lecture Notes in Computer Science, (2000). http://www.cs.duke.edu/∼reif/paper/SELFASSEMBLE/selfassemble.ps

[RL00] 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), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Leiden, The Netherlands, (June, 2000) ed. A. Condon. To be published by Springer-Verlag as a volume in Lecture Notes in Computer Science, (2000). www.cs.duke.edu/∼reif/paper/Error-Restore/Error-Restore.ps

[R 78] R. J. Roberts, *Restriction and modification enzymes and their recognition sequences*, Gene, **4**, 183–194, (1978).

[R71] R. M. Robinson *Undecidablility and Nonperiodicity for Tilings of the Plane*, Inventiones Mathematicae, **12**, 177–209, (1971).

[Ro95] P.W.K. Rothemund, *A DNA and restriction enzyme implementation of Turing Machines*, manuscript available at http://www.ugcs.caltech.edu/ pwkr/oett.html, (1995).

[RW95] D. Roo$\beta$ and K.W. Wagner, *On the power of Bio-Computers*, unpublished manuscript.

[RWBCG96] 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*, 2nd Annual Meeting on DNA Based Computers, Princeton University, June1996, Also as Laboratory for Molecular Science, Univ. of So. California technical report *A Sticker Based Model for DNA Computation*, May 1996.

[RS97] G. Rozenberg, A. Salomaa, ICEC'97 Special Session on DNA Based Computation, Indiana, April 1997.

[R96] H. Rubin, *Looking for the DNA killer app.*, Nature, **3**, 656–658, (1996).

[SFM89] J. Sambrook, E. Fritsch, T. Maniatis, *Molecular Cloning*, Cold Spring Harbor Lab, NY, (1989).

[S90] E. Schawbe *On the computational equivalence of hypercube-drived networks*, Proceeding of the 2nd Annual Symposium on Parallel Algorithms and Architectures, 388–397 (1990).

[S82] N.C. Seeman, *Nucleic Acid Junctions and Lattices*, J. Theor. Biol., **99**, 237-247, (1982).

[S97] N.C. Seeman, *personal communication to J.H. Reif*, (1997).

[S85] N.C. Seeman, *Macromolecular Design, Nucleic Acid Junctions and Crystal Formation*, Journal of Biomolecular Structure and Dynamics, **3**, 1-34, (1985).

[SC91] N. C. Seeman and J. Chen, *Synthesis from DNA of a molecule with the connectivity of a cube*, Nature, **350**, 631–633, (1991).

[SCDMZ93] 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).

[SCK89] N. C. Seeman, J.-H. Chen, N. R. Kallenbach, *Gel electrophoretic analysis of DNA branched junctions*, Electrophoresis, **10**, 345–354, (1989).

[SQLYL96] 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).

[SWLQ96] 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*, The Second Annual Workshop on DNA Based Computers, American Mathematical Society, June 1996.

[SZC94] N. C. Seeman, Y. Zhang, and J. Chen, *DNA nanoconstructions*, J. Vac. Sci. Technol., **12:4**, 1895–1905, (1994).

[SZDC95] 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).

[SZDWM94] 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).

[SM97] J. Setubal and J. Meidanis *Introduction to Computational Molecular Biology*, PWS Pub. Co., Chapt 9, (1997).

[S94] R. Sinden, *DNA Structure and Function*, Academic Press, (1994).

[S88] L. M. Smith, *Automated Synthesis and Sequence Analysis of Biological Macromolecules*, Anal. Chem., **60**, 381A–390A, (1988).

[SS95] W. Smith, A. Schweitzer, *DNA Computers in Vitro and Vivo*, NEC Research Inst. Tech Report 95-057-3-0058-3. (1995).

[S71] H.S. Stone *Parallel Processing with the perfect shuffle*, IEEE Trans. on Computers, **C-20**:2, 153-161 (1971).

[U84] J. Ullman, *Computational Aspects of VLSI*, Computer Science Press, (1984), Chapter 6.

[W68] A. Waksman *A Permutation Network*, JACM, **15(1)**,159-163 (1968).

[W61] H. Wang, *Proving Theorems by Pattern Recognition*, Bell System Technical Journal, **40**, 1–141, (1961).

[WDS96] H. Wang, R.J. Di Gate, and N.C. Seeman, *An RNA Topoisomerase*, Proc. Nat. Acad. Sci. (USA), **93**, 9477–9482, (1996).

[WGWZ92] J. Watson, M. Gilman, J. Witkowski, M. Zoller, *Recombinant DNA (2nd ed.)*, Scientific American Books, W.H. Freeman and Co., (1992).

[WHR87] J. Watson, N. Hoplins, J. Roberts, et. al., *Molecular Biology of the Gene*, Benjamin/Cummings, Menlo Park, CA, (1987).

[W95] E. Winfree, *Complexity of Restricted and Unrestricted Models of Molecular Computation*, California Institute of Technology technical report May, 1995, Also Princeton DIMACS Technical Report workshop on DNA-based computers, To appear, April 4, 1995.

[W96] E. Winfree, *On the computational power of DNA annealing and ligation*, DNA based computers, Lipton, R.J. and Baum, E.B. eds., Am. Math. Soc., Providence, RI, (1996).

[W97] E. Winfree, communication to J. H. Reif (June, 1997).

[WLW98] Erik Winfree, Furong Liu, Lisa A. Wenzler, Nadrian C. Seeman, *Design and Self-Assembly of Two Dimensional DNA Crystals*, Nature 394: 539–544, 1998. (1998).

[WYS96] E. Winfree, X. Yang, N. C. Seeman, *Universal Computation via Self-assembly of DNA: Some Theory and Experiments*, 2nd Annual Meeting on DNA Based Computers, Princeton University, June 1996.

[ZS92] 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).

[ZS94] Y. Zhang and N.C. Seeman, *The Construction of a DNA Truncated Octahedron*, J. Am. Chem. Soc., **116**, 1661–1669, (1994).

# 10 Appendix: Models of Unmediated Local Assembly and its Success Probability

We consider here random assembly models for the kinetics of assembly via hybridization at a fixed temperature. For certain setting of assembly parameters, we shall demonstrate that blockages in assembly can occur in these random assembly models, for certain setting of parameters.

**A Random Assembly Model.** We now consider a model for random assembly, where the assembly grows by tile placement at random untiled locations adjacent to the assembly, and the tiles are choosen by a random process depending on the number of matches with previously placed tiles. We presume that the initial tiles are initially placed as specified, without any initial blockage. As a *random assembly* model, we assume that the further assembly is done by choosing a random untiled location $L$ at a unit region which adjoins at least one location so far tiled. We then choose a tile $\tau$ to place in that chosen location $L$, where that $\tau$

is chosen randomly among all tiles with a relative probability $p_m$, which we call the *preferential matching* parameter. Here $m$ is the total number of base pairs in pads of $\tau$ that correctly match the pads of tiles abutting this location $L$. To approximately model the kinetics of assembly via hybridization, we define the preferential matching parameter as $p_m$ using the Boltzmann distribution from statistical mechanics [EC]; this is an increasing exponential function of $m$ of form $e^{c_0 m - c_{init}}$, where $c_0 > 0$ is the free energy of a single base-pair interaction and $c_{init} > 0$ is the initiation energy, which depend upon the concentration of units, temperature, and other physical parameters. Thus the preferential matching parameter for a tile $\tau$ is $p_m(\tau) = e^{c_0 m - c_{init}}$, for a constant $c_0 > 0$. To obtain the probability of choosing a tile $\tau$ from the relative probability $p_m(\tau)$, we normalize by dividing by the sum over the preferential matching parameters of all tiles that can be chosen (This random choice is done among all tiles, including an empty tile with $p_m = 1$.). The assembly progresses until the tiling is completed, or there is no position where further tiling is possible.

Let us assume for simplicity that the number of base pairs per pad of each tile is $\ell$. If a tile $\tau$ has $k$ matching pads at a given site, then $m = k\ell$, so the preferential matching parameter is $p_m(\tau) = e^{c_0 k\ell - c_{init}}$.

1. When $c_0 k\ell - c_{init}$ is very small, and negative, with high probability the empty tile is chosen; thus rather than placing an incorrect tile, no mistake is made and the same site might be chosen again later, when a correct $k$-pad-match might be placed (this is the case considered by Winfree [W97]).

2. When $c_0 k\ell - c_{init}$ is moderately small, but positive, (e.g., with constant tile pad lengths), we get a significant likelihood of placement of tiles with incorrect matches and placement of tiles which are not maximal matching.

3. However, for moderately large values of $c_0 k\ell - c_{init}$, which we would recommend, then the assembly is highly preferential to correct $k$ pad matches, and so pad mismatches among abutting tiles are very unlikely, and hence we are very likely to place a maximal matching tile in a given location. In the case of long pad matches and appropriate temperature, this leads to a model having correct maximal $k$ pad matches with high probability[¶]. For example, if say $c_0 k\ell - c_{init} = c' \log n$, for a sufficiently large constant $c' > 0$, then maximal matching tiles are placed with likelihood $\geq 1 - n^{-\Omega(1)}$.

**A Simplified Random Assembly Model.** We now consider a simplified model for random assembly. As in the previous described model for random assembly, the assembly grows by tile placement at random untiled locations adjacent to the assembly. However, we assume we place at those locations only tiles randomly chosen among all maximal matching tiles in that location. (Note that by the above discussion, for moderate values of $c_0 k\ell - c_{init} = c' \log n$, this simplified model which places maximal matching tiles, only differs with very low likelihood $n^{-\Omega(1)}$, from the previous described model for random assembly with a preferential matching parameter.) Unfortunately, even in this favorable case, where we place in randomly

[¶] Nevertheless, at constant temperature there will still be some finite likelihood of error due to mismatches. Winfree [W97] has suggested the following scenario where maximal matching might be a good model for step-wise assembly: At each step, add the tiles for that step and cool down slowly.

(a) At a temperature where $k$ pad matches is a favorable reaction but $k' < k$ pad matches are not favorable, sites where the maximal match has $k$ pads will be filled selectively with the correct tile.

(b) At a lower temperature, sites where the maximal match has $k-1$ pads can be filled selectively, etc. Note that at this lower temperature, an incorrect partial match of $k-1$ pads could be not be placed into a site with a match of $k$ pads, since that site is already filled.

(c) And so on, as the temperature is lowered, maximal matches with fewer and fewer pads may be placed.

However, it should be noted that in this scenario an incorrect tile with $k+1$ pad matches and 1 pad mismatch may be preferred to a correct tile with the intended $k$ pad match and 0 pad mismatches. Thus the assumption that partial mismatches are disallowed is confirmed by this scenario, only if the assembly has a geometry that rules out the possibility of incorrect tile assembly with $k+1$ pad matches and 1 pad mismatch.

chosen locations only maximal matching tiles, we will show the unmediated assembly of Winfree's [W96] can nevertheless fail with high likelihood.

**The Success Probability of Unmediated Local Assembly.** We now provide a probabilistic analysis of Winfree's [W96] direct simulation of a Universal TM (or cellular automata) running in time $T \geq n$ and space $S \geq n$ by assembly of an $S/2 \times T$ array described in detail at the end of Section 3 (also given in detail in [W96]). We assume the simple random assembly model described above, where we randomly choose an untiled location (abutting a previously tiled location) and choose a tile to be placed in that location, randomly among all maximal matching tiles in that location. We will prove that during an attempt to complete an unmediated DNA assembly of size $A = ST/2$, the probability $\beta$ of failure by blockage can be $\approx 1 - 2^{-cA^2}$, where $c = O(1/\sigma^2)$ and $\sigma$ is the maximum number of maximal matching tile types for any given location (in the construction of [W96], $\sigma$ is a constant depending on the number of tape symbols and states of the simulated TM). Hence, for $T \geq n$ and $S \geq n$, the probability of failure can be $\beta \geq 1 - 2^{-cn^2/2}$, which very quickly approaches 1 for even small $n$.

We first will (informally) describe what potentially may go wrong with this unmediated DNA assembly method. We assume the reader is familiar with this tiling construction. For general computation simulations such as these, the resulting tiles do not necessarily have identical pads. Recall a *blockage* is an untiled location where no tile can be placed there without a pad mismatch. Now since the assembly is unmediated, it can proceed both forward and backward in time. Then in general, the reverse time simulation can not be uniquely determined by the current state of the assembly. In particular, a partial assembly may choose a reverse simulation that is not consistent with the rest of the values of the tape cells. This can cause a blockage that halts the assembly locally. This blockage can occur even if the TM as a whole is reversible, since the entire tape contents are not determined by a partial assembly. In the simulation of a Universal TM, or any other universal machine, similar situations can be made to occur.

This argument can be formalized as follows. Let $\pi(i,t)$ denote a tile placed in location $(i,t)$. Ideally, we would like the assembly process to position tiles to simulate the machine's computation, as follows. For each $t, 1 \leq t \leq T$ and $i, 1 \leq i \leq S$ where $(i = t) \bmod 2$, there is a tile $\pi(i,t)$ placed in location $(i,t)$ representing the transition from time $t-1$ to time $t$ at the two consecutive tape cells at positions $i, i+1$. Note that Winfree's [W96] construction has tile $\pi(i,t)$ abut and match pads at neighboring tiles $\pi(i-1,t-1), \pi(i+1,t-1), \pi(i-1,t+1), \pi(i+1,t+1)$, if these neighboring tiles have been placed down. The tiles $\pi(1,0), \ldots, \pi(S,0)$ representing the initial configuration of the TM at time 0 can be assumed to be initially placed in the correct location. However, the assembly may not necessarily correctly position tiles associated with subsequent times $> 0$. Now suppose that the simulation assembly process has progressed with a partial subassembly at time step $t-1$, but where there remains at least a $3 \times 3$ area of the plane to tile associated with tape positions $i, i+1, i+2$ and time steps $t, t+1, t+2$. Further suppose that the simulation assembly process proceeds forward via placing a single tile $\pi(i,t)$ (bridging time $t-1$ to time $t$ at two consecutive tape cells $i, i+1$), and then the random assembly also proceeds via placing a tile $\pi'(i+1,t+1)$ (bridging time $t$ to time $t+1$ at the two consecutive tape cells $i+1, i+2$). Now further suppose the tile $\pi(i+2,t)$ has not yet been positioned by the random assembly, so the value of the tape cell $i+2$ at time $t$ is not determined by immediate context. Thus the tile $\pi'(i+1,t+1)$ abuts only one tile $\pi(i,t)$. Since the random assembly has not yet placed tile $\pi(i+2,t)$, $\pi'(i+1,t+1)$ does not abuts that tile. The key problem is that since the tile $\pi'(i+1,t+1)$ abuts only one tile $\pi(i,t)$ (rather than two tiles) of previous time step $t$, the tile $\pi'(i+1,t+1)$ may have been incorrectly placed in the assembly, so that if we later attempt (see Figure 2) to correctly place tile $\pi(i+2,t)$, its pad $in_2$ abutting (incorrectly placed) tile $\pi'(i+1,t+1)$ may not match, and the assembly is blocked (In contrast, a correctly placed tile $\pi(i+1,t+1)$ would have matched at its pads $in_1, in_2$ where it abuts both tile $\pi(i,t)$ and (a later placed) $\pi(i+2,t)$. However the random assembly instead incorrectly placed a tile $\pi'(i+1,t+1)$ which matches only at its pad $in_1$ abutting $\pi(i,t)$ but may not later match at its pad $in_2$ abutting where $\pi(i+2,t)$ would later be placed.)

The tile pads depend on the transition function of the simulated TM or cellular automaton, and this

transition function can be appropriately chosen to insure a possible mismatch. Thus, given any partial assembly where there remains a $3 \times 3$ area of the plane to tile, the tiling can in the worst case be so blocked by the further assembly of just 2 additional tiles into this area. The incorrectly placed tile $\pi'(i+1, t+1)$ matches at only one pad, but is a maximal matching tile in that location, since no other tile matches at that location within the current assembly with more than one pad. The other tile $\pi(i, t)$ also is a maximal matching tile in its location. Thus both these tiles are chosen with probability $\geq 1/\sigma$, where $\sigma$ is the maximum number of distinct tile types that can can be maximal matching tiles in any given location. Hence the probability both of these tiles are chosen, causing a blockage at any point in the assembly process, is at least $\beta_0$, where $\beta_0 = \Omega(1/\sigma^2)$. Thus, the probability of no failure by blockage during the $(S-3)(T-3)/2$ assembly steps, where there remains a $3 \times 3$ area of the plane to tile, can be $\leq (1-\beta_0)^{(S-3)(T-3)/2} \approx e^{-\beta_0(A-3(S+T)/2)} \leq e^{-cA}$, where the size of the assembly is $A = ST/2$ and $c = \beta_0 + 3$. Hence, since $T \geq n$ and $S \geq n$, the total probability of failure by blockage during an attempt to complete an assembly of size $A$ is $\beta \geq 1 - e^{-cA} \geq 1 - e^{-cn^2/2}$, which very quickly approaches 1 for even small $cn^2$.

**Further Discussion of Feasibility of Unmediated Local Assembly for Computation.**

A possible objection to our analysis of unmediated local assembly is that not every Turing Machine would result in such blockage problems; and indeed we have chosen an especially bad one. But we know of no non-trivial computation (other than permutations of the input) via unmediated DNA assembly that does not exhibit this blockage problem, for worst case inputs. Moreover, we found no effective way to insure against these blockage problems a priori, and it may be (we conjecture this) impossible to do so in the case of a Universal Turing Machine, since it simulates all others.

This blockage problem might be remedied by the use of DP-BMC to allow for many parallel assemblies. In particular, if we use $1/(1-\beta) \approx 2^{cA}$ parallel assemblies, then the probability of success increases to $1 - \beta^{1/(1-\beta)} \approx 1 - 1/e$, which is constant, but this requires extremely large volumes growing as $1/(1-\beta) \geq 2^{cn^2/2}$, which would overwhelm even DP-BMC for very small $n$. Alternatively, we might repeat the assembly process until success, but this requires a very large expected number $1/(1-\beta) \geq 2^{cn^2/2}$ of repeats.

**A Random Assembly-Disassembly Model.** A quite valid objection to our analysis is that our random assembly model is too simple to fully model the kinetics of assembly via hybridization at a fixed temperature. Indeed our random assembly model does not allow for later preferential disassembly of partially matching tiles (that may have been incorrectly placed) in favor of better matching tiles (that may now be correctly placed). We now consider a more sophisticated *random assembly-disassembly model* that allows both random preferential pairing of matched tiles and also a probability of preferential disassembly of a partially matched tiles disassociating, where for a randomly chosen region adjacent to the assembly, (i) if the region is untiled, then we choose a tile $\tau$ to place in that chosen location, where that tile is chosen randomly among all tiles with relative probability again given by the preferential matching parameter $p_m(\tau)$, and (ii) if the region is tiled, then the probability of disassembly of the tile $\tau$ is a function $q_m(\tau)$, where in both cases $m$ is the total number of base pairs in its matching pads (correctly matching pads of abutting tiles), that the tile has in this placement. To approximately model the kinetics of assembly via hybridization, we can let $p_m$ be defined by the Boltzmann distribution as an increasing exponential function $e^{c_0 m - c'_{init}}$ of $m$ for constants $c_0, c_{init} > 0$ and we let $q_m$ be a decreasing exponential function $e^{-c_0 m + c'_{init}}$ of $m$ for constants $c'_0, c'_{init} > 0$, where the constants depend upon the concentration of units and temperature. In practice, the on-rate for matches and mismatches is expected to be (roughly) identical if the DNA are present in equal concentrations, so there is little or no preference for association, and only preference for dissociation. This can be modeled by setting the constant $c_0$ very small; and might be considered 0 in practice. Thus $p_m$ is very nearly constant and nearly independent of $m$.

Such a random assembly-disassembly model is much harder to analyze rigorously. It remains an open problem to provide rigorous upper and lower bounds for such a random assembly-disassembly model. We know of no non-trivial examples of computations (other than the data permutation operations proposed

in this paper) via unmediated DNA assembly where there are provable success bounds in any reasonable probabilistic assembly-disassembly model.

Potentially the situation may be worse, rather than improved. Consider the situation of a large assembly composed of (i) a correct initial assembly, composed at a number of locations with (ii) a partial assembly that is incorrect; in that case the random disassembly process might occur preferentially at the mismatched boundaries of these two subassemblies, but would not be able to preferentially distinguish locally which is the correct subassembly and would appear to randomly disassemble both subassemblies. Thus the probability of a blockage at any step of the assembly may remain constant, and rather than halt the assembly, the number of blockages may increase as the assembly proceeds, with high likelihood. Since the probability of any pair of matched tiles disassociating is not dependent on the existence of blockages, we cannot derive a better upper bound than $2^{c'A} \geq 2^{c'ST}$ time for convergence to the required complete $S/2 \times T$ tiling, for a constant $c' > 0$.

In a random assembly-disassembly model, the success likelihood may be a very complicated function of chosen parameters such as temperature and match lengths, ruling out a rigorous analysis. It is conceivable that on certain computations, with carefully chosen parameters, the assembly can have reasonable likelihood of success. The kinetics of assembly via hybridization will vary if the temperature is gradually dropped, and thus we may obtain a transition through such a setting of parameters where the assembly has reasonable likelihood of success. While the lack of a proof for the success of unmediated DNA assembly is unsettling, it may happen that analytic proofs are not essential to resolve the issue. The issue is being resolved (without rigorous analysis) by computer simulations and experimental evidence, on a moderate scale, of the random assembly-disassembly process. Recently Winfree [W97] has done some promising computer simulations of a related random assembly-disassembly model, indicating that careful choice of settings of the parameters may allow unmediated self-assembly sometimes to succeed. Large scale unmediated self-assembly remains experimentally untested and we now await experimental evidence of this.

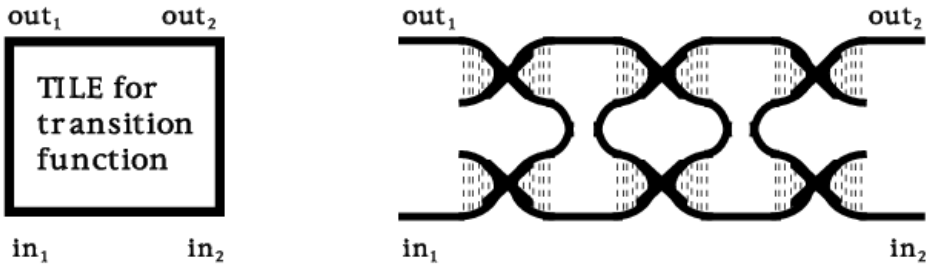


Figure 1: The tile transition function and a DNA DX nano-structure for the tile.

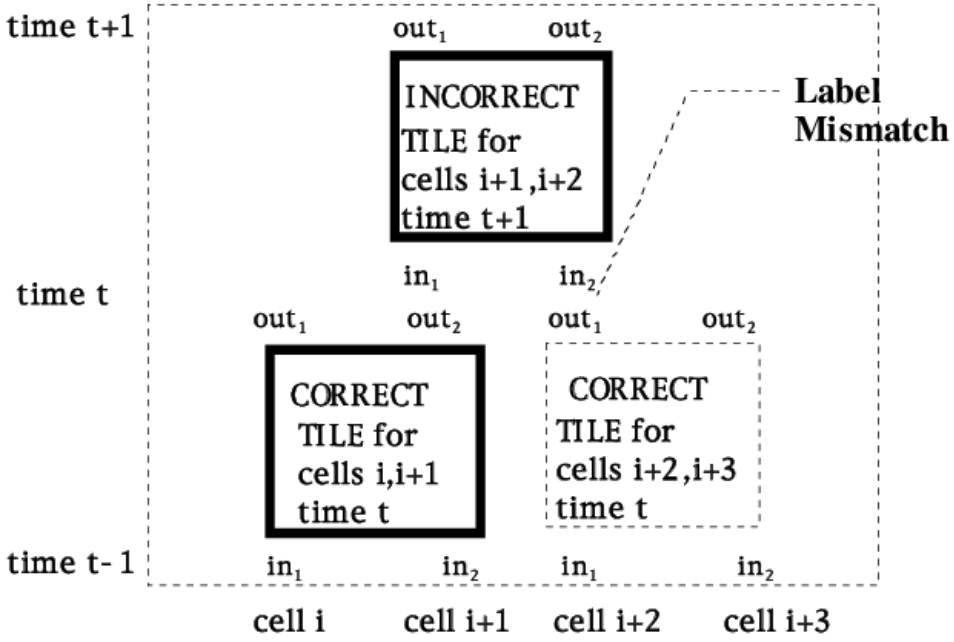


Figure 2: Unmediated self-assembly correctly places a tile at $(i, t)$ but incorrectly places a tile at $(i+1, t+1)$. Later the tile at $(i + 2, t)$ can not be placed, so assembly is blocked.

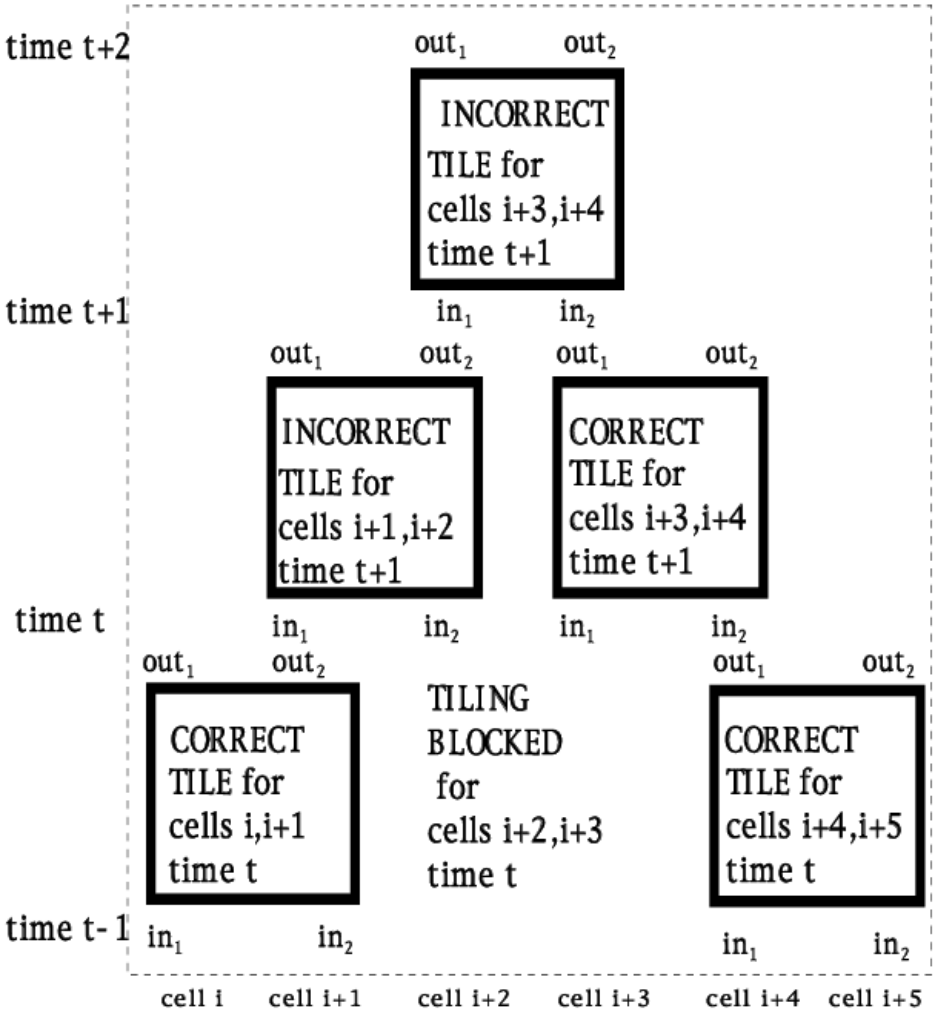


Figure 3: Unmediated self-assembly correctly places a tile at $(i, t), (i + 4, t)$ but incorrectly places a tile at $(i + 1, t + 1)$. Later tiles placed at $(i + 3, t + 1), (i + 2, t + 2)$ reinforce the incorrect tiling, so the tile at $(i + 2, t)$ can not be placed, and the assembly is permanently blocked.

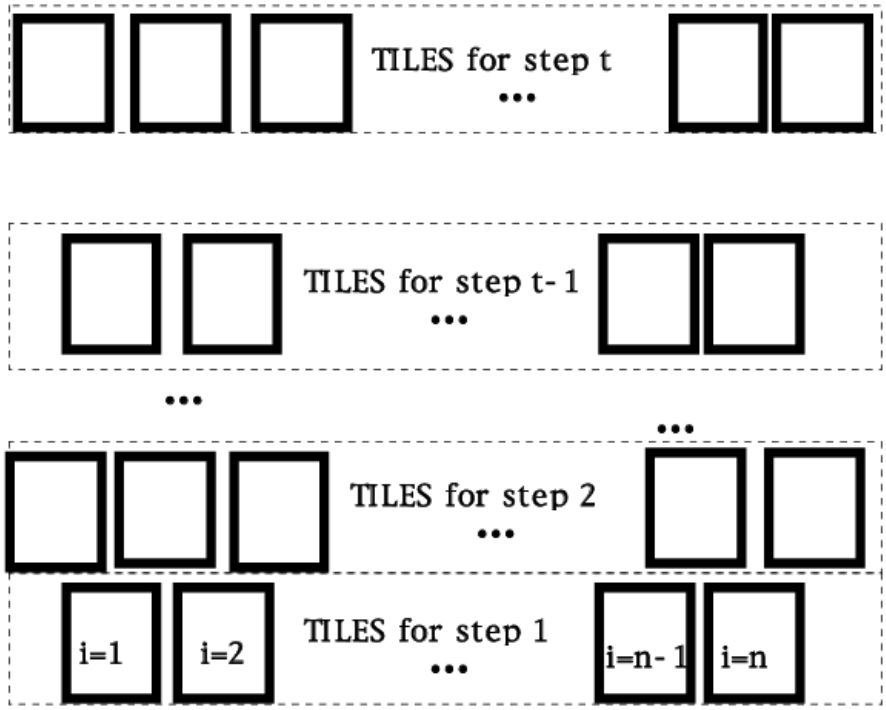


Figure 4: Step-wise assembly for step $t$.

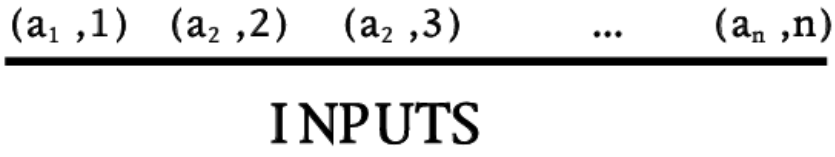


Figure 5: An ssDNA strand encoding input n-vector fixed in a straight line.

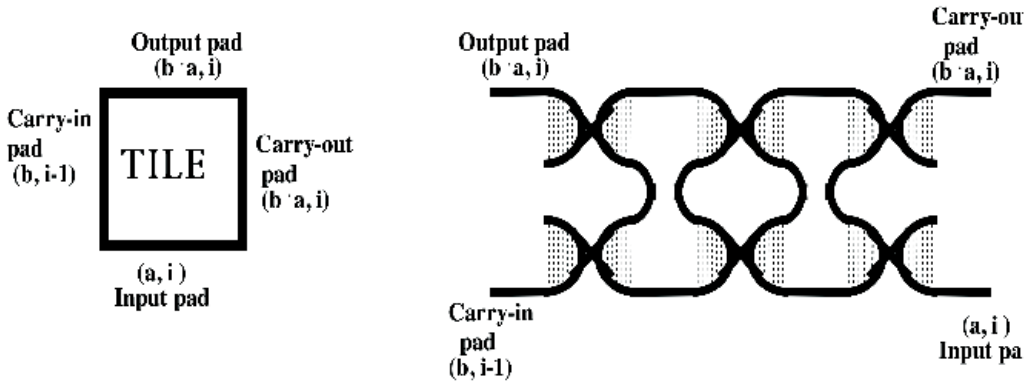


Figure 6: Square tile for sequential prefix computation and a DNA DX nano-structure for the tile.

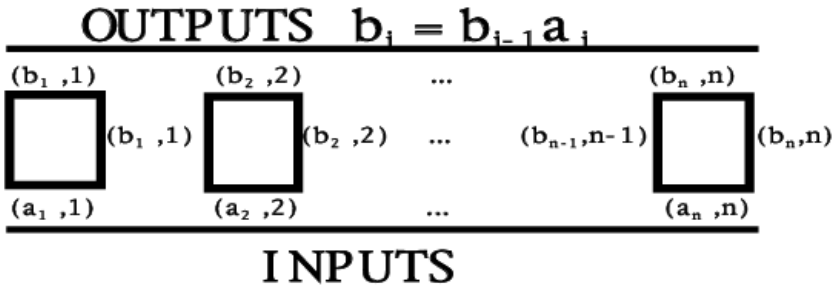


Figure 7: 2D assembly for sequential prefix computation.

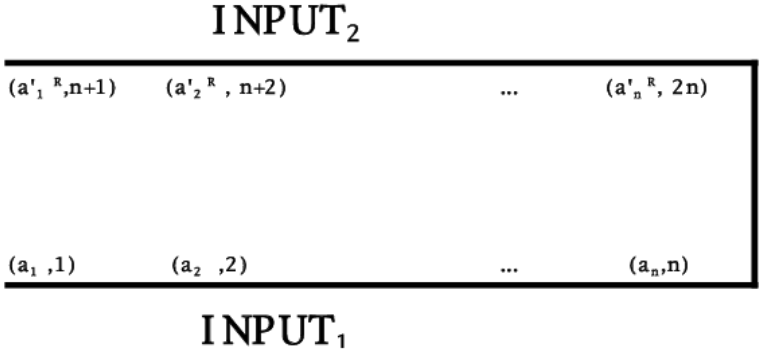


Figure 8: Input ssDNA encoding two binary numbers.

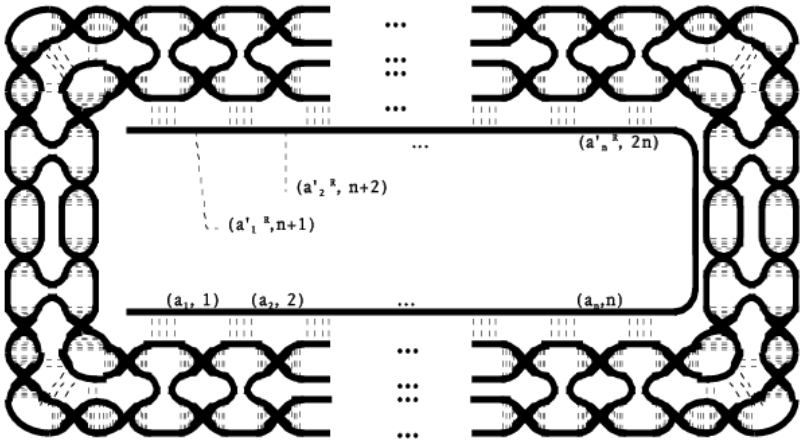


Figure 9: A DNA DX nano-structure for the rectangular frame.

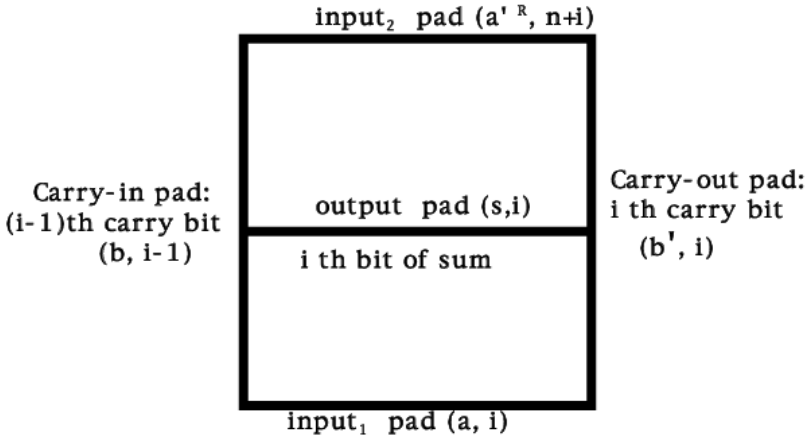


Figure 10: Square tile with middle segment, used for bit-serial addition.

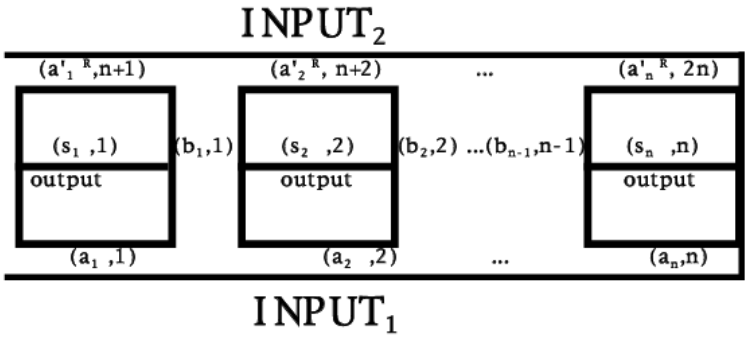


Figure 11: 2D assembly for bit-serial addition.

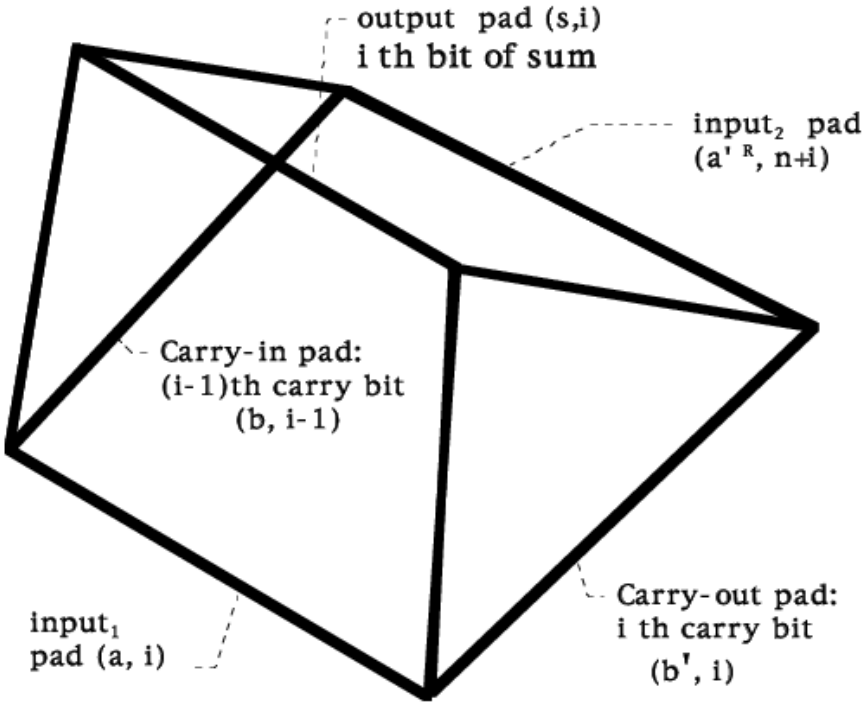


Figure 12: Alternative 3D polyhedral tile for bit-serial addition.

$(a_1,1,0)\ (a_2,2,0)\ (a_3,3,0)\ (a_4,4,0)\ (a_5,5,0)\ (a_6,6,0)\ (a_7,7,0)\ (a_8,8,0)$

Figure 13: Input ssDNA encoding an $n$-vector.

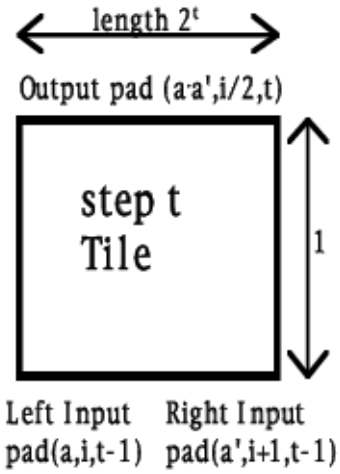


Figure 14: A Rectangular tile for parallel monoid sum computation.

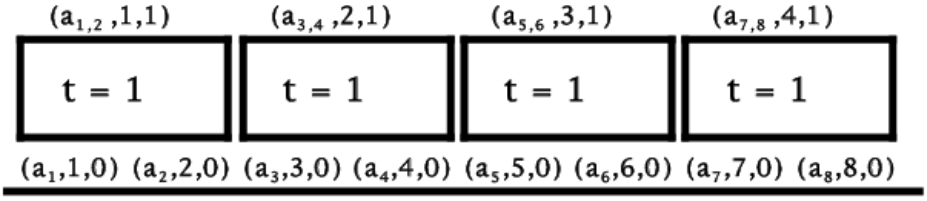


Figure 15: Step $t = 1$ of step-wise assembly for parallel monoid sum computes $a_{i,i+1} = a_i \cdot a_i + 1$ for odd $i$.

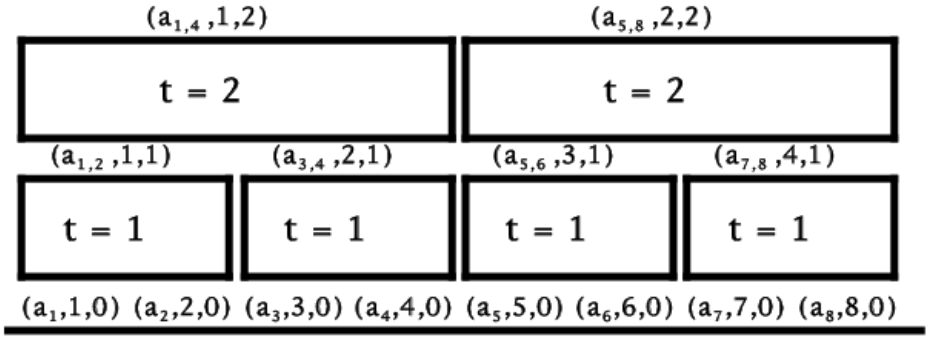


Figure 16: Step $t = 2$ of step-wise assembly for parallel monoid sum computes $b_4 = a_{1,4} = a_1 \cdot a_2 \cdot a_3 \cdot a_4$ and $a_{5,8} = a_5 \cdot a_6 \cdot a_7 \cdot a_8$.

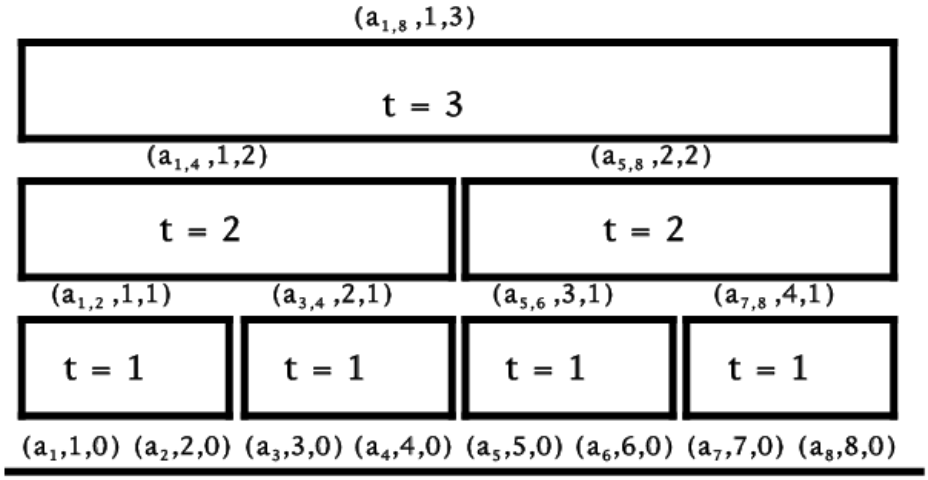


Figure 17: Final step $t = 3$ of assembly for parallel monoid sum computes $b_8 = a_{1,8} = a_1 \cdot a_2 \ldots a_8$.

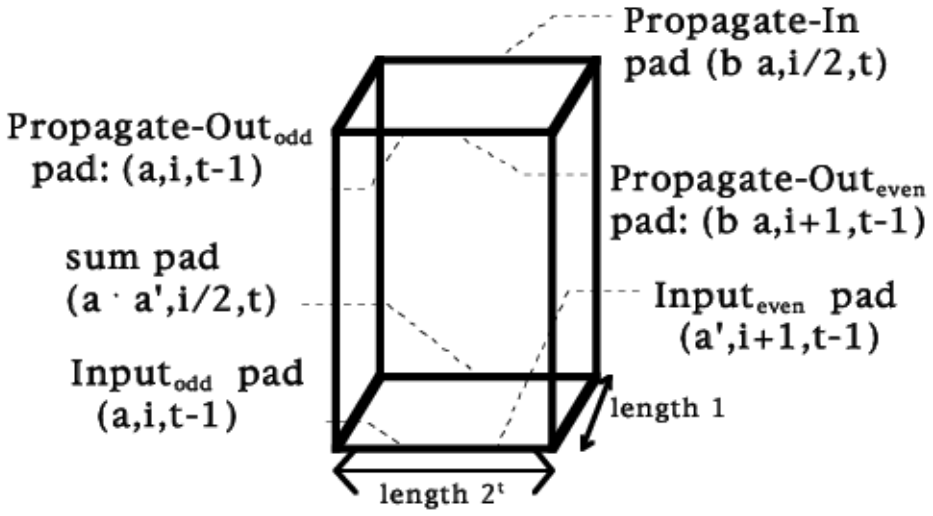


Figure 18: A 3D polyhedron tile for parallel prefix computation.

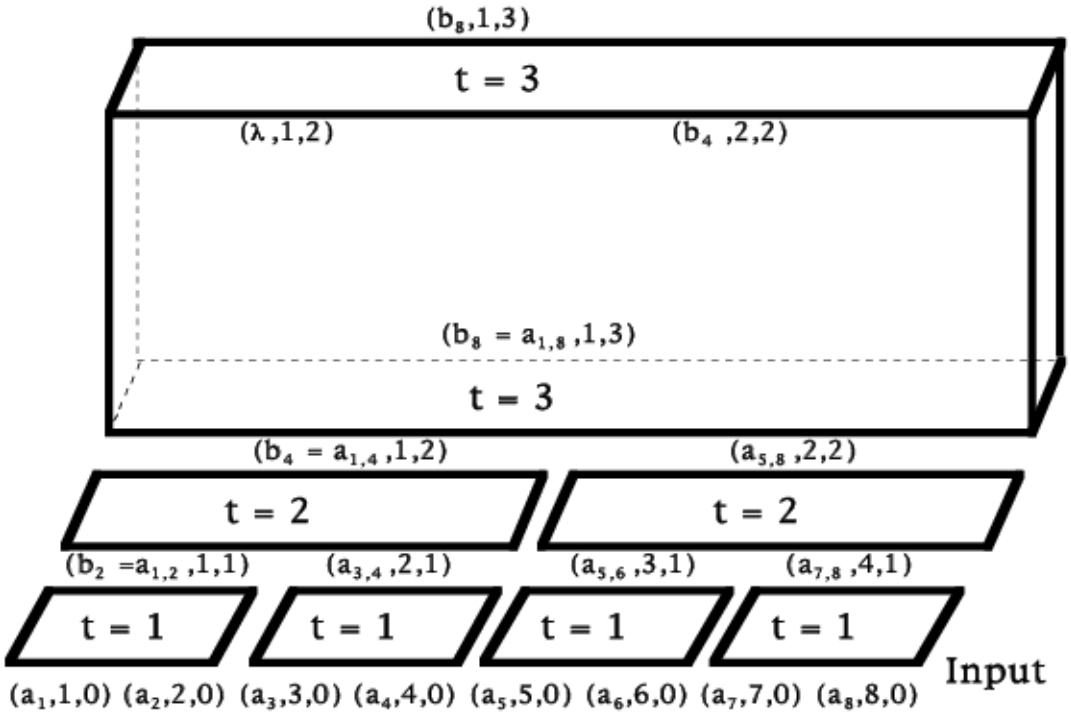


Figure 19: Stage $t = 3$ of 3D parallel prefix assembly: computation of $b_4$.

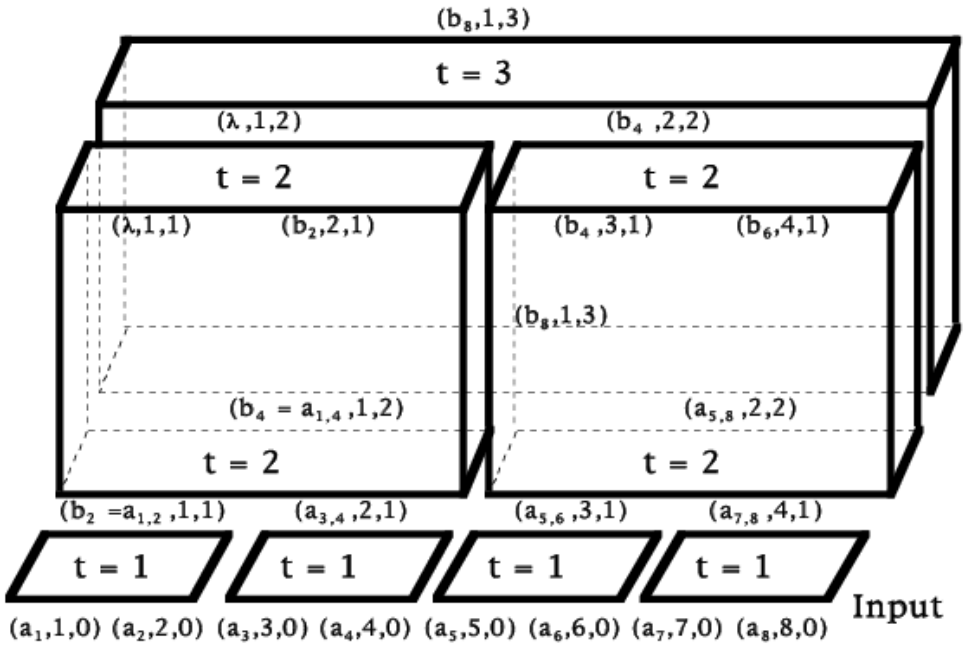


Figure 20: Stage $t = 2$ of 3D parallel prefix assembly: computation of $b_2, b_4$ and $b_6$.

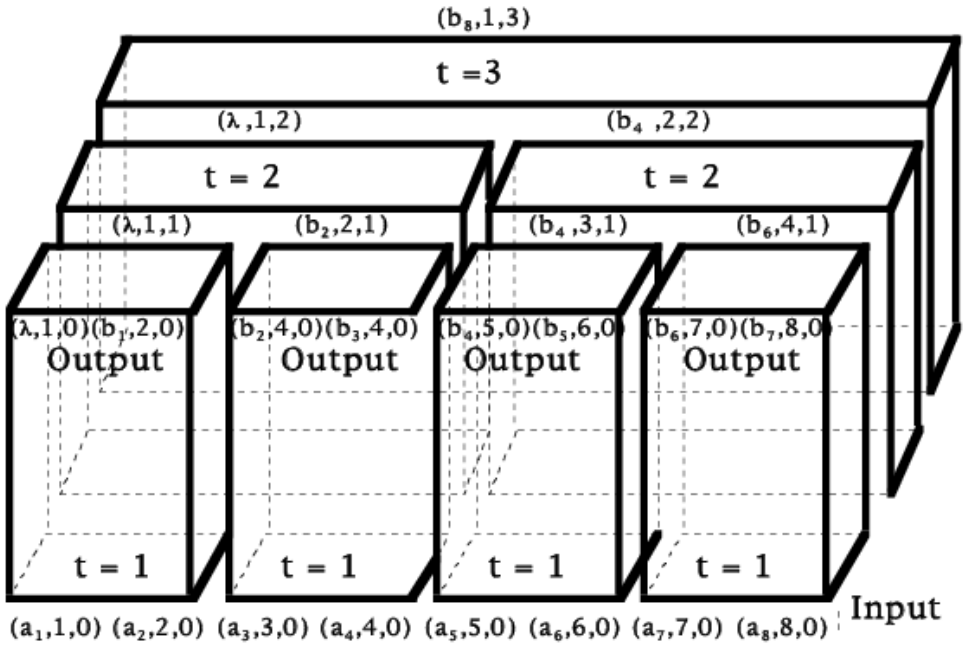


Figure 21: Final stage $t = 1$ of 3D parallel prefix assembly: computation of $b_1, \ldots, b_7$.

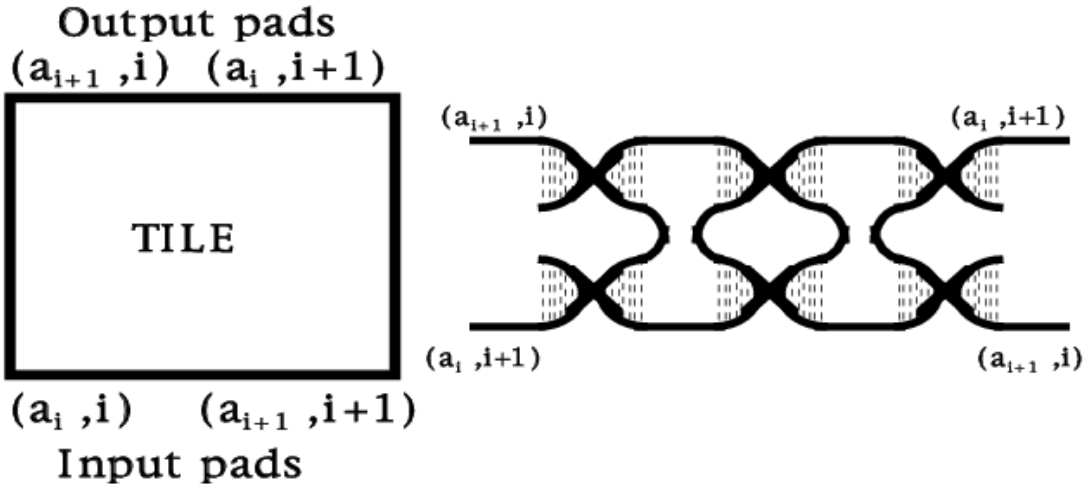


Figure 22: A square tile for pair-wise exchange and a DNA DX nano-structure for the tile.

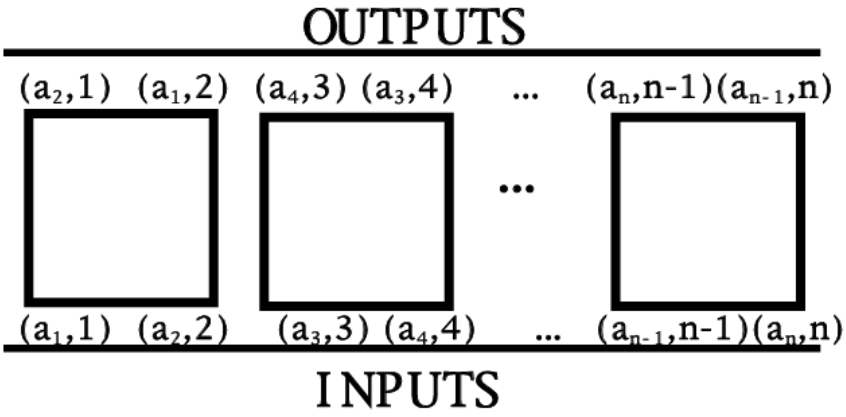


Figure 23: The 2-D assembly for pair-wise exchange.

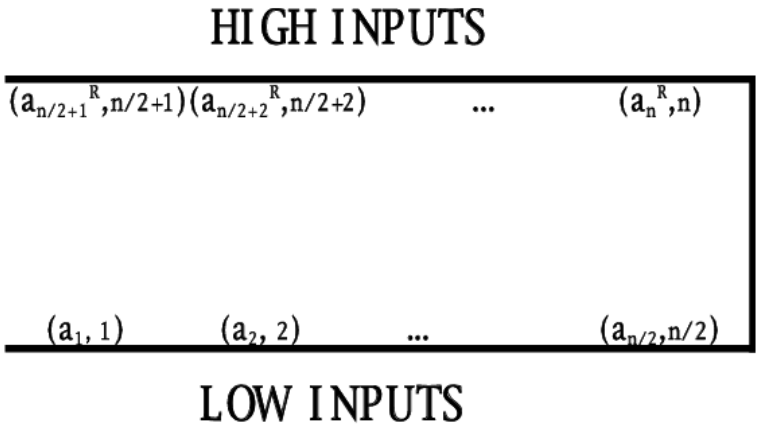


Figure 24: An ssDNA encoding input n-vector.

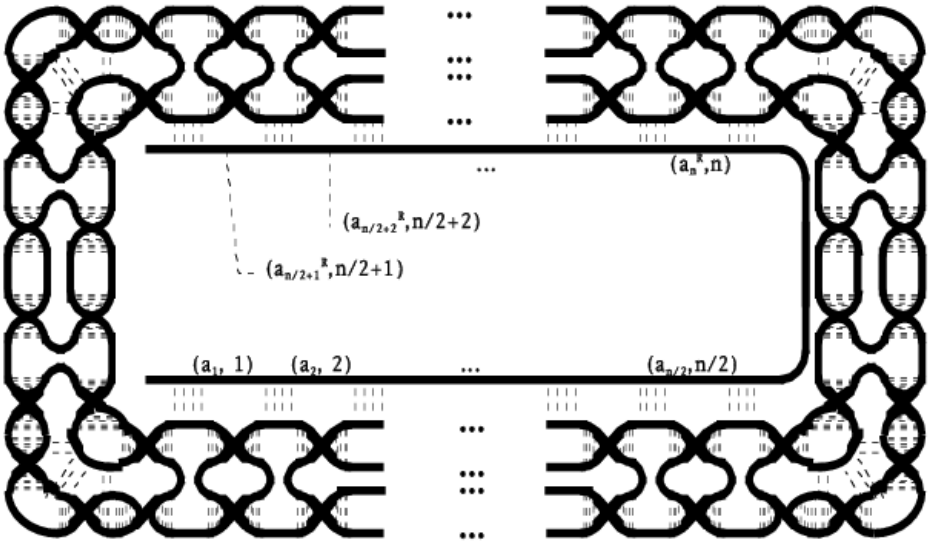


Figure 25: A DNA DX nano-structure for the rectangular frame.

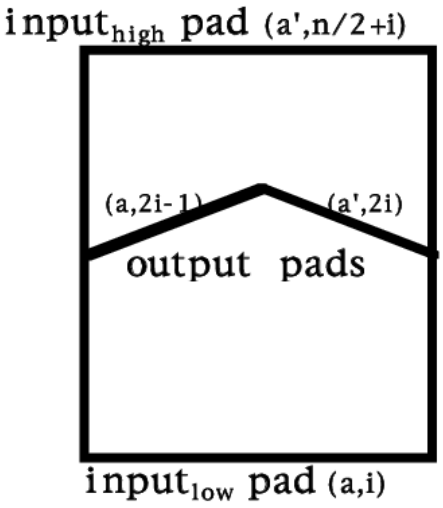


Figure 26: A square tile for perfect shuffle, with middle segment used for output.

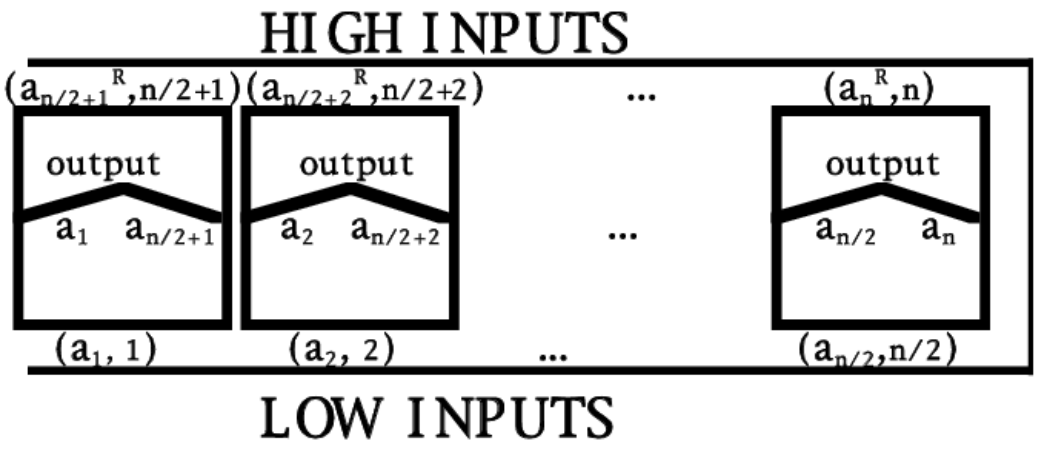


Figure 27: 2D assembly for perfect shuffle. The middle segments give output.

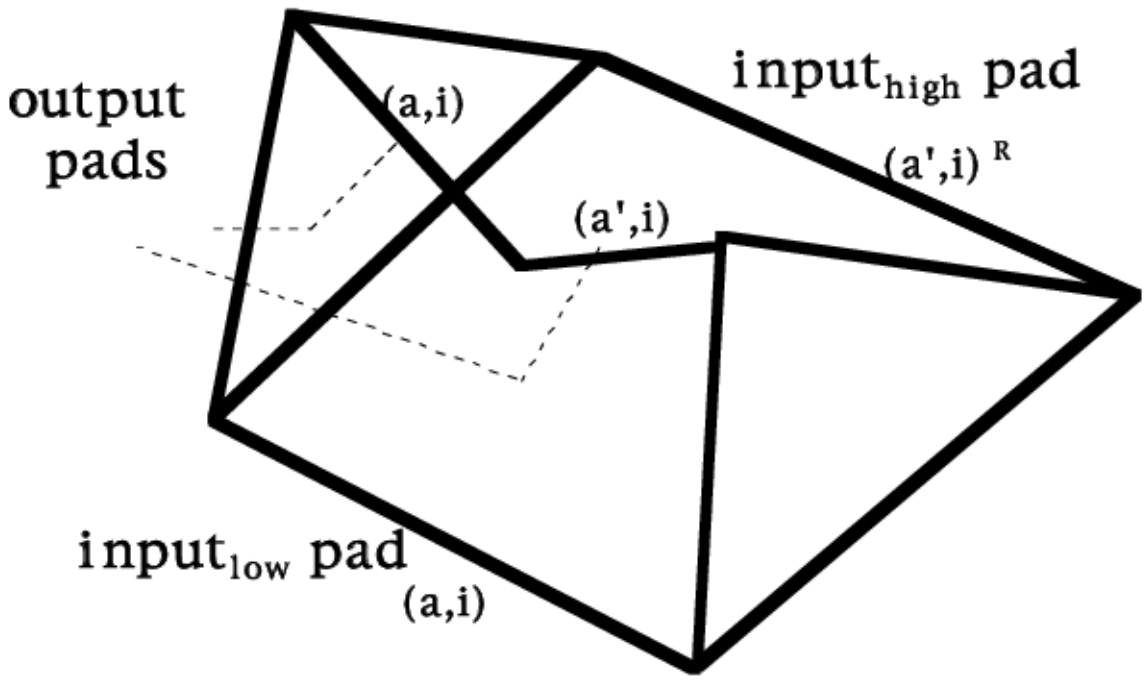


Figure 28: Alternative 3D polyhedral tile for perfect shuffle.