Parallel Biomolecular Computation: Models and Simulations[1]

J. H. Reif[2]

Abstract. This paper is concerned with the development of techniques for massively parallel computation at the molecular scale, which we refer to as *molecular parallelism*. While this may at first appear to be purely science fiction, Adleman [Ad1] has already employed molecular parallelism in the solution of the Hamiltonian path problem, and successfully tested his techniques in a lab experiment on DNA for a small graph. Lipton [L] showed that finding the satisfying inputs to a Boolean expression of size n can be done in $O(n)$ lab steps using DNA of length $O(n \log n)$ base pairs. This recent work by Adleman and Lipton in molecular parallelism considered only the solution of NP search problems, and provided no way of quickly executing lengthy computations by purely molecular means; the number of lab steps depended linearly on the size of the simulated expression. See [Re3] for further recent work on molecular parallelism and see [Re4] for an extensive survey of molecular parallelism.

Our goal is to *execute lengthy computations quickly by the use of molecular parallelism.* We wish to execute these biomolecular computations using short DNA strands by more or less conventional biotechnology engineering techniques within a small number of lab steps. This paper describes techniques for achieving this goal, in the context of well defined abstract models of biomolecular computation. Although *our results are of theoretical consequence only, due to the large amount of molecular parallelism* (*i.e., large test tube volume*) *required*, we believe that our theoretical models and results may be a basis for more practical later work, just as was done in the area of parallel computing.

We propose two abstract models of biomolecular computation. The first, the Parallel Associative Memory (PAM) model, is a very high-level model which includes a Parallel Associative Matching (PA-Match) operation, that appears to improve the power of molecular parallelism beyond the operations previously considered by Lipton [L]. We give some simulations of conventional sequential and parallel computational models by our PAM model. Each of the simulations use strings of length $O(s)$ over an alphabet of size $O(s)$ (which correspond to DNA of length $O(s \log s)$ base pairs). Using $O(s \log s)$ PAM operations that are not PA-Match (or $O(s)$ operations assuming a ligation operation) and t PA-Match operations, we can:

1. simulate a nondeterministic Turing Machine computation with space bound s and time bound $2^{O(s)}$, with $t = O(s)$,
2. simulate a CREW PRAM with time bound D, with M memory cells, and processor bound P, where here $s = O(\log(PM))$ and $t = O(D + s)$,
3. find the satisfying inputs to a Boolean circuit constructible in s space with n inputs, unbounded fan-out, and depth D, where here $t = O(D + s)$.

[1] A preliminary version of this paper appears in *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures* (SPAA95), Santa Barbara, CA, June 1995, pages 213–223. This research was supported by NSF/DARPA CCR-9725021, CCR-96-33567, NSF IRI-9619647, ARO Contract DAAH-04-96-1-0448, ARPA/SISTO Contracts N00014-91-J-1985, and N00014-92-C-0182 under Subcontract KI-92-01-0182. This paper, in postscript, can be found at http://www.cs.duke.edu/~reif/paper/Molecular.ps and figures can be found at http://www.cs.duke.edu/~reif/paper/mole.fig.ps.

[2] Department of Computer Science, Duke University, Durham, NC 27708-0129, USA. reif@cs.duke.edu.

Preprint of paper appearing in Algorithmica, special issue on Computational Biology, Vol. 25, No. 2, 142-176, 1999.

We also propose a Recombinant DNA (RDNA) model which is a low-level model that allows operations that are abstractions of very well understood recombinant DNA operations and provides a representation, which we call the *complex*, for the relevant structural properties of DNA. The PA-Match operation for lengthy strings of length s cannot be feasibly implemented by recombinant DNA techniques directly by a single step of complementary pairing in DNA; nevertheless we show this Matching operation can be simulated in the RDNA model with $O(s)$ slowdown by multiple steps of complementary pairing of substrings of length 2 (corresponding to logarithmic length DNA subsequences). Each of the other operations of the PAM model can be executed in our RDNA model, without slowdown.

We further show that, with a further $O(s)/\log(1/\varepsilon)$ slowdown, the simulations can be done correctly with probability $1/2$ even if certain recombinant DNA operations (e.g., Separation) can error with a probability ε. We also observe efficient simulations can be done by PRAMs and thus Turing Machines of our molecular models.

Key Words. Parallel computation, Parallel RAM, Nondeterministic computation, NP, Biomolecular computation, Biotechnology, DNA, Recombinant DNA.

1. Introduction

1.1. *Solving NP Search Problems by Molecular Parallelism.* Feynman [F] first proposed doing computation via molecular means, but his idea was not brought to test for a number of decades. The Hamiltonian path problem is to find a path in a graph that visits each node exactly once; it is a special case of the Traveling Salesman where the problem is to find the shortest such path in a network with positively weighted edges. Adleman's technique [Ad1] (also see comments in [G]) to solve a Hamiltonian path problem of n nodes and m edges required $O(n + m)$ lab steps employing short DNA strands with $O(n \log n)$ base pairs (see Section 6.1 for definition of DNA base pairs). Adleman was the first to do an experiment demonstrating biomolecular computation, solving by his method the Hamiltonian path problem for a graph of seven nodes. This was a major milestone in biomolecular computation.

Lipton [L] showed that finding the satisfying inputs to a Boolean expression of size n can be done in a linear number of lab steps. His method used DNA strands with $O(n \log n)$ base pairs, and requires a number of lab steps which depends *linearly on the size* (*rather than the depth*) *of the simulated expression*. Also, Beaver [Be1] made similar use of molecular parallelism in the solution of the integer factorization problem. These contributions and the work of Adleman in molecular parallelism have considered only the solution of NP search problems and provided no way of *quickly* executing lengthy computations by purely molecular means. They all used an exponential number of DNA strands.

Applegate and others have solved the Traveling Salesman and Hamiltonian path problems for problems of size well over 1000 cities, by the use of conventional high performance workstations; these methods avoid brute-force search and use instead sophisticated heuristics. This indicates that even for exact solutions of NP complete problems we require a more general type of biomolecular computation than simply brute-force search. For example, we would like the biomolecular computation to be general enough to implement such sophisticated heuristics in parallel. (See also Section 1.3 for a further discussion of independent work and also subsequent work since this paper was presented [Re2] at the SPAA95 conference.)

1.2. *New Results of This Paper.* Biomolecular computations have the potential to make use of molecular parallelism, on a wide class of computational problems well beyond NP search problems. We show that techniques used by computer scientists in the design of parallel algorithms may be quite useful in the new area of biomolecular computation. Since the number of elements within a test tube is limited to a large finite number of approximately 10^{20}, our results stated in the abstract are of theoretical consequence only, due to the large amount of molecular parallelism required. However, we believe that our theoretical models and results may be a basis for more practical later work, just as was done in the area of parallel computing. (Recently Gehani and Reif [GR] developed techniques that significantly decrease the volume required for our simulations; in particular, to implement the PAM operation they use microflow MEMS devices to route together DNA strands with complementary segments.)

It may also be of interest to biochemists that our low level molecular simulations use certain technical recombinant DNA techniques similar to those used for plasmids, which are small circular DNA found in bacteria (see pp. 46–69 of [OP], pp. 27–28 of [WGWZ], and pp. 129 and 168 of [Si]). Indeed, our construction was inspired by nature's and subsequent recombinant DNA engineers' ingenious use of plasmids for various basic DNA manipulations.

Potential applications of biomolecular computations considered in this paper include: (1) simulation of sequential space bounded nondeterministic computation, (2) PRAM emulation, and (3) parallel circuit satisfaction.

The molecular models of Adleman and Lipton, as well as our Recombinant DNA (RDNA) model, restrict matching by Separation operations to complementing strings of constant length that correspond to DNA of logarithmic base pair length. This is because in the laboratory, short matches of complementing strings can be done by the usual recombinant DNA techniques, but lengthy matches of complementing strings may be not done with surety.

In particular, our implementations in the RDNA model require only Separation operations using complementary matching of strings of length 2 which are represented in DNA as short (of length $O(\log s)$, where the size of the alphabet is $O(s)$ and s is a parameter of the simulations defined in the abstract) DNA strings. Our simulations in the RDNA model use encoding techniques to distinguish the position of matched characters or substrings in strings, so we can effectively match long strings by performing multiple short matches.

1.3. *Further Recent Related Work.* Recently Baum [Bau] has described how to build a large associative memory using molecular techniques.

A number of researchers, including Csuhaj-Varju et al. [CFKP], Rothemund [Rot], and Smith and Schweitzer [SS] recently also independently proved that a universal Turing Machine (TM) can be simulated by recombinant DNA operations, but gave no proof of a general speed up by biomolecular computation. Beaver [Be2], Papadimitriou [P], and Reif [Re2] independently proved that any lengthy linear space bounded sequential computation can be exponentially speeded up by PMC; in particular, they also showed that sequential TM computations with space s and time up to $2^{O(s)}$ can be simulated by PMC in polynomial time using constructions similar to our PAM simulation. In particular, Beaver [Be2], Papadimitriou [P], and Reif [Re2] all made independent use of the *pointer jumping* technique, which itself dates to the 1978 work of Fortune and Wyllie [FW], who gave a parallel simulation of a space bounded TM by a PRAM. This same technique of pointer jumping is essential also for our molecular simulations of PRAMs.

Baum [Bau] and Papadimitriou [P] implicitly assume a molecular model (similar to our Parallel Associative Memory (PAM) model) for recombinant DNA which can do *reliable* complementary matching of lengthy (length s) DNA strings in one step; but provide no detailed implementation into DNA to justify this assumption. To abide by the molecular models of either Adleman, Lipton, or our RDNA model, implementations in DNA which do complementary matching of length s DNA strings in a single step (without the encoding techniques developed for our simulation in the RDNA model) would appear to be restricted to only very short string segments, in particular of logarithmic length; thus allowing the simulation of only logarithmic space sequential computations.

In contrast, our paper considers simulations in the very high level PAM model, and we have described in detail its implementation by recombinant DNA operations. The molecular simulation proof of Beaver [Be2] may also be feasible. To implement his simulation in DNA, he executes a DNA string-editing operation on length $O(s \log s)$ DNA strings using $O(s)$ recombinant DNA operations, each of which does site-directed (local) mutagenesis (see pp. 192–193 of [WGWZ], pp. 191–206 of [OP], and Chapter 5 of [SFM]).

1.4. *Organization of the Paper.* In the Introduction we have reviewed previous results in the solution of NP problems and circuit satisfaction by molecular parallelism, and described our proposed applications of biomolecular computation. In Section 2 we describe abstract models for biomolecular computation including the previous models of Lipton and Adleman, as well as our PAM model. In Section 4 we describe fast parallel molecular simulation in our PAM model of lengthy space bounded nondeterministic sequential computations, of parallel RAMs and satisfaction of circuits.

Further, we describe simulations of the PAM model by conventional computation models. In Section 5 we define our RDNA model, and we describe implementation of our PAM algorithms in the RDNA model. In Section 6 we describe implementation of our RDNA model algorithms in recombinant DNA. In Section 7 we discuss some open problems. In Appendix A we discuss recombinant DNA and RNA manipulation technology, and in Appendix B we consider the resource bounds limiting biomolecular computation.

2. Abstract Molecular Models.

Here we discuss a number of abstract models for biomolecular computation. We first describe our PAM model. Then we briefly discuss and compare Lipton and Adleman's biomolecular computation models and operations. The main difference between these models is the introduction of the Parallel Associative Matching (PA-Match) operation $\bowtie$ in our model, which is essentially identical to the join operation (also denoted $\bowtie$) for relational databases, except that our PA-Match operation is done with respect to encoded strings. Later in Sections 5 and 6 we describe the efficient implementation of this PA-Match operation by recombinant DNA.

2.1. *The PAM Model for Biomolecular Computation.* Here we propose an apparently more powerful abstract model of biomolecular computation, which we call the *Parallel*

Associative Memory (PAM) model. Our PAM model is an extension of the model of Lipton [L] and the Memory model of Adleman [Ad2] to allow for more dynamic changes in the memory. In particular, we give (a) an extension of the Separation operation to allow for pattern strings of constant length $c_0 = O(1)$, and (b) a PA-Match operation. This operation maps naturally to lower-level recombinant DNA operations and distills the computational essence of lower-level operations involving complementary matching.

A *multiset* is a collection of elements that may be repeated. (Set operations such as union can be generalized to multisets in the natural way.) A *test tube* T is defined to be a multiset with elements[3] from a finite set $\mathcal{E}$. For the PAM model, it suffices that an element of a test tube be just a string used to model the information content of a DNA strand, so we let the set $\mathcal{E}$, of all possible elements a test tube, be the set of strings over a finite alphabet Σ.

We fix the alphabet $\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_{n-1}, \bar{\sigma}_0, \bar{\sigma}_1, \ldots, \bar{\sigma}_{n-1}\}$ of $2n$ distinct symbols where for each i, the symbols $\sigma_i, \bar{\sigma}_i$ are called *complements* (these will be use to model Watson–Crick complementary pairing). *Here n is a parameter used throughout the paper to denote the number of distinct complementing symbol pairs in* Σ. For each symbol $\sigma \in \Sigma$, we also let $\bar{\sigma} = \sigma_i$ if $\sigma = \bar{\sigma}_i$. Note that the Σ has $2n$ elements, whereas the number of DNA base pairs is 4. Hence a string over Σ of length k may be used to represent the information content of a DNA or RNA chain of $k \log_4(2n) = (k/2)\log(2n)$ base pairs.

Our PAM model allows any of the operations below to be executed in one step.

1. *Merge*. Given tubes T_1, T_2, produce the union $T_1 \cup T_2$.
2. *Copy*. Given a tube T_1, produce a tube T_2 with the same contents.
3. *Detect*. Given a tube T, say “yes” if T contains at least one element and say “no” if it contains none.
4. *Separation*. Given a tube T and a string α of length at most $c_0 = O(1)$ symbols of Σ, produce a tube consisting of all strings of T which contain α, and also a tube consisting of all strings of T which do not contain α. To decrease the likelihood of separation errors by mismatches, we restrict the Separation operation to match with a pattern string α of only constant length; in fact all our simulation results require only that the pattern length be $c_0 = 2$. For example, if the string σ is not too long, we can separate all single-stranded DNA of the form $\alpha\sigma\beta$.
5. *PA-Match*. Given tubes T, T', apply the operation $\bowtie$, defined in Section 2.2, to yield the tube $T \bowtie T'$.

[3] This definition of the the set $\mathcal{E}$, to be a set of strings over a finite alphabet, is all we require for our complexity and simulation results for the PAM model. In general, it may be useful in other models to allow an element of a test tube to be any fixed, finite representation of a molecule; the representation may be used to model both the information content as well as certain structural properties of a molecule. For example, in our RDNA model, which we define in Section 5, an element of a test tube is a structure, which we call a *complex*, used to model the information content and certain key properties (such as hybridization and secondary structure) of the three-dimensional structure of single- or double-stranded DNA, or RNA that may not be well represented simply by linear strings. Other representations for elements of a test tube might be used to model key properties of the three-dimensional structure of protein or even inorganic material.

2.2. *Parallel Associative Matching.* Let $\mathcal{E}$ be the set of all possible elements of a test tube, which we view as strings over alphabet Σ of size $2n$. Fix a finite *associative match alphabet* A of size $a = |A|$ (not to be confused with the alphabet Σ used by elements of test tubes). Let A^s be the set of strings of length s over A. For each $\alpha \in A^s$, let $E(\alpha) \in \mathcal{E}$ denote an element of a test tube which provides an encoding (i.e., a unique finite representation) of the string α. Also, for each $\alpha, \beta \in A^s$, let $E(\alpha, \beta) \in \mathcal{E}$ denote an element of a test tube which provides an encoding of the ordered pair of strings (α, β). Thus E is a mapping from strings and pairs of strings over A to $\mathcal{E}$.

For each $\alpha, \beta, \beta', \gamma \in A^s$, we define the *PA-Match* operation $\bowtie$ to be $E(\alpha, \beta) \bowtie E(\beta', \gamma) = E(\alpha, \gamma)$ when $\beta = \beta'$, and the operation $\bowtie$ yields no value if $\beta \neq \beta'$. Special cases of the PA-Match operation are $E(\beta) \bowtie E(\beta', \gamma) = E(\gamma)$ and $E(\gamma, \beta') \bowtie E(\beta) = E(\gamma)$ where $\beta = \beta'$, and the operation $\bowtie$ again yields no value if $\beta \neq \beta'$. For two tubes T, T' consisting of strings encoding pairs by $E(-, -)$, let $T \bowtie T' = \{x \bowtie x' \mid x \in T, x' \in T'\}$. Thus the PA-Match operation is essentially identical to the join operation for relational databases, except that the operation is done with respect to sets of encoded strings rather than relations.

2.3. *Comparison with the Models of Lipton and Adleman.* Lipton [L] defined the first abstract model of biomolecular computation. The elements of his test tubes are strings as in the PAM model of Section 2.1. His model, which we call the *Test Tube Model*, allows the unit step execution of any of the operations Merge, Copy, Detect, and Separation, as well as a *ligation* operation that allows formation of the contents of the initial test tubes by combining matching strings (for details, see [L]). The subsequent *Memory* model of Adleman [Ad2] allowed all of the operations of Lipton's model, except with only an implicit Copy (or Amplify) operation, and allowed test tubes to have a more general class of elements beyond just strings. It should be emphasized that the PA-Match operation is a significant generalization of the Detect and Separation operations of the Lipton/Adleman models.

2.4. *Resource Bounds of Abstract Molecular Models.* The key resource bounds of our (and the previous) abstract models for biomolecular computations are:

1. number of *steps* L required by a molecular algorithm,
2. *size* $|T|$ *of the test tube* T which is the total number of elements of T including replications, and
3. *maximum size of an element* of the test tube T.

3. PAM Computations.

Here we give some technical definitions and constructions that will be used in the proofs of our simulation results.

3.1. *A Standard String Encoding of String Pairs.* To encode strings of length s over the pattern matching alphabet A for our PA-Match operation, let each character $\sigma \in A$ be represented as a number in $\{0, \ldots, a-1\}$. We define the encoding alphabet $\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_{n-1}, \bar{\sigma}_0, \bar{\sigma}_1, \ldots, \bar{\sigma}_{n-1}\}$ where $n = 2sa + 8 + c_1$, where $c_1 = O(1)$.

(The last $8 + c_1$ distinct characters of Σ are not actually needed for our PAM simulations, but will be used in the recombinant DNA implementation of the PA-Match operation.)

We encode each string $\alpha \in A^s$, where $\alpha = \alpha_1 \cdots \alpha_s$, as $E(\alpha) = E_1(\alpha_1) \cdots E_s(\alpha_s)$ where $E_i(\alpha_i) = \sigma_{\alpha_i+(i-1)a}$, and we also define a distinct alternative encoding $\hat{E}(\alpha) = \hat{E}_1(\alpha_1) \cdots \hat{E}_s(\alpha_s)$ where $\hat{E}_i(\alpha_i) = \sigma_{\alpha_i+(i-1+s)a}$ for each $i = 1, \ldots, s$.

We reserve eight distinct *padding symbols*: $\#_j = \sigma_{2sa+j}$ and $\#'_j = \sigma_{2sa+j+4}$ for $j = 0, \ldots, 3$. The padding symbols $\#_j, \#'_j$ where $j = 0, \ldots, 3$ are termed *distinguished* padding symbols. Then to encode any string pair $\alpha, \beta \in A^s$ for the PA-Match operation, we generally use a *standard encoding* which may be a length $2s + 4$ string of the form $E(\alpha, \beta) = \#_1 E(\alpha)\#_0\#'_0\hat{E}(\beta)\#_2$. Note that this standard encoding consists of a string of distinct symbols, and, moreover, from each symbol we can determine its position in the original strings α, β.

We also extend the definition of standard encodings to *allow standard encodings to use the above form with any fixed permutation of the symbols of* Σ. Note: this extension of the standard encoding is of technical use in the proof of Lemma 5.1, where we implement the PA-Match operation, and we require only those additional standard encodings derived by permuting the distinguished padding symbols or permuting together all symbols σ_k with the symbols σ_{k+sa}, for all $k = 0, \ldots, sa - 1$. This standard encoding is used in the computational simulations we construct in this paper.

3.2. *Initial Contents of the Test Tubes in the PAM Computations.*

We assume a PAM computation may begin with a finite set of initial test tubes which are restricted to contain sets of form A^s for some finite alphabet A; that is all strings of a given length s over A. Some of these initial test tubes are distinguished as *input* test tubes, whose strings are standard encodings that provide the input to the PAM computation. The remaining initial test tubes do not necessarily need be sets of standard encodings. We have thus chosen essentially the weakest initial assumption possible that can be made. (An alternative assumption, used by Lipton [L] and Adleman [Ad2], and sometime called the ligation model, would be to allow test tubes to be constructed by ligations of a finite set of oligos.)

3.3. *Constructions of Encodings*

Lemma 3.1. *Given a tube T consisting of a subset of $E(A^s)$ (not necessarily a set of standard encodings), a test tube can be constructed in $O(s)$ Separation and Merge operations, consisting of all elements of T such that*

1. *there is one and only one symbol at each position i in an encoding, and*
2. *the symbols in an encoding are listed consecutively.*

Note that this lemma is immediate if we assume the ligation model mentioned in Section 3.2, however, our weaker assumptions for the PAM model require that we provide a proof.

PROOF. We initialize with test tube $T_0 = T$. We recursively construct a sequence of test tubes $T_0, T_1, \ldots, T_s$, where we construct each T_i by restricting the contents from T_{i-1}.

For $i = 1, \ldots, s$, **do**

1. For each character $u \in A$, construct tube $T_i[u]$ by copying T_{i-1} and applying the Separation operation with the character $\sigma_{u+(i-1)a}$. Then construct a test tube $T_{i,1}$ that is the merge of all these test tubes $\bigcup_{u \in A} T_i[u]$. (This ensures that each encoded configuration has an encoding of a configuration symbol at position i.)
2. **If** $i = s$ **then** let $T_i = T_{i,1}$ and **else do**
   For each pair of characters $u, v \in A$, construct tube $T_i[u, v]$ by copying $T_{i,1}$ and applying the Separation operation with the length 2 string $\sigma_{u+(i-1)a}\sigma_{v+ia}$. Then construct a test tube T_i that is the merge of all the test tubes $\bigcup_{u,v \in A} T_i[u, v]$. (This ensures that each encoding of a configuration has its symbols listed consecutively. Note that this is the only place in our PAM simulations where we need to apply the Separation operation with the length 2 string rather than to a single character. This step is not actually needed in our PAM simulations if we extend the standard encoding to allow for all possible permutations of the encoded symbols.)

OUTPUT T_s. □

3.4. *Shift-Invariant Predicate Testing in the PAM Model.* Fix $c = O(1)$ distinct integer *shift* constants $\delta_1, \ldots, \delta_c \geq 0$. Let an index i be *applicable* if for $j = 1, \ldots, c$ all the indices $i + \delta_j$ are in the range from 1 to s; note that this occurs if $1 \leq i + \max_{1 \leq j \leq c} \delta_j \leq s$. A set of strings $S \subseteq A^s$ is defined by a *shift-invariant predicate* if there exists a predicate $p(x_0, x_1, \ldots, x_c)$ and shift constants $\delta_1, \ldots, \delta_c$ such that $S = \{\alpha \in A^s \mid p(\alpha_i, \alpha_{i+\delta_1}, \ldots, \alpha_{i+\delta_c})$ holds for each applicable $i = 1, \ldots, s-1\}$. In other words, for each $\alpha = \alpha_1 \cdots \alpha_s$, where $\alpha_1, \ldots, \alpha_s \in A$, we have $\alpha \in S$ iff the predicate $p(\alpha_i, \alpha_{i+\delta_1}, \ldots, \alpha_{i+\delta_c})$ holds for each applicable $i = 1, \ldots, s-1$.

LEMMA 3.2. *Given a tube T consisting of a subset of $E(A^s)$, for any set S defined by a shift-invariant predicate, a test tube $E(S) \cap T$ can be constructed in $O(|A|^{c+1}s)$ Separation and Merge operations.*

PROOF. By Lemma 3.1, given a tube T consisting of a subset of $E(A^s)$, a test tube T' can be constructed in $O(s)$ Separation and Merge operations, consisting of all elements of T such that there is one and only one symbol at each position i in an encoding, and the symbols in an encoding are listed consecutively. We initialize with test tube $T'_0 = T'$. We recursively define a sequence of test tubes $T'_0, T'_1, \ldots, T'_s$, where we construct each T'_i by restricting the contents from T'_{i-1}.

For $i = 1, \ldots, s$, do
If i is not applicable **then** let $T'_i = T'_{i-1}$ **else if** i is applicable **then do**

1. For each $u_0, \ldots, u_c \in A$ such that $p(u_0, \ldots, u_c)$ holds, construct tube $T_i'[u_0, \ldots, u_c]$ by copying T_{i-1}' and applying the Separation operation first with the character $\sigma_{u_0+(i-1)a}$, then characters $\sigma_{u_j+(i+\delta_j)a}$ for $j = 1, \ldots, c$.
2. Then construct a test tube T_i' that is the merge of all these test tubes $\bigcup_{u_0,\ldots,u_c \in A} T_i'[u_0, \ldots, u_c]$. (This ensures that the shift-invariant predicate is applied at each applicable position.)

OUTPUT T_s'. □

Let a relation $R \subseteq A^s \times A^s$ be *defined by a shift-invariant predicate* if the set $\{\alpha\beta \mid (\alpha, \beta) \in R\}$ is defined by a shift-invariant predicate. (For example, the set $S = \{\alpha\beta \mid \alpha = \beta, \text{ for } \alpha, \beta \in \{0, 1\}^s\}$ and thus relation $R = \{(\alpha, \beta) \mid \alpha = \beta, \text{ for } \alpha, \beta \in \{0, 1\}^s\}$ are defined by a shift-invariant predicate that just tests, for all $i = 1, \ldots, s$, that the ith characters of α and β are the same.) By a simple extension of the proof of Lemma 3.2, it follows that:

LEMMA 3.3. *Given a tube T consisting of a subset of $E(A^s \times A^s)$, for any relation R defined by a shift-invariant predicate, a test tube consisting of $E(R) \cap T$ can be constructed in $O(|A|^{c+1}s)$ Separation and Merge operations.*

Note that $|A|$, c are constants in all applications of this lemma within this paper.

4. Computations Using the PAM Model.

Here we give simulations of sequential and parallel computations by using the PAM models. (See Figure 1.) These simulations are surprisingly succinct. We need some further description to show that we can implement

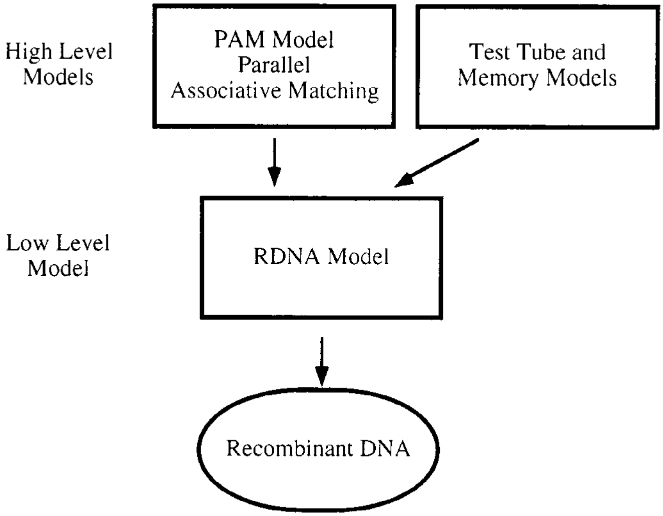

High Level Models
PAM Model Parallel Associative Matching
Test Tube and Memory Models
Low Level Model
RDNA Model
Recombinant DNA

Fig. 1. Molecular model simulation results.

the PA-Match operation by recombinant DNA techniques. This turns out to be a nontrivial task, and is the main technical contribution of the remaining portion of our paper. In Section 5 we show that Theorem 4.1 also holds for our RDNA model, and in Section 6 we show the simulation, at least up to a certain size, can be done in recombinant DNA by conventional biotechnology operations.

4.1. *Simulation of Lengthy Sequential Computations*

THEOREM 4.1. *A nondeterministic Turing Machine (TM) computation with space bound $s \geq$ the length of the input string, and time $2^{O(s)}$ can be executed in our PAM model using $O(s)$ PA-Match steps and $O(s \log s)$ other PAM steps, employing strings of length $O(s)$.*

PROOF. Fix a sequential nondeterministic TM (see [HU] for the formal definition of a nondeterministic TM and relevant computational complexity theory) which we wish to simulate. Let $\Im$ be the set of instantaneous descriptions (also known as configurations) of the TM. Let Q, Γ be the constant size state set and tape alphabet of the simulated machine. In the following we assume without loss of generality that the TM uses only one read/write tape. (See Figure 2.) We represent each configuration $I = I_1 \cdots I_s$ as the concatenation of characters $I_1, \ldots, I_s$ over an alphabet A of size $a = (|Q| + 1)|\Gamma| = O(1)$ where for each $i = 1, \ldots, s$ the character I_i encodes (1) the contents of the memory cell in location i, and (2) the state of the finite control if the head is scanning the ith cell, and otherwise a distinguished symbol not in Q indicating the head is not scanning that cell. Each configuration I will thus have length s. (See Figure 3.)

Let $NEXT \subseteq \Im \times \Im$ be the next move relation of M. We assume that once in the final configuration I_f, the TM stays in the final configuration: so $(I_f, I_f) \in NEXT$ and if $(I_f, I) \in NEXT$, then $I = I_f$. For each $t \geq 1$, let $NEXT^{(t)} \subseteq \Im \times \Im$ be the t'th move relation of the TM; that is, $(I, I') \in NEXT^{(t)}$ iff I' is reachable from I by t execution steps of the TM, where $I, I' \in \Im$. By well known complexity theory results (see [HU]) the time bound of the TM can be assumed to be upper bounded by 2^{cs} for some constant $c \geq 1$.

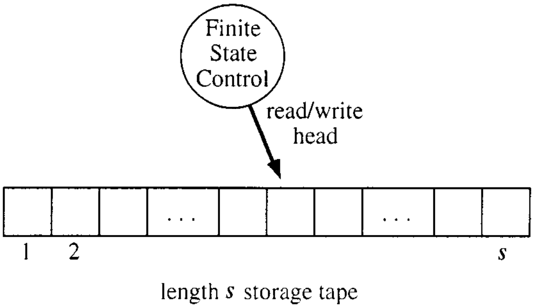


Fig. 2. A TM with space bound s.

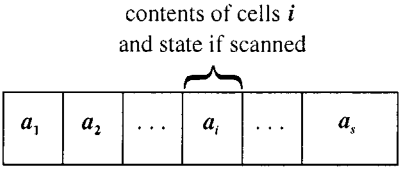


Fig. 3. Encoding of the configuration of a TM.

To encode any configuration pair $I, I' \in \Im$, we assume here, without loss of generality, a standard encoding for the PA-Match operation, $E(I, I') = \#_1 E(I) \#_0 \#'_0 \hat{E}(I') \#_2$.

(Note: recall that the definition of the standard encoding in Section 3.1 allowed also for any fixed permutations of the symbols of Σ. Repeated application of the operation $\bowtie$ will actually require us to use distinct standard encodings E, E', E'', as defined in the proof of Lemma 5.1, which simply differ by fixed permutations of the padding symbols of Σ. However, by Proposition 5.1, we can convert between these in $O(1)$ basic operations of the RDNA model. So for sake of clarity of notation in the PAM algorithm below, we ignore in this subsection these slight variants of standard encoding and just use the same symbol E for each.)

We assume that the initial and final configurations, I_0, I_f, have distinct symbols, so that in $O(s)$ Separation operations and a Detection operation, we can determine whether $E(I_0, I_f)$ is in a given test tube. Let $T^{(k)} = \{E(I, I') \mid (I, I') \in NEXT^{(2^k)}\}$ for each $k \geq 0$. Note that $T^{(k)} = T^{(k-1)} \bowtie T^{(k-1)}$. The simulation has three phases:

1. Initialization: construction of $T^{(0)}$.
2. For each $k = 1, \ldots, cs$, do PA-Match: $T^{(k)} = T^{(k-1)} \bowtie T^{(k-1)}$.
3. Final detection: test if there is some $E(I_0, I_f) \in T^{(cs)}$ where I_0, I_f are the initial and final configurations of the TM. □

4.2. *Initialization of the Space Bounded Simulation.* Here we provide the details of Step 1 in the TM simulation algorithm given in the proof of Theorem 4.1. It will be instructive to consider first a simpler construction:

PROPOSITION 4.1. *The tube $\{E(I) \mid I \in \Im\}$ can be constructed in $O(s \log s)$ steps in the PAM model, without use of the PA-Match operation.*

PROOF. We begin with test tube $E(A^s) = \{E(I) \mid I \in A^s\}$ defined by the trivial shift-invariant predicate that is true everywhere. Recall that we encode a symbol u within $E(I)$ as $\sigma_{u+(i-1)a}$. Given a tube T and integer i, where $1 \leq i \leq s$, let S_i (S'_i, respectively) be the set of characters $\sigma_{u+(i-1)a} \in \Sigma$, where $u \in A$ is a character in the ith position of the configuration that encodes that the head is (is not, respectively) scanning cell i. Note that both sets S_i, S'_i are of constant size $\leq a = |A|$. Let $S_i(T)$ ($S'_i(T)$, respectively) be the procedure that outputs the merge of all tubes derived from T by applying the Separation operation on any $\sigma_{u+(i-1)a}$ in S_i (S'_i, respectively).

Without loss of generality, we can again assume s is a power of 2. For each $k = 1, \ldots, \log s$, subdivide $\{1, \ldots, s\}$ into a set D_k of 2^k disjoint consecutive intervals, each of size $s/2^k$. There are a total of $s \log s$ such intervals. For each such interval $d \in D_k$, define $S'_d(T)$ to be derived from T by applying S'_i consecutively for each i, $1 \leq i \leq s$, not in d.

Note that D_1 consists of two intervals, $\{1, \ldots, s/2\}$ and $\{s/2 + 1, \ldots, s\}$, and it is easy to construct $S'_{\{1,\ldots,s/2\}}(T)$ in $s/2$ Separation operations by simply applying the Separation operation in sequence for every $\sigma_{u+(i-1)a}$ in S_i for $i = s/2 + 1, \ldots, s$. Similarly, we can construct $S'_{\{s/2+1,\ldots,s\}}(T)$ in $s/2$ Separation operations by simply applying the Separation operation in sequence for every $\sigma_{u+(i-1)a}$ in S_i for $i = 1, \ldots, s/2$. Thus in a total of s Separation operations we have constructed both $S'_{\{1,\ldots,s/2\}}(T)$ and $S'_{\{s/2+1,\ldots,s\}}(T)$.

We can generalize the above construction for $k = 1$ as follows: For each $k = 2, \ldots, \log s$, and for each $d \in D_k$, let d' be the unique interval in D_k distinct from d which is the same interval of D_{k-1} as d, and let d'' be the interval of D_{k-1} that does not contain d. (For example, for $k = 2$, the set D_k consists of four intervals: $\{1, \ldots, s/4\}$, $\{s/4 + 1, \ldots, s/2\}$, $\{s/2 + 1, \ldots, s3/4\}$, $\{s3/4 + 1, \ldots, s\}$. Each $d \in D_k$ is of the form $d = \{js/4 + 1, \ldots, (j + 1)s/4\}$ for some $j \in \{0, \ldots, 3\}$. Then $d' = \{(j + 1)s/4 + 1, \ldots, (j + 2)s/4\}$ if j is even and otherwise $d' = \{(j - 1)s/4 + 1, \ldots, js/4\}$. Also, $d'' = \{s/2 + 1, \ldots, s\}$ if the interval d is contained in $\{1, \ldots, s/2\}$, and else $d'' = \{1, \ldots, s/2\}$.)

Then we construct each $S'_d(T)$ by applying the Separation operation on each $\sigma_{u+(i-1)a}$ in sequence for all the $s/2^k$ elements $i \in d'$, starting from $S'_{d''}(T)$. Hence, by this divide and conquer method, we can construct all the $S'_d(T)$ in a total of s Separation operations for each k, for an overall total of $O(s \log s)$ Separation operations for for all $k = 1, \ldots, \log s$.

Note that the test tube $S'_{\{i\}}(T)$ is equivalent to applying S'_j consecutively for each j, $1 \leq j \leq s$, where $j \neq i$. Thus $HEAD_i = S_i(S'_{\{i\}}(T))$ encodes configurations where the head is scanning position i. The tube $\{E(I) \mid I \in \Im\}$ is the merge of the tubes $HEAD_1(T), \ldots, HEAD_s(T)$. □

A construction nearly identical to the proof of Proposition 4.1 gives:

PROPOSITION 4.2. *The tube $\{E(I, I') \mid I, I' \in \Im\}$ can be constructed in $O(s \log s)$ steps in the PAM model, without use of the PA-Match operation.*

PROOF. We begin with test tube $T_0 = E(A^s \times A^s) = \{E(I, I') \mid I, I' \in A^s\}$ defined by the trivial shift-invariant predicate that is true everywhere. Recall that we encode a symbol v within $\hat{E}(I')$ as $\sigma_{v+(i-1)a+a}$. We apply the procedure of Proposition 4.1 and then apply a modification of the procedure of Proposition 4.1 where each separation on any character of form $\sigma_{v+(i-1)a}$, where $1 \leq i \leq s$, is replaced with separation on the character $\sigma_{v+(i-1)a+a}$. □

Finally, to implement the state-transition function of the TM, we observe:

PROPOSITION 4.3. *There is relation $R \subseteq A^s \times A^s$ defined by a shift-invariant predicate such that*

$$\{E(I, I') \mid (I, I') \in NEXT\} = \{E(I, I') \mid I, I' \in \Im\} \cap E(R).$$

PROOF. For each state transition τ from a given state $q \in Q$, where the read/write head is at position i, the transition depends only on the value of the cell at position i, and on the next move, the read/write head can only move one cell to the right or left. Thus given standard encoding $E(I, I')$ where $I, I' \in \Im$, to choose those $E(I, I')$ such that $(I, I') \in NEXT$, we need only test a predicate depending on the values at a given location within $E(I, I')$, and predetermined locations within $E(I, I')$ of shifted distance $sa - 1, sa, sa + 1$. □

By Lemma 3.3, we can construct in $O(s)$ steps of the PAM model a tube consisting of

$$\{E(I, I') \mid (I, I') \in NEXT\} = \{E(I, I') \mid I, I' \in \Im\} \cap E(R).$$

Hence we have shown:

LEMMA 4.1. *The initialization, where we construct test tube $T^{(0)}$, can be done in $O(s \log s)$ steps in the PAM model, without use of the PA-Match operation.*

Note that each of the main simulation steps of our algorithm are simply the execution of the PA-Match operation, which is a primitive of our PAM model, taking by definition step 1. It is interesting to note that the molecular simulation algorithm we have given is similar to the parallel simulation due to Fortune and Wyllie [FW] of a space bounded TM by a PRAM; where we have replaced the pointer jumping operation by our PA-Match operation. This seems to indicate that the techniques used by computer scientists in the design of parallel algorithms may be quite useful in the new area of biomolecular computation. It should be noted however that pointer jumping of the relation *NEXT* entails test tubes of very large size. In particular, the TM simulation of Theorem 4.1 requires test tubes of size a^{2s}, where $a = (|Q| + 1)|\Gamma|$. This implies our result is of theoretical interest only.

Note that the ligation operation of the Adleman and Lipton models can be easily used to form in $O(s)$ steps the input test tube of step 1. Thus with the ligation operation, the time bounds of Theorem 4.1 decrease to $O(s)$ steps.

4.3. *Simulation of Parallel RAMs.* A *parallel RAM* (*PRAM*) is a generalization of the usual sequential Random Access Machine (RAM) to allow P sequential RAM processors to execute in parallel in lock step; for further details see [FW], [J], and [Re1]. (See Figure 4.) A CREW PRAM allows concurrent reading of any location by multiple processors, but only allows processors to do concurrent writing of distinct locations at the same time. We assume there are M shared memory locations, indexed by the integers $\{1, \dots, M\}$ where all the processors can read and write, and each processor has a constant

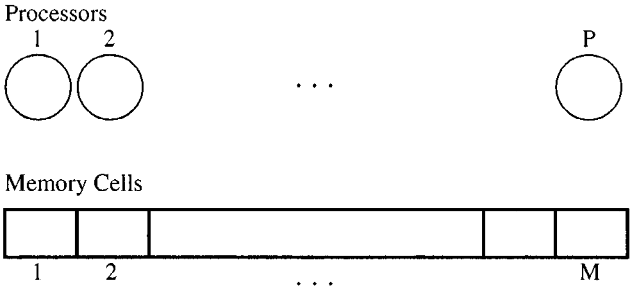


Fig. 4. A parallel RAM.

number of *registers*. Each memory location and register can hold a $b = O(\log M)$ bit integer. We assume the processors can execute unit cost arithmetic operations on b bit binary numbers. For simplicity, each PRAM processor is assumed to have a finite state control with a constant number of states, where the state of a processor does not include the value of its registers. Depending on its state, each PRAM processor can in one step make a state transition (the state transition can depend on the value of the registers) and apply the following operations:

1. apply an arithmetic operation on a pair of its registers and put the result in another of its registers,
2. load the value stored at a specified shared memory location into a register, and
3. write a value stored at a register into a specified memory location.

Fortune and Wyllie [FW] have given a simulation of a PRAM with time bound D by a $O(D^2)$ space bounded TM. Hence by combining this result with our simulation of space bounded TMs by the PAM model stated in Theorem 4.1, we have that a PRAM with time bound D can be simulated in $O(D^2)$ steps in the PAM model. However, a direct simulation gives much better bounds. A PRAM, where each location of memory is written on each time step by a unique processor, is called a *dynamic memory PRAM* (such a PRAM suffices for many applications, including the proof of Corollary 4.1).

THEOREM 4.2. *A dynamic memory CREW PRAM with time bound D, M memory cells, and processor bound $P \geq M$ can be simulated using $O(D + s)$ PA-Match steps and $O(s \log s)$ other PAM steps using strings of length $O(s)$, where $s = O(\log(PM))$.*

PROOF. Without loss of generality, with a constant factor increase in time bounds of the PRAM, we simplify the sequence of the PRAM operations by assuming that each processor has a single distinguished *cache* register and on each state transition, each processor executes the following round of operations in lock-step: (1′) writes from the cache into a memory location, (2′) then reads another distinct memory location into the cache, (3′) and applies an arithmetic operation on the registers, (4′) then changes state as specified by the state transition function (dependent on the value of one of the registers). Since we are simulating an exclusive-write PRAM, on each round, each location of memory can be assumed to be written by a unique processor. We let $\Im$

be the set of all possible instantaneous descriptions (i.e., configurations) of any processor; each configuration $I \in \Im$ will be an $s = O(\log(PM))$ length binary string encoding

1. the processor (identification) number in $\{1, \ldots, P\}$,
2. the current state of the processor, and
3. the contents of its registers.

We can assume the state transition function is the same for all the PRAMs, and that they all start at the same state, but the initial value of their cache register is their processor number. The *input* and *output* of the PRAM will be given by the memory locations at times 0 and D, respectively. Let V be the set of all $2^b \leq M^{O(1)}$ possible values of any register or memory cell and let *LOC* be the set of all M possible memory locations; we assume each value $v \in V$ and each address $a \in LOC$ are also represented as s length binary strings. Let $s' \leq cs$ for a constant integer $c \geq 1$. To encode any s' length binary strings $\alpha, \beta \in \{0, 1\}^{s'}$, we use here standard encodings defined in Section 3.1: $E(\alpha)$ and

$$\psi = E(\alpha, \beta) = \#_1 E(\alpha)\#_0\#'_0\hat{E}(\beta)\#_2$$

over the alphabet Σ. We now require a slightly larger encoding alphabet: $\Sigma' = \Sigma \cup \{\hat{\#}_0, \hat{\#}'_0, \hat{\#}_1, \hat{\#}_2\}$, where $\hat{\#}_0, \hat{\#}'_0, \hat{\#}_1, \hat{\#}_2$ are distinct symbols not in Σ. Let $\hat{E}(\psi) = \hat{\#}_1\hat{E}(\alpha)$ $\hat{\#}_0\hat{\#}'_0 E(\beta)\hat{\#}_2$. Given also any $\alpha' \in A^{s'}$, we define a secondary encoding (to be applied just once recursively on the second argument),

$$E_1(\alpha', \psi) = \#'_1 E(\alpha')\#_3\#'_3\hat{E}(\psi)\#'_2.$$

Note that this encoding simply differs from the form of the standard encoding by fixed permutations of the distinguished padding symbols $\#_i, \#'_i, i = 0, \ldots, 3$, of Σ, and it uses the distinct padding symbols $\#'_1, \#'_2, \#_3, \#'_3$ instead of the padding symbols $\#_0, \#'_0, \#_1, \#_2$ used by the standard encoding. Recall that for strings $\alpha, \beta, \beta', \gamma \in A^{s'}$ the PA-Match operation gives $E(\alpha, \beta) \bowtie E(\beta', \gamma) = E(\alpha, \gamma)$ and $E(\beta) \bowtie E(\beta', \gamma) = E(\gamma, \beta') \bowtie E(\beta) = E(\gamma)$ when $\beta = \beta'$, and the operation $\bowtie$ yields no value in the case $\beta \neq \beta'$. We also naturally generalize the PA-Match operation, so that, for strings $\alpha, \beta, \beta' \in A^{s'}$ and $\psi = E(\alpha, \beta)$, the generalized PA-Match operation gives:

- $E(\beta) \bowtie E_1(\beta', E(\psi)) = E(\psi)$ when $\beta = \beta'$, and the operation $\bowtie$ yields no value in the case $\beta \neq \beta'$.
- Also, $E_1(\alpha, E(\psi)) \bowtie E(\psi') = E(\alpha)$ when $\psi = \psi'$, and the operation $\bowtie$ yields no value in the case $\psi \neq \psi'$.

From the state transition function of the PRAMs we can define the following:

- Let $LOAD = \{E_1(I', E(I, v)) \mid$ configuration $I' \in \Im$ is derived from configuration $I \in \Im$ by having the processor specified by I load value $v \in V$ from memory into the cache register$\}$.
- Let $WRITE = \{E_1(I, E(a, v)) \mid$ at configuration $I \in \Im$, the processor specified by I writes value $v \in V$ into memory location $a \in LOC\}$.
- Let $READ = \{E_1(I, E(I, a)) \mid$ at configuration $I \in \Im$, the processor specified by I reads the value at memory location $a \in LOC\}$.

- For each time step t, $0 \leq t \leq D$,
  - let the memory cells be encoded as $T_{\mathrm{m}}^{(t)} = \{E(a, v) \mid \text{at time } t \text{ location } a \in \mathit{LOC}$ contains value $v \in V\}$, and
  - let the processor configurations be encoded as $T_{\mathrm{p}}^{(t)} = \{E(I) \mid \text{at time } t$ the processor specified by I is in configuration $I \in \Im\}$.

Note that the arithmetic operations, executed by any processor of the PRAM, are assumed to be on $b = O(\log M)$ bit numbers, and so can be certainly computed in space $O(\log^{O(1)} b) \leq O(s) = O(\log(PM))$ by a deterministic TM. These arithmetic operations are encoded into the tube *WRITE* . Thus tubes *LOAD, WRITE, READ*, $T_{\mathrm{m}}^{(0)}$, $T_{\mathrm{p}}^{(0)}$ can each easily be constructed in space $O(s) = O(\log(PM))$ by a deterministic TM. By our simulation of space bounded TMs by the PAM model stated in Theorem 4.1, it follows that:

LEMMA 4.2. *The tubes LOAD, WRITE, READ,* $T_{\mathrm{m}}^{(0)}$, $T_{\mathrm{p}}^{(0)}$ *can all be constructed in our PAM model using* $O(s)$ *PA-Match steps and* $O(s \log s)$ *other PAM steps using strings of length* $O(s)$.

Since we have assumed that each location of memory is written on each time step by a unique processor, observe that at time step $t + 1 \leq D$, the memory cell encoding $T_{\mathrm{m}}^{(t+1)}$ is given by the tube containing all $E(a, v)$ such that the processor specified by configuration $I \in \Im$ writes value $v \in V$ into memory location $a \in \mathit{LOC}$ at time t. By our definition of *WRITE*, it immediately follows that the memory cell encoding can be recursively defined as

$$T_{\mathrm{m}}^{(t+1)} = T_{\mathrm{p}}^{(t)} \bowtie \mathit{WRITE}\,.$$

Also observe that by our definition of *READ*, it also immediately follows that $T_{\mathrm{p}}^{(t)} \bowtie$ *READ* is a tube containing those $E(I, a)$ such that the processor specified by configuration $I \in \Im$ reads the value at memory location $a \in \mathit{LOC}$ at time t. Thus

$$(T_{\mathrm{p}}^{(t)} \bowtie \mathit{READ}) \bowtie T_{\mathrm{m}}^{(t)}$$

is a tube containing those $E(I, v)$ such that the processor specified by configuration $I \in \Im$ reads the value v from a memory location at time t. Hence by our definition of *LOAD* (and since we have assumed the order described above for the READ, LOAD, and state transition operations executed by each processor on each step), we conclude that the processor configurations can be recursively defined as

$$T_{\mathrm{p}}^{(t+1)} = \mathit{LOAD} \bowtie ((T_{\mathrm{p}}^{(t)} \bowtie \mathit{READ}) \bowtie T_{\mathrm{m}}^{(t)}).$$

Since these recursive definitions use only the PA-Match operation, we have:

LEMMA 4.3. *For each time step* t, $0 \leq t \leq D$, *given the tubes LOAD, WRITE, READ,* $T_{\mathrm{m}}^{(t)}$, $T_{\mathrm{p}}^{(t)}$, *then the tubes* $T_{\mathrm{m}}^{(t+1)}$, $T_{\mathrm{p}}^{(t+1)}$, *can all be constructed in our PAM model by* $O(1)$ *PA-Match steps.*

Theorem 4.2 follows immediately from Lemmas 4.2 and 4.3. □

The *fan-out* of a Boolean circuit is the number of gates that directly use the output value of a given gate. Theorem 4.2 combined with the nondeterministic TM simulation of Theorem 4.1 together imply:

COROLLARY 4.1. *Determining the satisfiability of a Boolean circuit constructible in s space with n inputs, unbounded fan-out, and depth D, can be done in our PAM Model using $O(D + s)$ PA-Match steps and $O(s \log s)$ other PAM steps using strings of length $O(s)$.*

Note that the initial Step 0 of the proof of Theorem 4.2 can easily be done in $O(s)$ steps assuming the ligation operation of the Adleman and Lipton models is used to form the input test tubes. Thus with the ligation operation, the time bounds of Theorem 4.2 and Corollary 4.1 reduce to $O(D + s)$ PA-Match steps and $O(s)$ other PAM steps.

Using similar techniques, the tree contraction technique developed by Miller and Reif [MR] can also be efficiently executed in the PAM model.

5. A Model for Recombinant DNA. Our previously defined PAM model is quite expressive and powerful for describing biomolecular computations at a very high level. Here we define a low level, but still abstract, model of biomolecular computation that we call the *Recombinant DNA* (*RDNA*) *model*. The RDNA model allows for operations which are abstractions of very well understood recombinant DNA operations (these include operations in which covalent bonds are broken and made, e.g., cleavage and ligation) as well as basic operations of molecular biology (separation, detect, denature, etc.).

The RDNA model provides a representation, which we call the *complex*, for the relevant structural properties of DNA or RNA. We use complexes to give rigorous definitions of the recombinant DNA operations. Our RDNA model is related to the PAM model, where we allow the PAM operations Merge, Copy, Detect, and Separation, but we disallow the PA-Match operation, and allow the following additional operations (which are defined in detail in Section 5.2):

1. *Cleavage* or cutting of strands based on short patterns,
2. *Annealing* of single-stranded linear structures into complex structures,
3. *Ligation* to form covalent bonds, and
4. *Denature* of complex structures into single-stranded linear structures.

The RDNA model is thus a bridge between the PAM model and the actual recombinant DNA operations done in a laboratory. Although the RDNA model is abstract (for example, the strings of the model are over an alphabet that grows as $2n$ but DNA has an alphabet of size 4, and we do not directly refer to the underlying organic chemistry), nevertheless our model is sufficiently low level to ensure that only very well understood recombinant DNA operations are allowed as basic operations. The actual recombinant DNA operations done in laboratories have been changing over time and have certain limitations (volume of test tube, etc.) that can not be well described by an abstract model such as RDNA that has a precise mathematical definition, as required for our simulation proofs.

We will prove that the PAM model can be simulated by the restricted RDNA model with only an $O(s)$ factor slowdown, which implies our low-level restricted RDNA model gets within a linear factor of the same bounds for simulations of conventional sequential and parallel computations as our high-level PAM model. In the following section we discuss in some detail how each operation of the RDNA model can be implemented by recombinant DNA operations in a laboratory, at least up to certain size parameters.

5.1. *Complexes*. As in the previous abstract model PAM of biomolecular computation, we define an alphabet of distinct symbols $\Sigma = \{\sigma_0, \sigma_1, \dots, \sigma_{n-1}, \bar{\sigma}_0, \bar{\sigma}_1, \dots, \bar{\sigma}_{n-1}\}$. A graph with directed edges is called a *digraph*. A string α over Σ has a naturally defined labeled digraph $G(\alpha)$ consisting of a simple (nonrepeating) directed path of length $|\alpha|$ whose consecutive directed edges are labeled with the consecutive characters of α, from first to last. (Note that in the special case of DNA, the direction of a path models the $3'$–$5'$ or $5'$–$3'$ orientation of the DNA chain; see Section 6.1.)

Fix a finite set S of k finite strings over Σ, which may include linear strings and also string loops. The digraph $G(S)$ is defined to be the union of the k disjoint labeled digraphs $G(\alpha)$ for each $\alpha \in S$; hence $G(S)$ consists of k disjoint directed paths labeled with the strings of S.

We define here a *labeled pairing* of the edges of $G(S)$ to be a set μ of unordered pairs of distinct directed edges of $G(S)$ such that (1) no directed edge appears more than once in μ, and (2) each of the pairs of edges in μ have complementary character labels in reversed order. To model the Annealing process of DNA, we define a *complex over S* to be the pair (S, μ), where μ is a labeled pairing of $G(S)$.

Note that each pair of directed edges $\{(i, j), (j', i')\} \in \mu$ point in the opposite direction, as occurs in DNA, when segments of two DNA chains with opposite $3'$–$5'$ and $5'$–$3'$ orientations are matched by Watson–Crick complementation. The complex (S, μ) has a naturally defined *annealed digraph* $G(S, \mu)$ derived from digraph $G(S)$ by merging together the vertices i, i' and merging together the vertices j, j' for all labeled pairs $\{(i, j), (j', i')\}$ in μ, so the resulting digraph has edges in both directions between the two merged nodes. (Note that three nodes may be merged into one, for example, j and j' would be merged with j'' if the pair $\{(j, k), (k', j'')\}$ is in μ, in addition to the pair $\{(i, j), (i', j')\}$.) The annealed digraph is useful in characterizing unlikely complexes, as mentioned below. However, for technical reasons, we later define the Denature and Ligation operations using the graph $G(S)$ and μ, rather than using directly the annealed digraph $G(S, \mu)$.

The complex (S, μ) is a *linear complex* if $\mu = \{\}$ so the digraph $G(S)$ is a set of simple directed paths. The complex (S, μ) is a *loop complex* if $\mu = \{\}$ and the digraph $G(S)$ is a loop (for example, if S consists of two strings so $G(S)$ has only two paths p_1, p_2 (where one is of length at least 2) and μ pairs both the first edges of p_1, p_2 as well as the last edges of p_1, p_2). (See Figure 5.)

Let COMPLEX(S) be the set of all possible complexes over S. Define the set of strings S to be *simple* if COMPLEX(S) has only the linear complex. Also define a single string α to be *simple* if the singleton set $\{\alpha\}$ is simple; hence it has no self-pairing in any possible complex containing only itself, and so it contains no pair of complementing symbols.

Note that COMPLEX(S) may contain structures that actual DNA may not allow, at least in low potential energy states. For example, DNA in a low potential energy state does not usually bend more than a few degrees per base pair (see [Si]), so DNA in a low

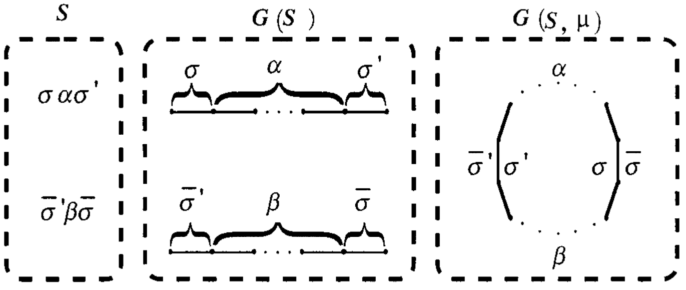


Fig. 5. Example of a set of strings $S = \{\sigma\alpha\sigma', \bar{\sigma}'\beta\bar{\sigma}\}$, and the digraphs $G(S)$, $G(S, \mu)$.

potential energy state usually has no loops of length less than a few dozen base pairs. We would like to define a notion of feasible complexes that (a) reflects physical constraints and (b) is also computationally efficient to verify. (For example, we might require that the annealed digraph $G(S, \mu)$ has an embedding to a one-dimensional manifold in $\Re^3$ (in 3-space) with a fixed upper bound on the curvature. Alternatively, we might bound the potential energy (due to curvature) of the allowed embedding. This latter constraint is the correct physical constraint that should be satisfied. However, testing either of these restrictions appears to be very time consuming.)

Therefore, we define the set of *feasible complexes* = FCOMPLEX(S) to be the set of complexes (S, μ) of COMPLEX(S) whose graph $G(S, \mu)$ has no cycles of length less than the constant c_1 (this is the same constant c_1 as defined in Section 3.1).

A feasible complex may be used to model both the information content and also the three-dimensional structure of single- or double-stranded DNA, or RNA, including hybridization and secondary structure. *We employ feasible complexes to allow us to model the effect of various recombinant DNA operations, and thus to provide rigorous definitions of the recombinant DNA operations.*

5.2. *Operations of the RDNA Model.* In the RDNA model we define a *tube* to be a multiset of feasible complexes over sets of strings over Σ. The RDNA model allows the following previously defined PAM operations:

1. *Merge*,
2. *Copy*, which duplicates the contents of a given tube consisting of only linear complexes,
3. *Detect*, and
4. *Separation*, where (a) to decrease the likelihood of Separation errors by mismatches, we again restrict the Separation operation to match with strings of length at most $c_0 = O(1)$; all our recombinant DNA simulations again require only $c_0 = 2$, and (b) we assume for simplicity that the Separation operation is only applied to tubes of linear or loop complexes (note that we might have defined the Separation to be a more general operation that allows us to identify a complex that has a strand sigma which happens to be annealed to its complement, but this may not always be feasible in practice);

and also allows the following further operations (note that we use complexes to give

$$\overbrace{A\ T\ G\ G\ A\ C}^{\alpha}\ \overbrace{A\ T\ C\ T}^{\sigma}\ \overbrace{T\ C\ G\ A\ T}^{\beta}$$

$$\Rightarrow \overbrace{A\ T\ G\ G\ A\ C}^{\alpha}\ \overbrace{A\ T\ C\ T}^{\sigma}$$

$$\overbrace{T\ C\ G\ A\ T}^{\beta}$$

Fig. 6. Cleavage of the DNA strand $\alpha\sigma\beta$ just after the sequence $\sigma = ATCT$, resulting in two disconnected strands $\alpha\sigma$ and β.

rigorous definitions of these RDNA operations):

5. *Selection.* Select complexes of a specified size (the size is the number of nodes of the complex).
6. *Cleavage.* Given a tube T, a symbol $\sigma \in \Sigma$, and *nick* $\in$ {before, after}, produce a new tube T' such that, for each complex (S, μ) in T, we substitute a new complex (S', μ') derived from (S, μ) as follows: at each directed edge e of $G(S, \mu)$ labeled σ we cleave the vertex where e is located, either just before or just after e, as specified by *nick*. (See Figure 6.) This Cleavage operation may be specified to be either double stranded or single stranded; in the former case it is required that there also be a reverse edge between the same vertices as e in the opposite direction, in the latter case it is required that there be no such reverse edge (if unspecified, we allow either case). We assume that the Cleavage operation be applied to a (small) constant subset of the symbols of the alphabet Σ, which are called *cleavage symbols*. This require a distinct restriction enzyme for each of the small number of cleavage symbols.
7. *Annealing.* Given a tube T, where S is the set of all strings appearing in complexes in T, let FCOMPLEX(S) be the set of all possible feasible complexes over S. We construct a new tube T' by taking from FCOMPLEX(S) all complexes that use matching between complementary pairs of strings of length $\leq c_0$, and also indeterministically choosing any further elements of FCOMPLEX(S); that is, we allow for the possibility, but not the surety, of any further subset of elements of FCOMPLEX(S) to be chosen. Thus the Annealing operation allows for the surety matching of Watson–Crick complementary subsequences of length $\leq c_0$, which is the only case that our algorithms will use (we can extend this definition to allow also the possibility of longer matches, and then it would be up to the algorithm designer to show that an RDNA algorithm is correct no matter what possible longer matches are made).

   Note. Depending on the number of strings appearing in complexes that can anneal, it may take a long time (or be unlikely over a short time duration) for one DNA string to meet another before annealing. This can be remedied by increasing the volume, so with the resulting large redundancy at least every instance of pairs of strings appearing in complexes can attempt to anneal. This increase in volume makes our implementation of the PA-MATCH operation of theoretical consequence only, due to the large test tube volume required to ensure annealing of many pairs of strings).

8. *Ligation*. We modify the set S as follows: For each linear strings $s, s' \in S$, let end(s) be the last edge of of the path p defined by the subgraph $G(\{s\})$ of $G(S)$, and let begin(s') be the first edge of the path p' defined by the subgraph $G(\{s'\})$ of $G(S)$. To implement ligation, we replace s, s' with their concatenation $s \cdot s'$ in the case where there is a string $s'' \in S$ with consecutive edges e, e' where $\{e,\ \text{end}(s)\} \in \mu$ and $\{e',\ \text{begin}(s')\} \in \mu$. (Note that both pairs $\{e,\ \text{end}(s)\}$ and also $\{e',\ \text{begin}(s')\}$ point in the opposite direction, as occurs in DNA, when segments of two DNA chains with opposite 3′–5′ and 5′–3′ orientations are matched by Watson–Crick complementation, and the edges end(s), begin(s') abut. Thus the edges end(s), begin(s') are ligated. Also note we can easily extend this definition to allow for blunt end ligation, but that operation is not required for our results.)
9. *Denature*. Given a tube T, for each complex (S, μ) in T, we substitute the set of complexes $\{(\{\alpha\}, \{\}) \mid \alpha \in S\}$. Thus the Denature operation transforms a tube of complexes into a tube of linear and loop strings.

5.3. *RDNA Simulation of the PAM Model*

THEOREM 5.1. *The PA-Match operation of the PAM model on strings of length s can be simulated in our RDNA model, with a $O(s)$ factor slowdown over the total step bound of the PAM model. Each of the other operations of the PAM model can be executed in our RDNA model without slowdown.*

PROOF. All but one of the operations of our RDNA model are those of the PAM model. However, the PA-Match operation requires detailed justification, to allow for its implementation by recombinant DNA. (In particular, note that a construction that attempts to match in one step lengthy Watson–Crick complementing pairs of lengthy subsequences is not feasible, due to the difficulty of ensuring such lengthy matches in DNA.)

We show that the execution of our PA-Match algorithm in the RDNA model only uses separation operations and annealing operations on strings of length $\leq c_0 = 2$, and there are no possible matches of length longer than 2. Thus the correctness of our PA-Match algorithm only depends on the surety of matches of length $\leq c_0 = 2$, which is an assumption of our RDNA model. The correctness of our PA-Match algorithm does not depend on the surety of matches longer than c_0, and, moreover, also does not depend on an assumption that there are no matches longer than c_0. An incorrect PA-Match algorithm in the RDNA model might, perhaps due to the redundant use of matched symbols, depend on an assumption (which we do not make) that there are no matches longer than c_0.

LEMMA 5.1. *We can implement the PA-Match operation using $O(s)$ lab steps in the RDNA model.*

PROOF. In this section, for the characters within the complexes of a tube in the RDNA model, we use an alphabet of distinct symbols $\Sigma = \{\sigma_0, \sigma_1, \dots, \sigma_{n-1}, \bar{\sigma}_0, \bar{\sigma}_1, \dots, \bar{\sigma}_{n-1}\}$, where $n = 2sa + 8 + c_1$. Let A be a constant size $a = |A|$ alphabet for the strings used for PA-Match. For two tubes T, T' containing linear complexes (i.e., strings) encoding pairs by $E(-,-)$, we wish to construct, by $O(s)$ operations in the RDNA model, the tube $T'' = T \bowtie T' = \{E(\alpha,\gamma) \mid E(\alpha,\beta) \in T,\, E(\beta,\gamma) \in T'\}$.

To encode in T any string pair (α,β) we assume, without loss of generality, the standard encoding $E(\alpha,\beta) = \#_1 E(\alpha)\#_0\#'_0\hat{E}(\beta)\#_2$, where we have reserved the eight distinct padding symbols $\#_j = \sigma_{2sa+j}$ and $\#'_j = \sigma_{2sa+j+4}$ for $j = 0,\ldots,3$. Recall that in the definition of a standard encoding in Section 3.1, we allowed a standard encoding to use any fixed permutation of these padding symbols. To encode in T' any string pair (α,β) we use a (slightly modified) standard encoding $E'(\alpha,\beta) = \#'_1 E(\alpha)\#_0\#'_0\hat{E}(\beta)\#'_2$. Also, to encode in T'' any string pair (α,β) we use a (slightly modified) standard encoding $E''(\alpha,\beta) = \#_1 E(\alpha)\#_0\#'_0\hat{E}(\beta)\#'_2$. These (slightly modified) standard encodings will not make any significant change in the encodings, but for technical reasons they will be useful in our cutting and splicing operations to implement PA-Match in the RDNA model.

The distinguished padding symbols can be easily permuted. For example, suppose we are given as input a set tube with contents in standard encoding E (with single strands). Then to convert the last symbol of each encoded string from $\#_2$ to $\#'{}_2$, we apply the following six recombinant DNA operations:

1. Cleavage just before $\#_2$, followed by
2. Merging the result with a fixed test tube containing length 2 strings of the form $\{\bar{\sigma}_{u+(2s-1)a}\bar{\#}'_2 \mid u \in A\} \cup \{\#'{}_2\}$,
3. Annealing and Ligation (thus forming segments of double strands),
4. Separation with match symbol $\#'{}_2$,
5. Selection of size $2s+4$, followed by
6. Denature (into single-stranded sequences).

It is easy to verify that in the resulting test tube the distinguished padding $\#_2$ has been edited to be $\#'{}_2$, as required in this example. The other operations required to convert between standard encodings E, E', E'' are similar, so we have:

PROPOSITION 5.1. *We can convert between standard encodings E, E', E'' in $O(1)$ basic operations of the RDNA model.*

Taking into account these (slightly modified) standard encodings, for each $\alpha,\beta,\beta',\gamma \in A^s$, we require the PA-Match operation $\bowtie$ to give $E(\alpha,\beta) \bowtie E'(\beta',\gamma) = E''(\alpha,\gamma)$ if $\beta = \beta'$, and otherwise, if $\beta \neq \beta'$, the operation $\bowtie$ yields no value. Hence, using these standard encodings, we have

$$(\#_1 E(\alpha)\#_0\#'_0\hat{E}(\beta)\#_2) \bowtie (\#'_1 E(\beta')\#_0\#'_0\hat{E}(\gamma)\#'_2) = \#_1 E(\alpha)\#_0\#'_0\hat{E}(\gamma)\#'_2$$

if $\beta = \beta'$, and otherwise no value. Let $\lambda = \sigma_{2sa+8}\sigma_{2sa+9}\cdots\sigma_{2sa+7+c_1}$ be the string listing the last c_1 distinct symbols of Σ. For any linear string α, let LOOP(α) denote a string loop derived from α by connecting the end to the start. Recall that we have assumed feasible complexes to have no cycles of length less than a constant c_1 (for example, 40–100 base pairs will suffice to allow for the easy (i.e., with small force) bending of a DNA chain into a loop; see [Si]).

INPUT: Tubes T and T'.

INITIALIZATION: We assume we have prepared fixed test tubes: $T_0[0] = \{\bar{\#}_0\bar{\#}'_0$, $T_0[1] = \{\bar{\#}_2\bar{\#}'_1\}$, $T_0[2] = \{\bar{\#}'_2\bar{\lambda}\bar{\#}_1\}$, where the notation $\{a\}$ indicates a multiset containing multiples of the given string a. (Note: To ensure the complementation required for our Annealing and Ligation steps, complexes derived from prepared tubes $T_0[0]$, $T_0[1]$, $T_0[2]$ will be assumed to have opposite orientation to the complexes of the input tubes T, T', e.g., in the case of DNA, the tubes $T_0[0]$, $T_0[1]$, $T_0[2]$ have DNA strands defined in 5′–3′ orientation from the strings given above, whereas tubes T, T' have DNA strands defined in 3′–5′ orientation from their given strings.)

STEP 1 (see Figure 7(b)). Let

$$T_1 = \{E(\alpha, \beta)E'(\beta', \gamma) \mid E(\alpha, \beta) \in T, E'(\beta', \gamma) \in T'\}$$

be the test tube constructed by

1.1. Merging T, T', and $T_0[1]$, and then
1.2. applying the Annealing and Ligation operations (thus forming segments of double strands), followed by
1.3. the Denature operation (resulting in single-stranded sequences),
1.4. applying the Separation operation and yielding a tube with complexes containing neither the character $\bar{\#}_2$ nor $\bar{\#}'_1$, and, finally,
1.5. Selection of size $2(2s + 4) = 4s + 8$.

COMMENT. The resulting test tube will contain simple complexes with single-stranded sequences of the form

$$E(\alpha, \beta)E'(\beta', \gamma) = \#_1 E(\alpha)\#_0\#'_0\hat{E}(\beta)\#_2\#'_1 E(\beta')\#_0\#'_0\hat{E}(\gamma)\#'_2$$
$$\text{for each} \quad E(\alpha, \beta) \in T, \quad E'(\beta', \gamma) \in T'.$$

This is because after Annealing and Ligation, we get complexes from elements of T, T', $T_0[1]$, and joined to form double-stranded segments at the complementing pairs $\#_2$, $\bar{\#}_2$ and $\#'{}_1$, $\bar{\#}'_1$. After Denature, the resulting tube contains single-stranded sequences $E(\alpha, \beta)E'(\beta', \gamma)$ for each $E(\alpha, \beta) \in T$, and $E'(\beta', \gamma) \in T'$ and the same resulting tube also contains $\bar{\#}_2\bar{\#}'_1$. The Separation operation eliminates $\bar{\#}_2\bar{\#}'_1$ from the tube.

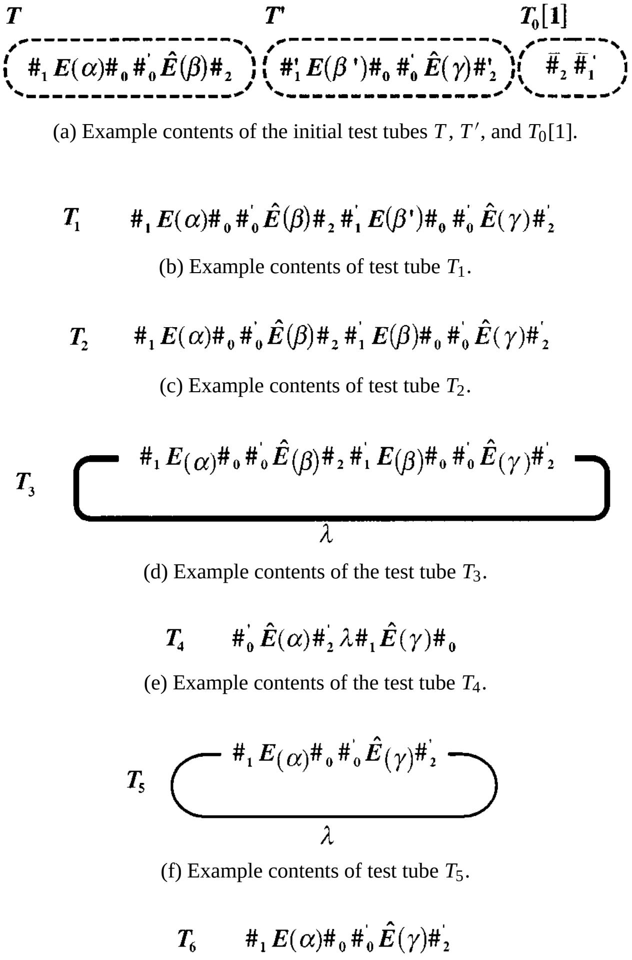
STEP 2 (see Figure 7(c)). Here we construct in $O(s)$ steps, by applying the proof of Lemma 3.3, the tube

$$T_2 = \{E(\alpha, \beta)E'(\beta', \gamma) \mid E(\alpha, \beta) \in T, E'(\beta', \gamma) \in T', \beta = \beta'\}.$$

COMMENT. We wish to Separate out all

$$E(\alpha, \beta)E'(\beta', \gamma) \in T_2$$

with $\beta = \beta'$. Note that we have restricted the Separation operation in our models to match with a pattern string over Σ of only constant length, and so at least s such Separation operations are required to do the required associative matching with the strings of length s. To do this, we observe:



(a) Example contents of the initial test tubes T, T', and $T_0[1]$.

(b) Example contents of test tube T_1.

(c) Example contents of test tube T_2.

(d) Example contents of the test tube T_3.

(e) Example contents of the test tube T_4.

(f) Example contents of test tube T_5.

(g) Example contents of the final test tube $T_6 = T \circ T'$.

Fig. 7. Example contents of the test tubes used to implement the PA-Match operation; all are single stranded strings or single stranded simple loops.

PROPOSITION 5.2. *There is a relation $R' \subseteq A^s \times A^s$ defined by a shift-invariant predicate such that $T_2 = T_1 \cap E(R')$.*

(In particular, the shift-invariant predicate just tests, for each $i = 1, \dots, s$, if the ith characters of β and β' are the same in the encoding.) By Lemma 3.3, we can construct in $O(s)$ Separation and Merge operations the tube $T_1 \cap E(R')$.

STEP 3 (see Figure 7(d)). Let

$$T_3 = \{\text{LOOP}(E(\alpha, \beta)E'(\beta', \gamma)\lambda) \mid E(\alpha, \beta) \in T, E'(\beta, \gamma) \in T_2\}$$

be the test tube constructed by

3.1. Merging T_2 and $T_0[2]$, and then
3.2. applying the Annealing and Ligation operations (thus forming segments of double strands), followed by
3.3. the Denature operation (resulting in single-stranded sequences),
3.4. applying the Separation operation yielding a tube with complexes containing neither the character $\bar{\#}_1$ nor $\bar{\#}'_2$.

(we may optionally apply also Selection of size $2(2s + 4) + c_1 = 4s + 8 + c_1$, but this has no effect if there are no errors.)

COMMENT. The resulting test tube will contain simple complexes with single-stranded loops of the form

$$\text{LOOP}(E(\alpha, \beta)E'(\beta', \gamma)\lambda) = \text{LOOP}(\#_1 E(\alpha)\#_0\#'_0\hat{E}(\beta)\#_2\#'_1 E(\beta)\#_0\#'_0\hat{E}(\gamma)\#'_2\lambda)$$
$$\text{for each} \quad E(\alpha, \beta) \in T, \quad E'(\beta', \gamma) \in T'.$$

This is because after Annealing and Ligation, we get cyclic complexes from elements of T_2, $T_0[2]$, joined to form double-stranded segments at the complementing pairs $\#_1, \bar{\#}_1$ and $\#'{}_2, \bar{\#}'_2$. After Denature, the elements of $E(\alpha, \beta) \in T$, $E'(\beta, \gamma) \in T'$, and λ are composed of single-stranded cyclic complexes of the form $\text{LOOP}(E(\alpha, \beta)E'(\beta, \gamma)\lambda)$ as well as $\bar{\#}'_2\bar{\#}_1$. The Separation operation eliminates the linear complexes $\bar{\#}'_2\bar{\#}_1$.

Note: the Separation step in Step 3 decimates the population of the test tube, thus requiring in practice the use of Amplification.

STEP 4 (see Figure 7(e)). Let T_4 be the test tube constructed by

4.1. Cleavage in the T_3 tube just after $\#_0$,
4.2. applying the Separation operation to the resulting tube, to obtain a tube not containing the character $\#_2$.

COMMENT. The Cleavage will result in linear complexes with single-stranded sequences of the form $\#'_0\hat{E}(\gamma)\#'_2\lambda\#_1 E(\alpha)\#_0$ and also of the form $\#'_0\hat{E}(\beta)\#_2\#'_1 E(\beta)\#_0$ for each $E(\alpha, \beta) \in T$, $E'(\beta, \gamma) \in T'$. The Separation operation eliminates the linear complexes of form $\#'_0\hat{E}(\beta)\#_2\#'_1 E(\beta)\#_0$. The resulting test tube T_4 contains linear complexes of the form $\#'_0\hat{E}(\gamma)\#'_2\lambda\#_1 E(\alpha)\#_0$ for each $E(\alpha, \beta) \in T$, $E'(\beta, \gamma) \in T'$.

STEP 5 (see Figure 7(f)). Let T_5 be the test tube constructed by

5.1. Merging T_4 and $T_0[0]$, and then
5.2. applying the Annealing and Ligation operations (thus forming segments of double strands), followed by
5.3. the Denature operation (resulting in single strands),
5.4. applying the Separation operation yielding a tube with complexes containing neither the character $\bar{\#}_0$ nor $\bar{\#}'_0$, and, finally,
5.5. Selection of size $2s + 4 + c_1$.

COMMENT. The resulting test tube will contain simple complexes with single-strand loops of the form

$$\mathrm{LOOP}(\#_0'\hat{E}(\gamma)\#_2'\lambda\#_1 E(\alpha)\#_0) \qquad \text{for each} \quad E(\alpha,\beta) \in T, \quad E'(\beta',\gamma) \in T'.$$

This is because after Annealing and Ligation, we get cyclic complexes from elements of T_4, $T_0[0]$, joined to form double-stranded segments at the complementing pairs $\#_0, \bar{\#}_0$ and $\#_0', \bar{\#}_0'$. After Denature, the elements of $E(\alpha,\beta) \in T, E'(\beta,\gamma) \in T'$, and λ are composed as single-stranded cyclic complexes of the form claimed. The Separation operation eliminates the linear complexes $\bar{\#}_0\bar{\#}'_0$.

STEP 6 (see Figure 7(g)). Let T_6 be the test tube constructed by

6.1. Cleavage in test tube T_5 just after $\#_2'$ and just before $\#_1$, and
6.2. applying the Separation operation yielding a tube with complexes not containing the first character σ_{2sa+6} of λ.

COMMENT. The Cleavage will result in linear complexes of the form

$$E''(\alpha,\gamma) = \#_1 E(\alpha)\#_0\#_0'\hat{E}(\gamma)\#_2' \qquad \text{for each} \quad E(\alpha,\beta) \in T, \quad E'(\beta,\gamma) \in T',$$

and also of the form λ. The Separation operation eliminates the linear complexes of form λ. The resulting test tube T_6 will contain only linear complexes of the form

$$E''(\alpha,\gamma) = \#_1 E(\alpha)\#_0\#_0'\hat{E}(\gamma)\#_2' \qquad \text{for each} \quad E(\alpha,\beta) \in T, \quad E'(\beta,\gamma) \in T'.$$

OUTPUT: $T \bowtie T' = T_6$.

Observe that, due to our use of distinct padding symbols, we have defined our PA-Match algorithm so it only uses matches of length $\le c_0 = 2$, and there are no possible matches of length longer than 2. Thus the correctness of our PA-Match algorithm only depends on the surety of matches of length $\le c_0 = 2$, which is an assumption of our RDNA model. (Also, note that this implies that the correctness of our PA-Match algorithm does not depend on the surety of matches longer than c_0, and also does not depend on an assumption that there are no matches longer than c_0.)

A similar, but slightly simpler, construction can be used to implement the special cases of the PA-Match operation $T \bowtie T'$ where the elements of one of the input tubes T or T' are of the form $E(\beta)$ for $\beta \in A^s$, and the elements of other tube are of the form $E(\beta',\gamma)$ for $\beta',\gamma \in A^s$. □

6. Implementation by Recombinant DNA. We briefly review the usual recombinant DNA technology, and its application to our operations. We observe that the basic operations of the RDNA model can be done, at least up to a certain size, by recombinant DNA using conventional biotechnology engineering techniques. A number of the operations of our RDNA model are those of the Memory model of Adleman, and their lab implementation by recombinant DNA techniques are detailed in his papers [Ad1],

[Ad2]. In particular, he has shown that each of the operations Merge, Detect, Separation, and Amplify can be done in $O(1)$ lab steps by recombinant DNA operations which can be considered to be basic operations in our RDNA model.

6.1. *DNA Base Pair Sequences*: *Their Orientation and Complement*. We require some elementary definitions. DNA is a linear polymer composed of two complementary linear sequences of bases: adenine (A), cytosine (C), guanine (G), and thymine (T). *Watson–Crick complementary* pairing is denoted by bars: $\bar{A} = T, \bar{T} = A, \bar{G} = C, \bar{C} = G$. The Watson–Crick complementation describes the hydrogen bonding between distinct subsequences of base pairs of DNA molecules. A *DNA* (*base pair*) *string* is a string over the alphabet of *DNA bases* $\{A, T, G, C\}$. The DNA base sequence provides the information content of the DNA strand in base 4. The unit measure of length of a DNA strand is given in terms of number of base pairs. A DNA strand has two distinguished ends which give the DNA strand polarity:

1. the $3'$ end, with normally a hydroxyl group ($3'$ OH), and
2. the $5'$ end, with normally a phosphate group ($5'$ PO_4).

This polarity is represented in the RDNA model by directed edges and directed paths. The two DNA strings $\alpha, \bar{\alpha}$ are defined to be *Watson–Crick complementary* where $\bar{\alpha}$ denotes the Watson–Crick complementation of the characters of the string α. Two subsegments of DNA strands can be joined by hydrogen bonds if the DNA strands are put in opposite $3'$–$5'$ and $5'$–$3'$ orientation, the corresponding base sequence strings being Watson–Crick complementary. Due to Watson–Crick complementation, the DNA molecule may form complex structures which may be singly or doubly stranded, or may contain segments that are doubly stranded. The COMPLEX of our RDNA model is used to represent these possible DNA structures. In the RDNA model the direction of a path models the $3'$–$5'$ or $5'$–$3'$ orientation of the DNA chain. For example, if we associate a DNA string with the $3'$–$5'$ orientation of the corresponding DNA strand, then the $3'$ end is associated with the start of the string and the $5'$ end is associated with the end of the string.

Certain recombinant DNA techniques, such as Amplification, require the DNA strings to be nondegenerate in a well understood sense. (On the other hand, certain other recombinant DNA operations use the degeneracy of a DNA string.) A string of even length is a *palindrome* if it is of the form $\alpha\alpha^{\mathrm{R}}$ where α^{R} is the reverse of string α. A string has a *replication* of a string α if it has two occurrences of α. A DNA strand is generally considered *nondegenerate* if it is free of palindromes, replications, and pairs of Watson–Crick complementary subsequences (or their reverses) above some proscribed small length ρ and else *degenerate*. (See Figure 8.)

To implement our RDNA model in actual DNA, we represent each character of Σ by a DNA strand (also known as an *oligonucleotide*) of length $c_3 \log n$, for some constant $c_3 \geq 1$. If we use degenerate DNA strings to encode pairs of configurations, these may form complex structures, including self-loops, that are not allowed for certain recombinant DNA operations. Adleman [Ad1] and Lipton [L] have noted that we can use a random string over $\{A, T, G, C\}$ of length $2 \log n$ to represent each character of Σ, and the resulting DNA strings are nondegenerate with high likelihood $1 - n^{-c}$, for a constant $c \geq 1$. Also, it is easy to show that, with high likelihood, each pair of such

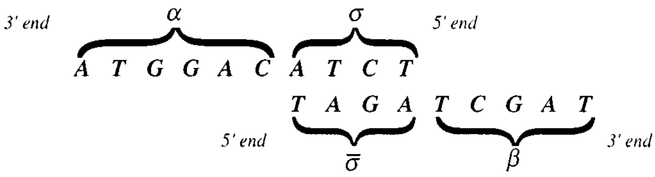


Fig. 8. Two single DNA strands $\alpha\sigma$ (oriented 3′–5′) and $\bar{\sigma}\beta$ (oriented 5′–3′) joined at Watson–Crick complementary sequences $\sigma = ATCT, \bar{\sigma} = TAGA$ to form a partial double strand of DNA where $\alpha = ATGGAC$, $\beta = TCGAT$.

random strings differs in $\Omega(\log n)$ of their bits. Hence a string over Σ of length k may be used to represent the information content of a DNA or RNA strand of $k \log(2n)$ base pairs and the *base pair length* of a string α over Σ is a multiplicative factor of $\log(2n)$ times the length of α. In the following, we let σ denote a short DNA string for length $c_0c_3 \log n \leq O(\log n)$ (such a string can be encoded as a string of length $\leq c_0$ in the alphabet Σ of the PAM or RDNA models.)

6.2. *Primitive Operations Used in Recombinant DNA.* There are a vast number of known recombinant DNA techniques; however, the techniques we employ are few, and are now routine. We have provided references to texts and a lab manual [SFM]; this and other lab manuals detail the elementary lab steps required for most recombinant DNA operations, and also have voluminous references to articles on these well understood techniques.

1. *Merge* can be done by simply combining the contents of the given test tubes and mixing. This can be facilitated by ultrasonic mixing devices.
2. *Copy*, which duplicates the contents of a given tube consisting of only linear complexes, can be done by Amplification via the polymerase chain reaction (PCR) (see p.79 of [WGWZ], Chapter 1 of [SFM], and also [Rob] and [Bar] for high yield PCR) resulting in the replication of given DNA strands. Amplification can also be of use in the practical implementation of our simulation results as an aid to decrease errors.
3. *Detect* is a standard operation molecular biology (e.g., [SFM]) which generally is preceded by Amplification.
4. *Separation*, by use of synthetic oligonucleotide probes (these probes may be attached to magnetic beads or attached to a structure; see Chapter 11 of [SFM]) containing a predetermined short subsequence. (If the matched subsequence is too long, we can get separation errors by mismatches.) The Separation operation can also be used in parallel, to allow for Separation of DNA strands containing as a subsequence any of a set of strings.
5. *Selection* of DNA strands of a given length of base pairs. This can be done by various methods (e.g., by use of gel electrophoresis; see Chapter 4 of [SFM]) which select by molecular weight. To ensure accuracy, the Selection must not be more than a few hundred base pairs.

6. *Cleavage* (see p. 64 of [WGWZ], Chapter 3 of [OP], Chapter 5.10 of [SFM], and [Ro]) and other operations using restriction endonucleases that nick (i.e., splice) a double strand of DNA at predetermined locations. These locations are determined by Watson–Crick complementary matching with a short pattern depending on the Cleavage. For example, if the string σ is not too long or degenerate, a DNA restriction endonuclease can be found that matches to the string σ, and cleaves each strand of the DNA just after or just before any instance of that string σ. The cleavage may be the same for both strands, and if so the cleavage is called *blunt ended*, and otherwise the cleavage is called a *cohesive overhang*. The cleavage can be done also on single strands either by (i) certain restriction endonucleases (however, the number of known restriction endonucleases for single strands is very limited, and they do so at a very low rate), or, more commonly, (ii) by creating a short double-strand segment within the single strand by annealing a strand complementing the restriction pattern, and then applying the required restriction endonucleases that nicks this double-strand segment as required. For example, if an appropriate restriction endonuclease is added to the contents of a test tube containing multiple DNA strings $\alpha\sigma\beta$, the result will be a test tube containing multiple DNA $\alpha\sigma$ and β. (See again Figure 6.)
7. *Annealing* of single-stranded DNA into complex structures including double-stranded DNA via cooling (see Chapter 11.8 of [SFM] and see [MH] and [OP]). For example, if the DNA string σ is not too long and does not appear in α, and if the contents of a test tube containing multiple single-stranded DNA with the string (in $3'$–$5'$ orientation) $\alpha\sigma$ is combined with the contents of another test tube containing multiple single-stranded DNA with the string (in $5'$–$3'$ orientation) $\bar{\sigma}\beta$, and cooled appropriately. (The Annealing allows for matching of Watson–Crick complementary subsequences. The schedule of cooling must be carefully controlled to avoid excessive mismatches.) Then the two DNA strands will be joined (in opposite $3'$–$5'$ and $5'$–$3'$ orientation) at $\sigma, \bar{\sigma}$. (See Figure 8.) If the strands are sufficiently long, such Annealing may also result in various types of DNA structures including loops; for example, if the strings σ, σ' are not too long, and do not appear in α, β, and if one of the strings α, β are sufficiently long (approximately 40–100 base pairs; see [Si]), and if the contents of a test tube containing multiple single-stranded DNA with the string (in $3'$–$5'$ orientation) $\sigma\alpha\sigma'$ is combined with the contents of another test tube containing multiple single-stranded DNA with the string (in $5'$–$3'$ orientation) $\bar{\sigma}\alpha\bar{\sigma}'$, and cooled appropriately, then the two strands will be joined at $\sigma, \bar{\sigma}'$ and also at $\sigma', \bar{\sigma}'$ thus forming a DNA loop. (See Figures 9 and 10.)

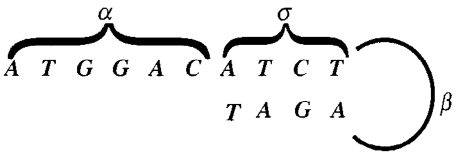


Fig. 9. A segment of DNA with a loop resulting from a degenerate single strand $\alpha\sigma\beta\bar{\sigma}^R$ with complementary substrings $\sigma, \bar{\sigma}$.

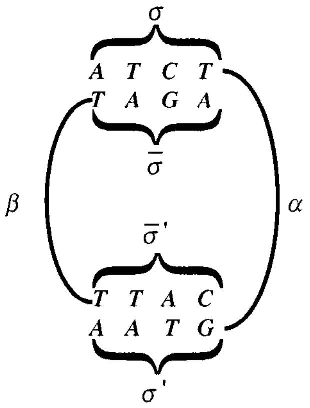


Fig. 10. Two single DNA strands $\sigma\alpha\sigma'$ (oriented 3′–5′) and $\bar{\sigma}'\beta\bar{\sigma}$ (oriented 5′–3′) joined at two pairs of Watson–Crick complementary sequences: $\sigma = ATCT, \bar{\sigma} = TAGA$, and also $\sigma' = GTAA, \bar{\sigma}' = CATT$, to form a cyclic strand of DNA.

8. *Ligation* uses a ligation enzyme to covalent bond annealed double-stranded DNA (see Chapter 11.8 of [SFM]).
9. *Denature* of double-stranded DNA into single-stranded DNA via heating.

Separation, Cleavage, Annealing, and Ligation operations based on complementary matching can be used to implement DNA string-editing, also known as *site-directed or localized mutagenesis* (see pp. 191–206 of [OP], pp. 192–193 of [WGWZ], and Chapter 5 of [SFM]), however, DNA string-editing using these recombinant DNA operations is only applicable to very short pattern strings.

7. Open Problems. Our simulation of the PA-Match operation in the RDNA model uses techniques similar (at least in certain technical details) to those used for small circular DNA found in bacteria known as plasmids.

1. Are there other recombinant DNA techniques used by bacteria or related living forms that can be of use in biomolecular computation? Our simple loop construction is not the only way of proving Lemma 5.1; other alternative constructions can be made using a simple loop plus dangling end segments (and can use a variant of our standard encoding with the pair of pattern strings reversed); however, all such constructions also appear to require at some stage some variant of our use of a loop or cyclic substructure and some variant of our standard encoding.
2. It is an interesting open question to determine if such techniques are inherent in any simulation of the PA-Match operation in the RDNA model. Other open problems include:
3. improvement (or matching lower bounds) of our simulation step bounds, especially in our RDNA model (Do we need a factor s slowdown?), and
4. development of further methods for ensuring the reliability of biomolecular computations. For example, can Separation errors (or other recombinant DNA operation errors) be reduced without a slowdown?

Acknowledgments. We wish to thank the following for their comments on the paper: Shenfeng Chen, Pavankumar Desikan, Ming Kao, Thom LaBean, Chris Lambert, Zhiyong Li, Peter Mills, Cecilia Procopiuc, William T. Rankin, Arnold Reif, Steve Tate, Neil Tweedy, Hongyan Wang, and also Ken Robinson for his aid in preparation of the paper.

Appendix A. Recombinant DNA Technology. In the last decade there have been revolutionary advances in the field of biomedical engineering, particularly in recombinant DNA and RNA manipulation. Basic principles of recombinant DNA technology are described in [WGWZ] (along with a detailed account of the history of discovery), as well as in [WHR] and [OP]. Detailed theoretical discussions of dynamics, thermodynamics, and structure of DNA, RNA and certain proteins are given by Brooks et al. [BKP] and Sinden [Si].

Due to the industrialization of the biotechnology field, laboratory techniques for recombinant DNA and RNA manipulation are becoming highly standardized, with well written lab manuals (e.g., [SFM]) detailing the elementary lab steps required for recombinant DNA operations. Thus these recombinant DNA operations which where once considered highly sophisticated are now routine. As a further byproduct of the industrialization of the biotechnology field, many of the constraints (such as timing, pH, and solution concentration, contamination, etc.) critical to the successful execution of these recombinant DNA techniques are now very well understood, both theoretically and in practice.

Biological researchers have recently made very creative use of recombinant DNA techniques for various biological applications (these are unrelated to computation, but using what may perhaps be viewed in hindsight as biomolecular computation techniques); for example, the work by Alper [Al] in drug discovery. Also, Bartel and Szostak [BS] isolated new ribozymes from a large pool of random sequences and Eigen and Rigler [EiR] developed techniques for sorting molecules by closeness metrics.

Appendix B. Limitations of Biomolecular Computation. Molecular computation seems to have certain advantages of scale over conventional methods of computation; nevertheless, if biomolecular computations simply utilize the current and well established lab methods of recombinant DNA manipulation, they have some severe limitations. In fairness we must also take into account many factors, such as the bit width of the conventional computations, as well as volume limitations and errors of biomolecular computation. So the full potential of biomolecular computation is by no means well understood or realized.

The main bounding parameters of biomolecular computations are:

1. *Total number of distinct elements of a test tube T.* This may be the most important limitation since the volume of the test tubes to be processed can be at most some small number of liters, unless industrial size operations are to be done. Thus the number

of distinct elements of a test tube is limited to some number not much more than 10^{20}. This limits the size of NP search problems that may be solved by biomolecular computation using brute-force search techniques: this number of distinct elements would allow for the encoding of all 66 bit strings and hence brute-force search on SAT problems with up to 66 boolean variables. This upper bound on the number of distinct elements of a test tube limits the tape length s of the TMs that may be simulated by our techniques to at most 33.

2. *Time duration τ of each lab step*. The time duration τ of each lab step is limited by the time required for basic recombinant DNA operations required to ensure that the number of inclusion and exclusion errors are not excessive. While some operations, such as the Merge of test tubes by mixing, can be done in at most a few seconds, the times of operations (such as Separation, Cleavage, Annealing, Ligation, Denature) depending on complementary matching of DNA substrings are strongly dependent on (a) the length of required match, and (b) the allowed number of errors of matching (and may also depend on temperature, pH, solution concentration, and possibly other parameters). In general, annealing times increase with the number of unique DNA sequences in a population. If the required length of matching is small and the allowed error rate is moderate, then annealing times can range from a minute up to a few hours. Thus the time duration τ is considerably more than the step rate of a conventional machine, e.g., a 100 megahertz high-performance workstation runs at a factor 10^8 faster step rate. However, the potential advantages of molecular parallelism over sequential computation may outweigh this step rate disadvantage.
3. *Number of lab steps L* required by a molecular algorithm. Due to the extremely slow step rate of the lab operations, the number of steps should be very small.
4. *Length of complementary matching in DNA strings*, e.g., by associative matching operations.

   In conventional recombinant DNA operations, complementary matching in DNA strings is generally done on sequences of small length. In fact, all the basic lab steps which are discussed here can be done in moderate time duration, in part since we only match DNA strings of very short length.
5. *Energy*. There is a very large potential advantage with respect to energy consumption. Certain of the molecular computing operations we consider, such as Separation, may operate at approximately 10^{-19} Joules per operation. Other operations, such as Denaturing and Annealing, which use heating or cooling, may require significantly more energy. In contrast, conventional electronic computers may operate in the range of 10^{-9} Joules per operation.
6. *Error induced by the operations*. It is well understood how to cope with certain errors that may arise in biomolecular computations.

   For example, for certain operations such as Amplification the DNA strands must be *nondegenerate*: strands must be free of lengthy palindromes, replications, and Watson–Crick complementary subsequences. (Still, degenerate strands can be of specific use in certain procedures.) However, there remain other possible errors, due to inadvertent contamination and other factors, including the unsuccessful execution of each operation. Adleman [Ad2] has considered the accuracy of the operations, and argued in detail that conventional techniques would suffice to limit the errors in a restricted set of applications.

References

[Ad1] L. Adleman, Biomolecular computation of solutions to combinatorial problems, *Science* **266** (1994), 1021–1224.

[Ad2] 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

[Al] J. Alper, Drug discovery on the assembly line, *Science* **264** (1994), 1399–1401.

[Bar] W. M. Barnes, PCR amplification of up to 35-kb DNA with high fidelity and high yield from λ bacteriophage templates, *Proc*. *Nat*. *Acad*. *Sci*. *U*.*S*.*A*. **91** (1994), 2216–2220.

[BS] D. Bartel and J. Szostak, Isolation of new ribozymes from a large pool of random sequences, *Science* **261** (1991), 1411–1418.

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

[Be1] D. Beaver, Factoring: the DNA solution, *Advances in Cryptology—AsiaCrypt*94 *Proceedings*, Lecture Notes in Computer Science, Springer-Verlag, Berlin (1994). (http:// www.cse.psu.edu/ beaver/ publications/ pubindex.html) Also as Computing with DNA, *J*. *Comput*. *Biol*. **2**(1) (1995), 1–7.

[Be2] D. Beaver, A Universal Molecular Computer, revised as Molecular Computing, Penn State University Technical Memo CSE-95-001, Pond Lab, Penn State University, http:// www.cse.psu.edu/ beaver/ publications/ pubindex.html (1995).

[BKP] C. Brooks, M. Karplus, and M. Pettitt, *Proteins*, *a Theoretical Perspective of Dynamics*, *Structure*, *and Thermodynamics*, Wiley, New York (1988).

[CFKP] E. Csuhaj-Varju, R. Freund, L. Kari, and G. Paun, DNA computing based on splicing: universality results, *Proc*. 1*st Annual Pacific Symposium on Biocomputing*, Hawaii (L. Hunter and T. Klein, eds.), World Scientific, Singapore, pp. 179–190.

[EiR] M. Eigen and R. Rigler, Sorting single molecules—applications to diagnostic and evolutionary biotechnology, *Proc*. *of the National Academy of Science* **91** (1994), 5740–5747.

[F] R. Feynman, There's plenty of room at the bottom, in *Miniaturization* (D. Gilbert, ed.), Reinhold (1961), pp. 282–296.

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

[GR] A. Gehani and J. H. Reif, Microflow Bio-Molecular Computation, presented in the 4th DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June, 1998. To appear in *DNA Based Computers*, *IV*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science (ed. H. Rubin), American Mathematical Society, Providence, RI (1999). Also accepted and to appear in a special issue of *Biosystems*, 1999.

[G] D. Gifford, On the path to computing with DNA, *Science* **266** (1994), 993–994.

[HU] J. Hopcroft and J. Ullman, *Introduction to Automata Theory*, *Languages*, *and Computation*, Addison-Wesley, Reading, MA (1979).

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

[L] R. Lipton, Speeding Up Computations via Molecular Biology, Princeton University Draft (1994). (/ftp/pub/people/rjl/bio.ps on ftp.cs.princeton.edu).

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

[MR] G. Miller and J. H. Reif, Parallel tree contraction and its application. *Proc*. 26*th Annual IEEE Symposium on Foundations of Computer Science*, Portland, OR, October 1985, pp. 478–489. Published as Parallel tree contraction, Part I: Fundamentals, *Adv*. *Comput*. *Res*. **5** (1989), 47–72. Parallel tree contraction, Part II: Further applications, *SIAM J*. *Comput*. **20**(6) (1991), 1128–1147.

[OP] R. Old and S. Primrose, *Principles of Gene Manipulation*, *an Introduction to Genetic Engineering*, 5th edn., Blackwell Scientific (1994).

[P] C. Papadimitriou, Personal communication, 1995.

[Re1] J. Reif (ed.), *Synthesis of Parallel Algorithms*, Morgan Kaufmann, San Mateo, CA (1993).

[Re2] J. Reif, Parallel biomolecular computation: models and simulations, *Proc*. *Seventh Annual ACM Symposium on Parallel Algorithms and Architectures* (*SPAA*95), ACM, Santa Barbara, CA (1995), pp. 213–223.

[Re3] J. H. Reif, Local parallel biomolecular computation, Postscript versions of this paper and its figures are at http://www.cs.duke.edu/∼reif/paper/Assembly.ps and http://www.cs.duke.edu/∼reif/paper/Assembly.fig.ps, presented at 3rd DIMACS Meeting on DNA Based Computers, University of Pennsylvania (June, 1997). Also published in *DNA Based Computers*, *III*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 48 (ed. H. Rubin), American Mathematical Society, Providence, RI (1999), pp. 217–254.

[Re4] J. Reif, Paradigms for biomolecular computation, presented at First International Conference on Unconventional Models of Computation, Auckland, New Zealand, January 1998. Published in *Unconventional Models of Computation*, C. S. Calude, J. Casti, and M. J. Dinneen (eds.), Springer-Verlag, New York (January 1998), pp. 72–93. A postscript version of this paper is at http://www.cs.duke.edu/∼reif/paper/paradigm.ps.

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

[Rob] S. S. Roberts, Turbocharged PCR, *J. N.I.H. Res*. **6** (1994), 46–52.

[Rot] P. W. K. Rothemund, A DNA and restriction enzyme implementation of Turing machines, manuscript available at http://www.ugcs.caltech.edu/∼pwkr/oett.html.

[SFM] J. Sambrook, E. Fritsch, and T. Maniatis, *Molecular Cloning*: *A Laboratory Manual*, Vols. 1–3, Cold Spring Harbor Laboratory Press, New York (1989).

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

[SS] W. Smith and A. Schweitzer, DNA Computers *in vitro* and *vivo*, NEC Research Inst. Tech Report 95-057-3-0058-3.

[WGWZ] J. Watson, M. Gilman, J. Witkowski, and M. Zoller, *Recombinant DNA*, 2nd edn., Scientific American Books, Freeman, San Francisco, CA (1992).

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