ハイパフォーマンス
コンピューティング 57－5
（１９９５．８．２４）

並列オブジェクト指向言語 ABCL/f によるＲＮＡの２次構造予測

中谷 明弘　田浦 健次朗　米澤 明憲
{nakaya, tau, yonezawa}@is.s.u-tokyo.ac.jp
東京大学 理学部 情報科学科
〒 113 東京都 文京区 本郷 7-3-1

並列オブジェクト指向言語 ABCL/f と並列計算機 AP1000+ を用いた一本鎖ＲＮＡのエネルギー的に安定な２次構造の予測について論じる。塩基列から抽出された「スタック領域」の組合せを調べる探索木を並列計算機のプロセッサ上に並列に生成することで、エネルギー的安定解を求めている。従来の２次構造予測法が最安定解を唯一つ求めるものであるのに対し、我々の並列アルゴリズムは上位 K kcal （K はパラメータ）の安定構造を全て求めることができる。アルゴリズム中で用いた、探索木の枝刈と動的なロードバランスの有効性についても述べる。

RNA SECONDARY STRUCTURE PREDICTION USING PARALLEL OBJECT-ORIENTED LANGUAGE ABCL/f

Akihiro NAKAYA　Kenjiro TAURA　Akinori YONEZAWA

Department of Information Science, Faculty of Science, The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113 Japan

An RNA secondary structure prediction method using parallel object-oriented language ABCL/f and a highly parallel computer AP1000+ is reported. We focus on finding thermodynamically stable structures of a single-stranded RNA molecule. Our approach is based on a parallel combinatorial method which calculates the free energy of a molecule as the sum of the free energies of all the physically possible *stacking regions*. Our parallel algorithm finds many highly stable structures all at once, while most of the conventional prediction methods find only the most stable structure. The important idea in our algorithm is search tree pruning, and dynamic load balancing.

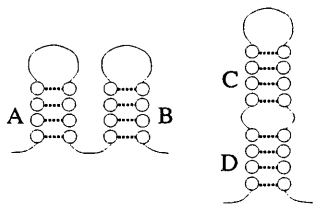
1 はじめに

ＲＮＡは4種類の塩基（Adenine, Uracil, Guanine, Cytosine）が一本鎖状に連なったものである。A–U, G–C, G–U 間に生じる水素結合によって高次構造を作る。本論文では、この高次構造のうち2次元平面上で自らと交差することなく生成されるもの（2次構造）のエネルギー的安定解について述べる。

従来から多くの2次構造予測法 [1, 5, 6, 7, 8] が試みられているが、その多くは最適解を1つ求めるものである。実際のＲＮＡ解析の場において、最安定解1つのみでは実用的ではない状況が生じている。我々は、最安定解のみではなく、最安定解から K kcal（K はパラメータ）だけ不安定な準安定解も全て求めるアルゴリズムを考案し、並列計算機上に実装した。

2 ＲＮＡの2次構造予測

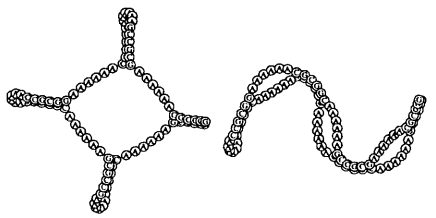
2次構造は下図のような連続した水素結合（スタック領域）からなる。図中、円が塩基を、点線が塩基間の水素結合を、A, B, C, D がスタック領域を表している。この論文では、スタック領域の組 A と B、C と D のような位置関係のスタック領域のみが共存可能であるとし、pseudoknot は考慮しないものとする。



次のような塩基列を例に挙げると、

5′ gggcgc aaaaaa gcgccg aaaaaa cggcgc aaaaaa ggcgcc aaaaaa ggcgcc aaaaaa gcgccg aaaaaa cggcgc aaaaaa gcgccc 3′

互いに共存可能なスタック領域の組合せから、次に示すような2次構造が得られる。

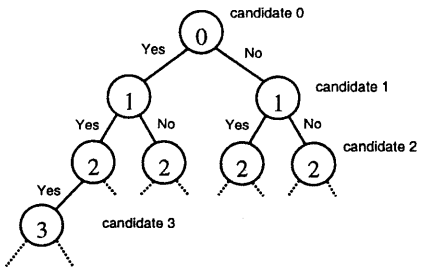


スタック領域が2次構造の安定に寄与するエネルギーの値は [3] による実験結果に基づいて算出している。

3 アルゴリズム

まず、与えられた塩基列からスタック領域となり得る水素結合の連続部分（候補:candidate）を求める。この塩基列の2次構造はこれら候補の組合せとして求められることから、全ての組合せを考えて安定するものを求めればよい。

我々のアルゴリズムでは、候補はエネルギーが安定する順に整列され、番号が付けられている（最も安定な候補を 0 番とする）。この元で、深さ i のノードが i 番目の候補を 入れる／入れない を判断する探索木を生成する。2次構造はこの木の葉として表される。ただし、この木の葉は 2^L（L は候補の数）だけあるため、不必要な部分木を効率よく刈る必要がある。この方法については次節以降に述べる。



3.1 Incompatibility islets

枝刈を効率良く行なうために [2] による incompatibility islets（単に islets とも呼ぶことにする）を用いている。Islets は全ての候補を非連結な部分に分割したものである。各部分集合（islet）に属する任意の2つの候補は互いに共存できないという条件を満たすものとする。このとき、2次構造には各 islet から高々1つの候補のみが参加できる。

3.2 枝刈の方法

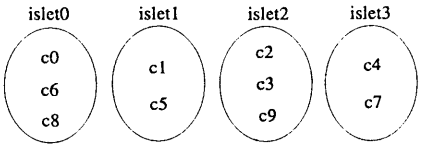
時刻 t までに求められた最も安定な2次構造のエネルギーを E_t に記録するものとする。E_0 は何らかの方法で求めた実際に存在可能な2次構造のエネルギーである。0番目の候補を 入れる／入れないを判断する根から始めて、枝を伸ばしてゆく。物理的制約から存在し得ない枝は生成しない。葉に達したときその葉が相当する2次構造の

エネルギーが E_t より安定していれば、E_t はこの2次構造のエネルギーに変更される。各ノードで枝を生成する前にその枝の下に生成し得る2次構造のエネルギーの評価を行なう。この評価値 E が

$$|E - E_t| < K \qquad \cdots\cdots \quad (*)$$

を満たしたときのみ、枝は生成される。この条件によって上位 K kcal 以内の全ての2次構造が解として得られることが保証される。

評価値 E は islets を用いて求める。c0 ～ c9 の候補が次図のような islets を形成している場合を例に説明する。



例えば、 c0, c1 が2次構造に参加することが確定しているノードでは、islet0, islet1 からの候補は既に選ばれているので、islet2, islet3 から高々1つずつのみの候補が選択される。よって、このノードの下には 候補 E = (c0, c1, c2, c4 のエネルギー和) より安定する2次構造は存在しないことが分かる。この E が条件 $(*)$ を満たさないときは枝が刈られることになる。

4 ABCL/fの概要

我々のプログラムは、富士通研究所製並列計算機 AP1000+ 上に並列オブジェクト指向言語 ABCL/f を用いて実装されている。ここでは、実装に用いた言語 ABCL/f について例を用いて簡単に述べる。詳しくは [4] を参照。

ABCL/f は表面的には Common Lisp に類似しており、並列オブジェクトの定義や、Multilisp の future を変更／拡張したものを備える。また、静的に型付けされており、コンパイル時に型エラーを検出することができる。

4.1 Future

Future は関数、メソッドの実行を呼び出し側と並列に実行する機能を提供する。例えば、$\alpha \times \beta \rightarrow \gamma$ という型をもつ関数 `f` を future を用いて呼び出し側と並列に呼び出す場合、`(future (f x y))` と記述する。この future 呼び出しの返り値は (future γ) という型をもつ future object である（ここでは、変数 `rep` に束縛しておくものとする）。上記の future を用いて呼び出された`(f x y)` の実行結果は future が返した `rep` と 関数`touch` を用いて `(touch rep)` のようにして取り出すことができる。このとき、関数 `f` の実行が終了していれば、`f` の返り値が返され、まだ終了していないときは完了するまでブロックされる。また、`(future :type` α `)` によって α 型の future object を明示的に生成することができ、`(reply x F)` によって α 型の future object `F` に α 型の値を明示的に返すことができる。また、future object は first class であり関数やメソッドの引数として渡すことが可能である。これによって、例えば、あるプロセッサが生成した future object への返答の処理を他のプロセッサに委譲することも可能になっている。

4.2 並列オブジェクト

クラスは `defcalss` 宣言によって定義することができる。定義されたクラスに対して、新しいメソッドを `defmethod` 宣言によって定義することができる。クラスインスタンスは指定したプロセッサ上に動的に生成することができる。次に示すのは、クラス `queue` とそのメソッド`enqueue` の定義の例である。ここでは、`qentry` はキューに入れられるものの型としている。

```
(defclass queue ()
 (mylength :type fixnum)
 (members  :type (list qentry)))

(defmethod queue enqueue (x)
 (declare (type qentry x) (reply-type unit))
 (setq members (cons x members))
 (setq mylength (+ mylength 1)))
```

5 並列プログラムの概略

5.1 並列化

枝刈の促進は、安定な2次構造をいかにして探索の早い段階で求めることができるかに大きく依っている。ある探索木のノードにおいて、前節で述べた評価値が良い値であっても、必ずしもそのノードの下に安定な2次構造が存在するとは限らない。我々のプログラムでは、探索木を部分木に分け、それぞれの部分木を並列計算機のプロセッサ上に同時に生成することによって、探索の早い段階でより安定な2次構造を求めている。

また、同様の理由から、安定解が全て求まっていたとしても、他に安定な解が存在しないことを確めるために探索木を作り続ける必要がある。こ

の計算の並列化も効率改善に大きく貢献することが期待できる。

5.2 各プロセッサ上のオブジェクト

プログラムが起動されると、各プロセッサ上にworker オブジェクトと queue オブジェクト（以降、それぞれ単に worker, queue と呼ぶ）が1つずつ生成される。探索木のルートに相当するノードが 0 番プロセッサ上の queue に enqueue される。以上の準備が整うと、全ての worker が探索を開始する。また、各プロセッサ上の worker は前述した E_t の値をローカルに保持しており、あるプロセッサでの E_t の変更は全てのプロセッサ上の worker にブロードキャストされる。

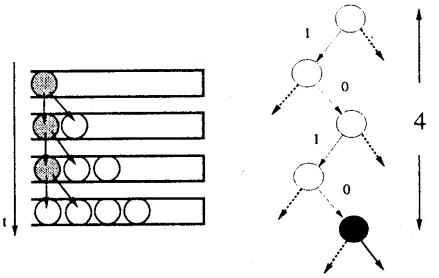
5.3 探索木の生成

探索木は queue 中に生成される。queue に入れられるものは探索木のノードを表す情報、すなわち、

- ノードの木の中での位置
- ノードの評価値 E
- 終了判定のために用いる future object

である。以降、これらのデータをまとめてタスクと呼ぶことにする。プログラム中ではこのタスクがスケジュールの単位となる。各ノードの位置は下右図が示すように探索木のルートからの枝分かれをどちらに辿ったかを示すビット列によって表される。i ビット目が 1 (0) であるとき、$i-1$ 番目の候補が選択されている（いない）ことを表す。この図の黒く塗り潰されたノードは、1010 というビット列で表される。ノードの評価値 E は前述した通りであり、future object を用いた終了判定については後述する。

探索が開始されるとき、下左図が示すように 0 番プロセッサ上の queue にのみ探索木のルートに相当するタスクが存在している。この状態で、worker がこのタスクを取り出し、高々2つの子ノードに相当するタスクを queue に入れる。queue 中のタスクは評価値の順にソートされる。



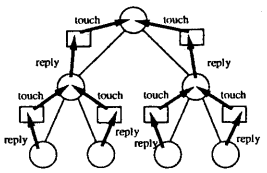
図中は簡単のため、枝刈が起こっていない状態を表しているが、実際には worker が queue から取り出したタスクに相当するノードの子が枝刈によって生成されない状況が生じるため、最終的には queue は空になる。また、E_t の値が他のプロセッサからのブロードキャストによって更新されると、前述した条件 (*) を満たさないタスクは queue から外される。

以上の操作は、次のような ABCL/f 疑似プログラムによって記述される。worker クラスの search-node メソッドは、タスクを queue から取り出し（queue が空ならば、他のプロセッサの queue に request し）、探索を行なう。探索本体は、関数 expand-node を future を用いて呼び出すことで行なっている。

```
(defmethod worker search-node ()
 (do (探索終了まで)
  (if (myqueue が空)
   (let (node (他の PE 上の queue に request))
    (future (expand-node node myqueue)))
   (let ((node (dequeue myqueue)))
    (future (expand-node node myqueue))))))
```

関数 expand-node は次のようなこと行なっている。タスク P を引数として呼ばれたとき、P が葉であれば、P 中の future object に reply をして終了する。P が葉ではないとき、左右の子 L, R を生成する。2つの子それぞれが終了したときに reply する future object を生成し、それぞれを repL と repR と共に queue の中に入れる。この後、L, R を根とする部分木に相当する探索が終了するのを touch を用いて待つ。探索木のルートの2つの子に相当するノードから reply があったとき、探索全体が終了する。探索木のノードの親と子は future object を介した木構造を形成している（次図参照）。探索木のノードは複数のプロセッサ上に動的に分散配置されるため、最初に enqueue された queue からなくなったとしてもそのノードに関する処理が終了したか否かは明らかではない。このため、ABCL/f で提供される future object を介した終了判定を行なっている。次図に

示す通り、探索木の葉（もしくは枝刈が起こり、子が作られなかったノード）が future object を介して終了メッセージを発行する。これが、次々と伝達され探索木の根に達したとき、探索が終了したことを知ることができる。future object は、複数のプロセッサ上に分散して生成された探索の終了を特定のハードウェア（プロセッサ間の同期用のビットなど）を利用しない汎用な記述を可能にしている。

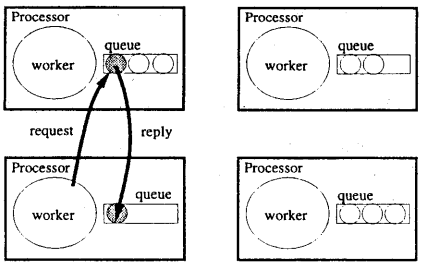


関数 expand-node は ABCL/f擬似プログラムによって次のように記述される。

```
(defun expand-node (node q)
  (let ((rep (node 中の future object)))
   (if (node は葉)
     (reply unit rep)    ; 親への返答
     (let ((L (node の左の子))
           (R (node の右の子))
           ;; ↓子に渡す future obj. 生成
           (repL (future :type unit))
           (repR (future :type unit)))
       (enqueue q L repL) ; 左の子の enqueue
       (enqueue q R repR) ; 右の子の enqueue
       (touch repL)       ; 左の子の返答待ち
       (touch repR)       ; 右の子の返答待ち
       (reply unit rep))); 親への返答
```

5.4 タスク要求先の選択

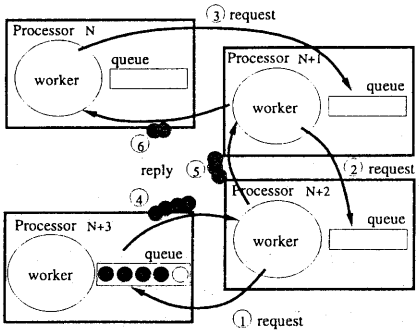
worker がデキューを行おうとするとき、queue が空であったならば、この worker は他のプロセッサ上の queue に対してタスクの要求を行なう。このとき、要求先のプロセッサは乱数で選ばれる。選ばれた先のプロセッサ上の queue も空であったときは、この要求は空でない queue に行き当たるまで転送される。



5.5 タスク要求転送の軽減

前節で述べたように乱数によるタスク要求先の選択はプログラムの構造としては単純であるが、探索の初期及び末期段階の各プロセッサの queue が空になるものが多くなったとき、タスク要求メッセージの発行とその転送が非常に多数繰り返される現象が起こる。

そこで、タスク要求の方針を次のように変更した。N 番プロセッサ上の worker は $N+1$ 番プロセッサ上の queue にタスク要求を出す。このとき、(i) $N+1$ 番プロセッサ上の queue が空でなければ、高々 i 個のタスクを貰う (i はパラメータ）。(ii) また、$N+1$ 番プロセッサ上の queue が空であれば、この queue にタスクが入るのを待つ。各プロセッサは以上の動作を行なうことにする。その結果、タスクをもたない worker が隣のプロセッサ上の queue へのタスクのエンキューを待っている鎖が生成されるが、いずれはタスクをもつ queue に行き当たることになり、必要以上のタスク要求を減らすことができる。この様子を示したのが、次の図である。



6 考察

2次構造予測について　初期段階にタスク要求が多数行なわれるのは、スケジュールの問題のみではなく、探索木の各ノードでの評価値 E の精度の低さにも原因している。あるプロセッサがタスク要求の返答としてタスクを得たとしても、このタスクの処理に要する計算時間が短ければ、再びタスク要求を発行することになる。評価値 E の精度を高めるために incompatibility islets を用いているが、更に改善する必要があるといえる。

これまで、2次構造予測に関してさまざまな手法が試みられているが、全解探索に関しては否定的な立場にあるものが多かったと思われる。しか

し、我々のアルゴリズムでは、適切な方法によって探索木のノードの評価値 E を求めれば実用的な時間内に解の集合を求められることが期待できる。

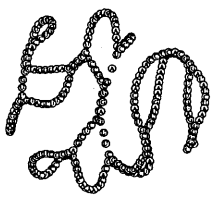
一方で、従来の逐次プログラムでは致命的であった評価値 E の不正確さを我々のプログラムでは、並列計算機を用いて並列処理することによって補っているといえる。初期段階における多数のタスク要求はアルゴリズムの性質から、完全に取り除くことは不可能であることを念頭に置きながらも、より効率的な方法の考察を進めている。

ABCL/fプログラムの性能評価 現在は AP1000+ 上の ABCL/f をターゲットに実装を行なっているが、特定のハードウェアに依存した機能は利用していないため、ABCL/f が実装されれば、いかなる計算機上でソースコードの変更なくして実行することができる。実行効率については、前述したようにタスク要求の処理に関する問題に取り組んでいるところであり、具体的なパフォーマンスデータを報告する段階には至っていない。

7 応用例

7.1 CCCV

実際に行なった2次構造予測の結果を示す。Cadang-Cadang Coconut Viroid (246塩基) のスタック領域候補の上位 95 個 ($\leq$ −9.7 kcal) からなる2次構造の予測を上位 4 kcal範囲内で行なった結果、最安定解 (−136.2 kcal) を2つ、−135.8 kal ~ −132.3 kcal の範囲に準安定解を 20 個、合計 22 個の2次構造を発見した。前に説明した通り、これらが −136.2 kcal ~ −132.3 kcal の範囲の全ての2次構造であることが保証されている。得られた2次構造の最安定解の1つ (−136.2 kcal) は次の図のような形状をしている。



7.2 ポリオウィルスの解析

我々の進めているプログラムの開発は、単に並列計算機アプリケーション作製の一例題にとどまらず、実際に、医学や生物学といった分野の現場研究者に有効なツールを提供することを最終目的としている。現在は、ポリオウィルスの強毒株と弱毒株の差異の解析について、2次構造という側面のデータの提供を目指している。従来の2次構造予測、すなわち、最安定解、もしくはそれに準ずる解を1つ求める方法ではこの解析に有益なデータを与えることはできていない。1つの RNAから得られる複数の2次構造の集合が性質の決定に関与しているのであれば、我々のプログラム、すなわち、上位 K kcal範囲内の2次構造 (K はパラメータ) を全て求める方法によって、有効なデータを提供することができる。

参考文献

[1] Y. Akiyama and T. Furuya. Folding of large RNA sequence using a hopfield-type neural network. Technical report, IEICE Tech. Rep. NC90-62, The Inst. of Elec. Info. and Comm. Eng. Japan, 1992. in Japanese.

[2] J.-P. Dumas and J. Ninio. Efficient algorithms for folding and computing nucleic acid sequences. *Nucleic Acid Research*, Vol. 10, No. 1, pp. 197–206, 1982.

[3] W. Salser. Globin mRNA sequences; analysis of base pairing and evolutionary implications. *Cold Spring Harbor Symp. Quant. Biol.*, Vol. 42, pp. 985–1002, 1977.

[4] Kenjiro Taura, Satoshi Matsuoka, and Akinori Yonezawa. ABCL/f: A future-based polymorphic typed concurrent object-oriented language – its design and implementation –. In G. Blelloch, M. Chandy, and S. Jagannathan, editors, *Proceedings of the DIMACS workshop on Specification of Parallel Algorithms*, 1994.

[5] K. Yamamoto, Y. Kitamura, and H. Yoshikura. Computation of statical secondary structure of nucleic acids. *Nucleic Acids Research*, Vol. 12, pp. 335–346, 1984.

[6] K. Yamamoto and H. Yoshikura. Computer program for prediction of the optimal and suboptimal secondary structure of long RNA molecules. *CABIOS*, Vol. 1, pp. 89–94, 1985.

[7] M. Zucker. On finding all suboptimal folding of an RNA molecule. *Science*, Vol. 244, pp. 48–52, 1989.

[8] M. Zucker and P. Stiegler. Optimal computer folding of large RNA sequence using thermodynamics and auxiliary information. *Nucleic Acids Research*, Vol. 9, pp. 133–148, 1980.