DNA コンピュータ制御コードの最適化

Code Optimization for a DNA Computer

阿部 正佳　　萩谷 昌己
Seika ABE　　Masami HAGIYA

東京大学情報理工学系研究科
Graduate School of Information Science and Technology, University of Tokyo.
{abe,hagiya}@is.s.u-tokyo.ac.jp

瀬川 修
Osamu SEGAWA
Precision System Science.
segawa@pss.co.jp

本論文では ANP-96 という DNA 計算の実験を自動的に行うロボットに対する，整数線形計画法を用いたコード生成の実装について述べる．このロボットは与えられたプログラムに従って複数の実験操作を並列に実行することができる．しかし，現在の ANP-96 のプログラミング環境は，ロボットのリソースアロケーションと命令スケジューリングが自動化されておらず，実験に本質的ではない低レベルな記述を必要とするため多くの手間を必要とする．そこで，我々はコンパイラの分野で提案された整数線形計画法に基づく最適化手法を用いて，自動化された ANP-96 プログラミング環境を実装中である．

図 1: ANP-96

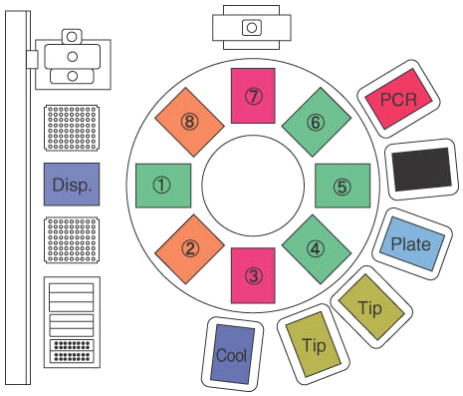

図 2: ANP-96 模式図

1 はじめに

ANP-96 (Algorithmic NA Processor)[1](図 1) はプレシジョンシステムサイエンスがオリンパスの協力のもとに共同開発した遺伝子解析実験のためのロボットであり，遺伝子発現頻度計測等の生物学実験におけるサンプル注入から反応までのプロセスを自動的に行う装置である．ロボットの動作は制御プログラムによって与えられ，ロボットの各部分を制御するコマンドに変換される．このロボットは複数の実験を並列かつ効率的に実行することが可能であり，従来は約 3 日費やしていた実験が 6 時間で行えるようになった．

DNA コンピュータの分野は， Adleman の DNA 分子を用いたハミルトン経路問題の解決 [7] に始まる．その後，特に実用的な観点からは， DNA コンピュータの技術を遺伝子の発現解析や SNP 解析に用いる試みが活発に行われており [4]，特に陶山たちは，DNA 分子を用いて 3SAT [12] 等の組合せ問題を解く

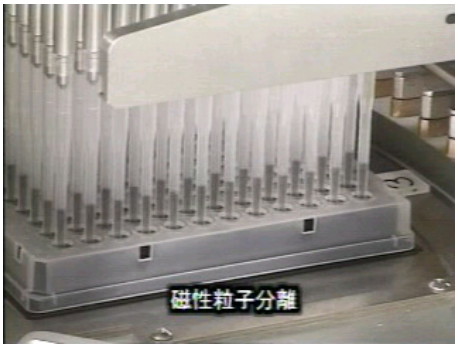

図 3: 磁性粒子分離 (Magtration)

ように設計された実験操作が， 発現解析や SNP 解析にそのまま適用できることを示した [11]．彼らは本論文で扱うような自動ロボットを DNA コンピュータとして用いて遺伝子解析を行っている [6]．

ANP-96 の特徴として， 磁気ビーズを伴う実験操作 (Magtration)(図 3) の完全な自動化が挙げられる．このようなロボットの制御プログラムの効率的実行は，はリソースを有効に割り当て，各装置を可能な限り並列に実行して，複数の実験をなるべく短時間に終了させるという点で，従来のコンパイラにおけるコード最適化に類似しているが，この分野では従来のコンパイラ技術をそのまま適用することが出来ないために，今だ自動化はされていないのが現状である．そこで，我々は従来のコンパイラ技術を発展させ，ANP-96 のような自動ロボットの自動的かつ最適なコード生成に適用することを提案した [2][3]．本稿では特に ANP-96 に対して実装中のプログラミング環境について紹介する．本節で ANP-96 の概要を説明し，2 節ではコード生成系の概要，3 節では実装上の問題点について述べる．

最後に Appendix として，比較的大きな制御プログラムのコード生成例を載せた．

1.1 ANP-96 の動作

以下，ANP-96 の動作を通常の CPU に近い視点で説明する (図 2)．ロボットの行うことを抽象的に表現すると,「入力」である複数の与えられた溶液から始まり，以下のような操作を繰り返し適用することで，新たな溶液 (これも複数) を作り出すことである．この最終的な「出力」である溶液を調べることにより，実験が終了する．

(1) 特定の溶液を混合して，新しい溶液を作る (Mixing，懸濁)．

(2) 溶液から磁気ビーズという仕掛けにより，ある物質 (溶液と考えて良い) を取り出すこと (Magtration，磁性粒子分離)．

(3) 溶液を特定の温度にまで暖めること (Warming，温度制御)．

(4) 溶液を特定の温度で一定時間維持すること (Wait，維持)．

溶液を保持する器として，プレートとチップがある．プレートは溶液を入れる器である (図 3 下の穴のあいた板)．またチップとはガラス製の先 (下) 細りの管 (図 3 の複数の管) であり，溶液を保持することが出来る．実際にはチップは複数の管からなり，チップラックというチップを納める複数の穴をもつ容器に置かれており，チップに対する操作はこのチップの 1 セットに対して同時に行われる．プレートも同様に，複数の管に対応する複数の穴からなり，この穴の各々に溶液が入る．

プレートとチップの個数は事実上制限はないが，ロボットは上記の操作を有限 (8 個) のテーブル (実験場所) で行なわなければならない．また上記の (3) が可能なテーブルは 2 つしかなく，さらに (1) や (2) を行うための IMU という装置は一つしかない．以下がロボット本体の主要モジュールである．

1. IMU (Integrated Magtratoin Unit)
2. Dispenser (ディスペンサ)
3. Conveyer (ターンテーブル，ロボットアーム)
4. Stocker (ストッカ)

ターンテーブルは上述の 8 個のテーブル (うち 2 つは温度制御可能) を備えた回転式の装置であり，IMU 直下にあるテーブルのみが IMU に関連した処理を直接行うことが出来る．IMU はチップを装着し，先細りの管の下部から溶液を排出，あるいは吸入する．例えば懸濁 (mix) という操作は IMU がチップを装着し，その下のテーブルに置かれたプレートにはチップと別の溶液が入っている状態で，IMU がチップの溶液の吸入，排出を繰り返すことで行われる．磁性粒子分離 (Magtration) も同様であるが，溶液にビーズ (金属粒子) が入っており，IMU は磁気をかけてビーズをチップに付着させるという処理を行う．ストッ

カはプレートやチップラックを提供する装置，ディスペンサは溶液を提供する装置である．またロボットアームはテーブルに置かれたプレートやチップを別のテーブルに移動する装置である．

1.2 値とレジスタ

ANP-96 にコンパイラの技術を適用する場合，値とレジスタをどう考えるかが問題である．溶液は実験対象であり，それを保持するのはプレートであるが，これらを値とレジスタと考えても意味がない．問題なのは有限のリソースをうまく割り当てることと，各モジュールの並列性を有効に利用することであり，割り当てを最適化すべき有限のリソースは，実験が行われる場所を提供するテーブルである．

よって，我々はこの有限個のテーブルを CPU の「レジスタ」と見なす．溶液はかならずプレートに保持されていなければならないので，溶液自体を単独で値と考えることは出来ない．代って，我々はテーブルに対して行われる「動作」のリストを値と考える．例えばあるテーブルにプレートが置かれ，さらに，そのプレートにある溶液が分注された場合，そのテーブルの「値」はその二つの動作のリストになる．

このような抽象的な扱いでも，リソース割り当てと命令スケジューリングは十分に記述可能であり，従来のコンパイラ技術が利用出来る．

2 コード生成系

ANP-96 のようなロボットの命令を記述するために，我々が利用している汎用的アセンブラ言語を簡単に紹介する．詳しくは [2][3] を参照のこと．以下は (簡略化された) ANP-96 のためのアセンブラ言語の宣言及び入力プログラムである．

```
/* レジスタ集合 */
IMU = {IMU1}
TBL = {T1, T2, T3, T4, T5, T6, T7, T8}
T37 = {T3, T7}
STK = {STK1}
DSP = {DSP1}

/* 型と実行時間の宣言 */
dispense : (t:TBL,s:DSP).t=(),s=()    1
newrack  : (t:TBL,s:STK).t=(),s=()    1
attach : (t:TBL,i:IMU).t=(t),i=(t)    1
detach : (t:TBL,i:IMU).t=(t,i),i=()   1
warm : (t:T37).t=(t)                 10
wait : (t:T37).t=(t)                 10
mix :(t:TBL,i:IMU).t=(t,i),i=(t,i)   10
mag :(t:TBL,i:IMU).t=(t,i),i=(t,i)   10
move : (t1:TBL,t2:TBL).t2=(t1),t1=()  1
```

```
/* プログラム */
warm(t1)      t1 を暖める (時間かかる)
move(t0,t1)   t0 上の分注済みプレートを t1 へ
wait(t1)      t1 を特定温度で維持 (時間かかる)
move(t1,t2)   t1 上の成果物を t2 へ
mag(t3,i)     t3 を磁性粒子分離 (時間かかる)
mix(t2,i)     t2 を懸濁 (時間かかる)
```

レジスタ集合はコード生成系が割り当てるべきリソースの種類を表す．TBL は ANP-96 のテーブルを，T37 はその中で温度制御可能なものを表す．IMU は一台しか無いが，複数の命令で同時に使用することは出来ないという制約を表現するために，明示的に扱う必要がある．

型と実行時間の宣言で，各命令の仕様を定義する．例えば IMU にチップを装着する attach は二つの引数 t:TBL と i:IMU を取る．そしてその実行により t は t 自身に依存した変化をし，i は t に依存した変化を受ける．最後の数字は命令の実行時間である．Magtration を行う mag では t と i それぞれが，両者に依存している．

このような命令の仕様と直線的にならんだプログラムをコード生成系に与えると，以下のようなリソース割り当て及び並列化されたプログラムが出力される．

```
warm(T3)          mag(T2,IMU1)
move(T1,T3)
wait(T3)
move(T3,T1)
                  mix(T1,IMU1)
```

リソース割り当てと命令スケジューリングを同時に最適化する問題は NP-完全であることが知られているが，我々の対象とするプログラムの規模が比較的小さいこと，そして何度でも再利用されるため時間をかけて最適化する価値がある等の理由から，我々のコードジェネレータの実装は SILP[13][9] をベースにしている．詳細は [2] を参照のこと．

3 実装

現在の制御プログラムは以下のような表の形式を取る．ここに I, D, C, S の各列にはそれぞれ IMU, Dispencer, Conveyer, Stocker に関連するコマンドが並ぶ．

I	D	C	S
i_1	d_1		s_1
	d_2	c_1	s_2
⋮	⋮	⋮	⋮

このプログラムを読み込み，実際に各装置を動かすコマンドを発行するのがスケジューラである．コマンドは基本的に 1 行づつ，そして各行のコマンドは I, D, C, S の順に，先行するコマンドが終了してから実行されて行く．ただし下に縦線を持つコマンド i_1 はその終了を待たずに次の d_1 が実行され，以後縦線が続く間並列に実行される．現在のシステムを簡単に図示すると以下のようになる．

制御プログラム → スケジューラ → ANP-96

この表形式のプログラムは先に定式化のところで述べた単純なものとはかなり異なっており，実際プログラミングも難しい．現在のスケジューラは四つの装置 (列) に対応した四つのプロセスからなり，シーケンシャル実行や並列実行はそれらプロセス間での共有メモリを使ったロックにより実現されており，かなり複雑な構造をしている．

以下が我々の新たなシステムである，ここに「制御プログラム'」は定式化で述べた形式のアセンブラ言語である．コードジェネレータはそれを ILP の問題に変換し，ILP ソルバ [5] を利用してリソースアロケーションとスケジューリングを同時に最適化し，スケジューラに渡す．

制御プログラム' → コードジェネレータ → スケジューラ'
↑ ↓
ILP ソルバ

現行の制御プログラムの形式と，我々の最適化結果の形式の食い違いによりスケジューラの修正 (上図の「スケジューラ'」) が必要である．というのは，現在の制御プログラムの形式が I，D，C，S というシーケンシャル実行の上に，拡張の形で並列制御機構を導入しているからである．

我々の定式化では，コードジェネレータが生成するのは各コマンドと，その実行開始時点の組のリストである．各コマンドが実行されるユニットは確定しているので，例えば先の表に対応する制御プログラムは以下のようなものになる．

時点	I	D	C	S
t_0	i_1	d_1		
t_1				s_1
t_2		d_2	c_1	s_2
⋮	⋮	⋮	⋮	⋮

ただし，ここでは d_1 と s_1 は並列実行が不可能であり，時点 t_2 のコマンド (d_2 と c_1 と s_2) は並列実行が可能であるとした．

しかし，現行の制御プログラムでは，時点を指定してコマンドを実行することはできない．並列制御の指定 (縦線) を伴わない，左右の隣接コマンド間では，前者が終了することを監視し，前者が終了してから後者のコマンドを発行している．従って，先の表では d_2 が終了してから c_1 が実行され，c_1 の終了後に s_2 の実行が開始される．このような制御形態を採用しているのは以下の理由による．

- 実際のコマンドは多くのパラメータを持ち，その実行時間はパラメータに依存する．
- 人間が制御プログラムを書くことを想定しているので，並列制御をデフォールトにすると，誤った制御によってロボットが破壊される危険性が高まる．

後者の理由に関して補足すると，現行の制御プログラムにおいては，コマンドの相互干渉の検査を部分的に行っているので，少なくとも逐次実行される隣接コマンド間では，それらの干渉によりロボットが破壊されないことが保証されている．しかしながら，並行制御機構を用いた場合には，そのような保証は全くない．

我々の新たなシステムを実装するにあたり，いくつかのアプローチについて検討した．一つの安全で安直なアプローチとして，我々のコードジェネレータが生成する，実行開始時点を明示する形式の生成コードを，さらに現行の制御プログラムの形式に戻し，現在のスケジューラを変更せずに利用するという方法が考えられる．この方法では，上に述べた検査機構を利用することもできる．しかし，この方法では，実行開始時点を正確に指定することはできず，コードジェネレータによる最適化の効果が半減してしまう．

上述したように，我々のコードジェネレータでは，あらかじめ実行時間が定まっている必要があるが，ANP-96 の構造上，ロボットの動作は発行されたコマンドに従って決定的に動作するのみであり，例えば対象となる溶液の状態に従って実行時間が変化するようなコマンドは無く，たとえ多くのパラメータがあったとしても，原理的にはあらかじめコマンドの実行時間を特定することは可能である．実際に，現行のスケジューラには仮想実行の機能，つまり実機

と接続しないまま実行する機能が組み込まれており，この機能を利用することで各コマンドの実行時間を特定することは原理的には可能である．

また，上のような実行時点を明示する形式のプログラミングは，人手で行うのは無理であっても，スケジューラ自体の構造は，先行コマンドの終了の監視処理が省かれた単純な構造になる．我々のコードジェネレータは，リソースの割り当てを通して，コマンドが相互干渉しないことを保証しているので，このような単純な構造であっても，安全性は十分に保たれる．以上の考察に従って，時点を明示した制御プログラムのためのインタフェースをもつように修正したスケジューラが,「スケジューラ'」である．

4 まとめと課題

DNA 実験ロボット ANP-96 のコード生成系に向けて，従来のコンパイラ技術の枠組みを応用するための抽象的なアセンブラ言語を定義し，そのコード生成系を実装した．コード生成系の実装においては，レジスタアロケーションと命令スケジューリングを同時に最適化するために，ILP に基づく手法を用いた．

実機での実行のために，既存のスケジューラを修正中である．実際の制御コマンドは多くのパラメータを持ち，それらの実行時間を正確に見積もることは難しいが，現行のスケジューラに組み込まれている仮想実行の機能を使うことで，原理的には可能であると考えている．

ILP に基づくコード生成は理論的に最適なコードが得られる一方で，本質的に NP 完全な問題な問題であることから，プログラムサイズの増加に伴って求解時間は著しく増大する．現在のコード生成系と ILP ソルバでは，十数ステップの最適化が限界である．従って実用化のためにはグラフカラーリング等に基づく安定したアルゴリズムの併用も必要である．

謝辞

ANP-96 について情報を提供して下さったノバスジーン株式会社の中島隆夫氏と，オリンパス株式会社の森本伸彦氏に感謝する．

参考文献

[1] プレシジョン・システム・サイエンス株式会社: ANP-96 マニュアル, http://www.pss.co.jp.

[2] 阿部正佳, 萩谷昌己: 線形計画法を用いた DNA コンピュータ制御コードの生成, PRO Vol.45, No.SIG9, 2004.

[3] Seika Abe, Masami Hagiya and Takao Nakajima: Code Generation for a DNA Computer by Integer Linear Programming, Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems, Singapore, 1–3 December, 2004, pp.268–273.

[4] 萩谷昌己, 横森貴: **DNA コンピュータ**, 培風館 2001.

[5] 株式会社数理システム: NUOPT マニュアル.

[6] 陶山明: 分子コンピュータの実験. **数理科学**, No.445 (7 月号), サイエンス社, pp.27–31, 2000 年 7 月.

[7] Adleman,L.: Molecular Computation of Solutions to Combinatorial Problems, *Science,* Vol.266, pp.1021–1024, 1994.

[8] Gebotys,C.H. and Elmasry,M.I.: Global Optimization Approach for Architectual Synthesis, *IEEE Transanctions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-12(9):1266–1278, 1993.

[9] Kästner,D: *Retargetable Code Optimization by Integer Linear Programming*, PhD thesis, Saarland University, 2000.

[10] Kästner,D: PROPAN: A Retargetable System for Postpass Optimisations and Analyses, *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems*, 2000.

[11] Suyama,A.,Nishida,N.,Kurata,K. and Omagari,K.: Gene Expression Analysis by DNA Computing, *Currents in Computational Molecular Biology*, pp.12–13, 2000.

[12] Yoshida,H. and Suyama, A.: Solutions to 3-SAT by Breadth First Search, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol.54, pp.9–22, 2000.

[13] Zhang,L.: *SILP. Scheduling and Allocating with Integer Linear Programming*, PhD thesis, Technische Fakultät der Universität des Saarlandes, 1996.

A 実行例

比較的大きなプログラムのコード生成例:

```
 0: (newrack t0 s0)
 1: (dispense t1 s1)
 2: (dispense t2 s2)
 3: (dispense t3 s3)
 4: (dispense t4 s5)
 5: (warm60 t3)          ; 60 度の warm
 6: (warm95 t5)          ; 95 度の warm
 7: (attach t0 i0)
 8: (mag t2 i0)
 9: (mix t1 i0)
10: (mag t1 i0)
11: (mix t3 i0)
12: (wait3 t3)
13: (mag t3 i0)
14: (move t4 t5)
15: (wait t5)
16: (mix t5 i0)
17: (wait t5)
18: (mag t5 i0)
```

レジスタアロケーションの結果:

```
  s5 : DSP1       t5 : T7         t4 : T5
  s3 : DSP1       t3 : T3         t2 : T4
  s2 : DSP1                       t1 : T2
  s1 : DSP1                       t0 : T1
  s0 : STK1                       i0 : IMU1
```

最適化されたプログラム:

```
 0: (newrack T1 STK1)       (dispense T2 DSP1)      (warm95 T7)
 1: (dispense T4 DSP1)      (attach T1 IMU1)
 2: (mag T4 IMU1)
 3:
 4:
 5: (mix T2 IMU1)
 6:
 7:
 8: (dispense T3 DSP1)    (mag T2 IMU1)
 9: (warm60 T3)
10:
11: (mix T3 IMU1)
12:
13:
14: (wait3 T3)
15: (dispense T5 DSP1)
16: (mag T3 IMU1)         (move T5 T7)
17: (wait T7)
18:
19: (mix T7 IMU1)
20:
21:
22: (wait T7)
23:
24: (mag T7 IMU1)
```