

チューナブル・アーキテクチャ計算機システムにおける  
チューニング過程について

深沢良彰\*・岸野 覚\*\*・門倉敏夫\*\*

## A Tuning Process in a Tunable Architecture Computer System

Yoshiaki FUKAZAWA, Satoru KISHINO and Toshio KADOKURA

A tuning process in a tunable architecture computer is described.

We have designed a computer system with tunable architecture. Main components of this computer are four AM 2903 bit-slice chips. The control schema of micro instructions is horizontal-type, and the length of each instruction is 104 bits.

Our tunable algorithm utilizes an execution history of machine level instructions, because the execution history can be regarded as a property of the user program. In execution histories of similar programs, same sequences of machine instructions must appear. Each sequence is synthesized into a new machine instruction by means of microprogramming. In order to select new machine instructions, this algorithm uses not only the enlarged amount of necessary control storage but also the improvement of execution efficiency.

## 1. はじめに

従来の汎用計算機は、広範囲の問題を平均的な効率で実行、処理することを目的として設計されてきた。ところが、ユーザは計算機に対し、より高い処理能力を求めてきている。この要求に対応して、さまざまな専用アーキテクチャをもつ計算機が開発されてきた。しかしながら、このような専用アーキテクチャをもつ計算機の性能を十分に発揮できる応用問題は限られているのが現状である。例えば、Floating Point Systems 社の AP-120B<sup>1)</sup> は、フーリエ変換をもちいた応用問題に対してしか優れた性能を示さない。

我々は汎用性を重視し、対象とする応用問題は限定せず、同種の問題が繰り返し実行される場合に処理効率が向上していく計算機システムを設計した。本システムは、機械語の実行履歴を解析し、まず、出現する命令列すべてを1命令に合成し、その中から適当な命令（合成

命令）を選択することにより実行効率の向上をはかるものである。

本論文ではチューニングのためのアルゴリズムおよびシミュレーションによる結果について述べる。

## 2. チューニングの概要と特徴

プログラムが実行されたときの機械語の履歴は、処理する問題の特性であると考えられる。そして、同種の問題を処理するプログラムが繰り返し実行される場合、それらの実行の履歴には、せまい範囲で見ればよく似た命令列が出現していると考えられる。このような命令列を1つの命令に合成し、命令セットに追加すれば、実行効率は向上すると考えられる<sup>2)</sup>。

本システムでは、合成を行なう命令列の対象として、ブロックと隣接命令を考えている。

ブロックとは、プログラムの制御フローに注目して分割された単位である。実行の制御はブロックの先頭命令に渡され、ブロックの最後の命令は、制御を他のブロックに移す。本システムでのブロックは一般のコンパイラ

\* 情報工学科 講師 \*\* 早稲田大学理工学部  
1985 年 10 月 30 日受付

がオブジェクト・コードの最適化を行なうときに使用する基本ブロック<sup>3)</sup>とほとんど同じである。

隣接命令とは、ブロック内で続けて実行される命令のペアのことである。

本研究と目的を同じくする Abd-Alla らの手法<sup>4)</sup>は、プログラムのループだけに注目して、これをマイクロ・プログラムで合成し、新しい 1 つの命令にすることによって、実行効率の向上を目指すものである。また、坂村らの手法<sup>5-7)</sup>は、続けて実行される命令の出現頻度だけに注目するものである。

本チューニングの特徴は、隣接命令とブロックの出現回数の他に合成、追加された命令によって増加する制御記憶の量にも考慮し、合成された命令の中から適当な命令を選択している点である。これによって、経済的で効率の良いチューニングを可能としている。本チューニングと同様の手法を用いたチューニング法の研究に坂村らの研究<sup>7)</sup>があるが、命令の選択に用いる評価関数の求め方が異なっている。

本チューニングを繰り返し行なうことにより、対象とする問題に対する機械語レベルでのアーキテクチャの是非を考察することも可能である。

### 3. 本システムのハードウェア構成の概略

本システムのハードウェアは既存のマイクロ・プロセッサ (EP) と、ビット・スライス・チップ AM 2900 シリーズ<sup>8)</sup>を用いたマイクロ・プログラム可能なプロセッサ (MP) から構成されている。これを図 1 に示す。

#### 3.1. MP の概要

MP の構成、MP のマイクロプログラムについて以下に述べる。

##### 3.1.1. MP の構成

MP は ALU 部、プログラム制御部、データバス部、I/O インターフェース部、メモリ制御部、制御記憶、主記憶から構成されている。図 2 に MP の構成図を示す。

ALU 部は 4 個のビット・スライス・チップ AM2903 で構成されている。データバス部には、16 ビットのデータバスを 4 ビット単位で取り出すセレクトを設け、オペランド圧縮を行なった命令の実行が可能となっている。プログラム制御部はプログラム・カウンタを制御する部分であり、カウンタを 1 語 (16 ビット) または 2 語増減することが可能である。制御記憶は書き換え可能であ

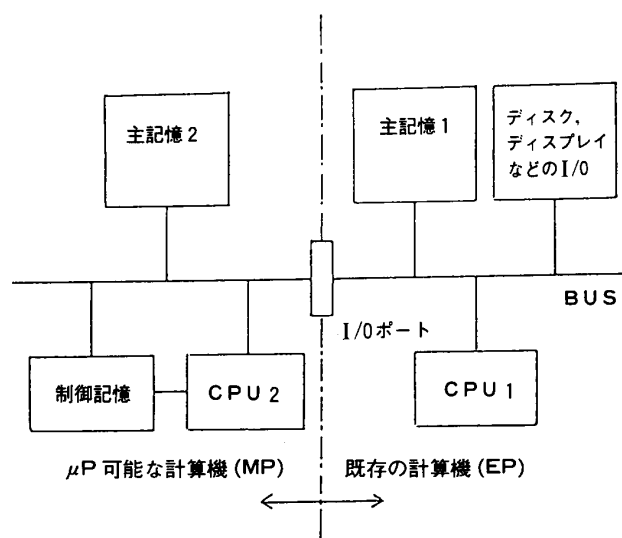


図 1 本システムのプロセッサ構成

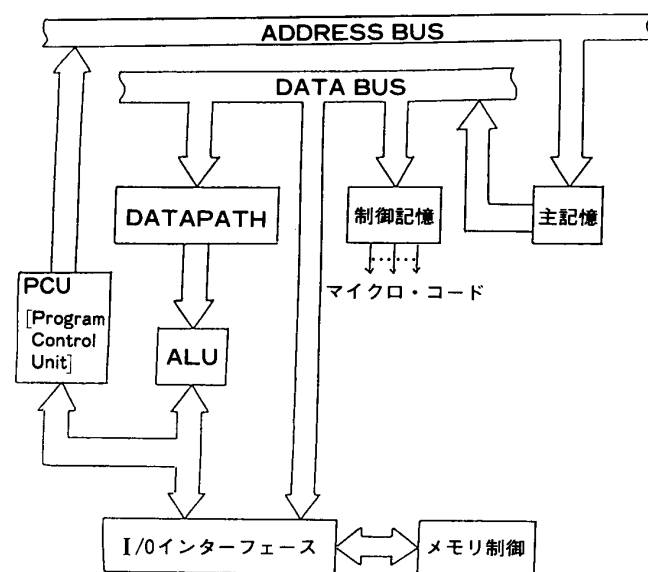


図 2 MP の構成

る。マイクロ・プログラムは、データバスを通してロードされる。制御記憶の 1 語は 104 ビットであり、容量は 2K 語である。データバス部の構成を図 3 に示す。

##### 3.1.2. MP のマイクロ・プログラム

マイクロ命令の形式は間接水平型であり、MP の各部を柔軟に制御することが可能である。マイクロ命令のフォーマットを図 4 に示す。また、命令フェッチ、命令デコード、オペランド・フェッチ、命令実行の各フェーズはすべてマイクロ・プログラムで実現されており、プログラムの実行履歴の収集も、マイクロ・プログラムで実現している。

チューナブル・アーキテクチャ計算機システムにおけるチューニング過程について（深沢良彰・岸野 寛・門倉敏夫）

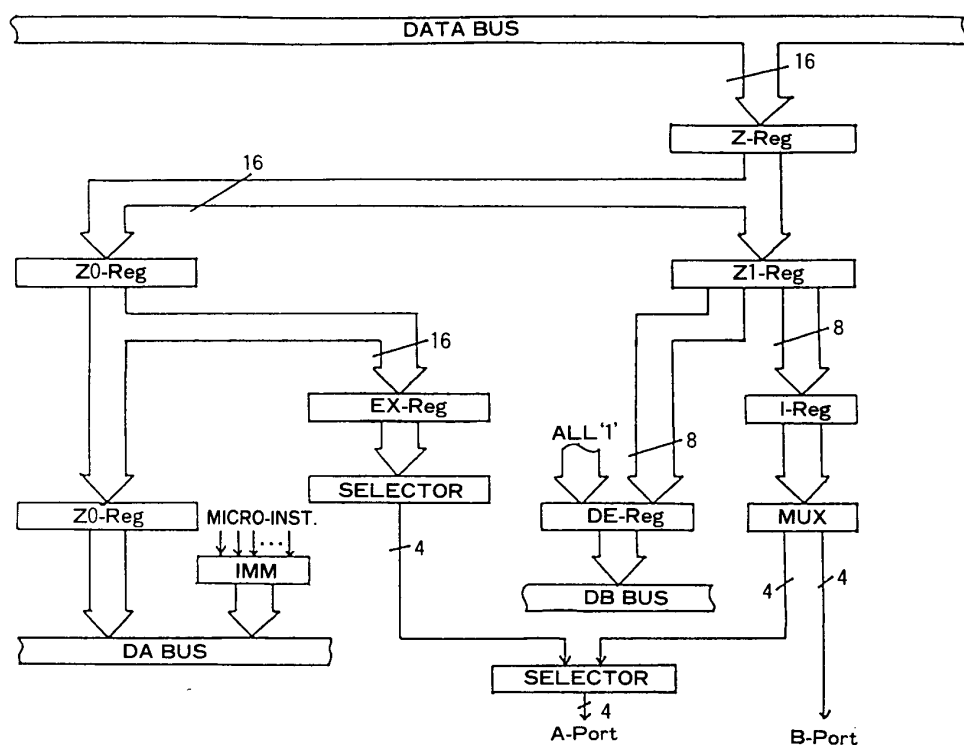


図 3 データバスの構成

Extra	Selector and Decode Control	A L U	Data Path	Program Control	Memory Control
103-96	95-91	90-78	77-65	64-54	53-49

Control Strobes	Control Bit	Status	Test	Sequence Control	Next Address & Immediate
48-43	42-35	34-26	25-20	19-16	15-0

図 4 マイクロ命令のフォーマット

#### 4. 本システムの流れ

本システムの流れを図5に示す。チューニングは以下の手順に従って行なわれる。

- (1) 対象となるプログラムを EP 上で MP 用にクロス・コンパイル、アセンブルする。
- (2) 生成したオブジェクト・プログラムを MP 用の主記憶にロードし、実行する。
- (3) 実行された機械語のアドレスが、MP のマイクロ・プログラムによって、実行履歴として出力される。
- (4) 実行履歴が解析されて隣接命令、ブロックそれぞれの出現回数を出力する。

(5) アセンブル時に生成される情報とあらかじめ設定されているマイクロ・プログラムのソースから、すべての隣接命令、すべてのブロックに対して命令合成が行なわれる。

(6) 出現回数、制御記憶の増加、実行時間の向上を考慮した評価関数を用いて合成された命令の中から新しく命令セットに加えられるべき機械語が選択される。

(7) 選択された命令のためのマイクロ・プログラムを初期設定されているマイクロ・プログラムに組み込み、統合されたマイクロ・プログラムを生成する。

(8) 命令選択によって選択された機械語に関する情報は合成機械語命令情報の作成、更新部に渡される。

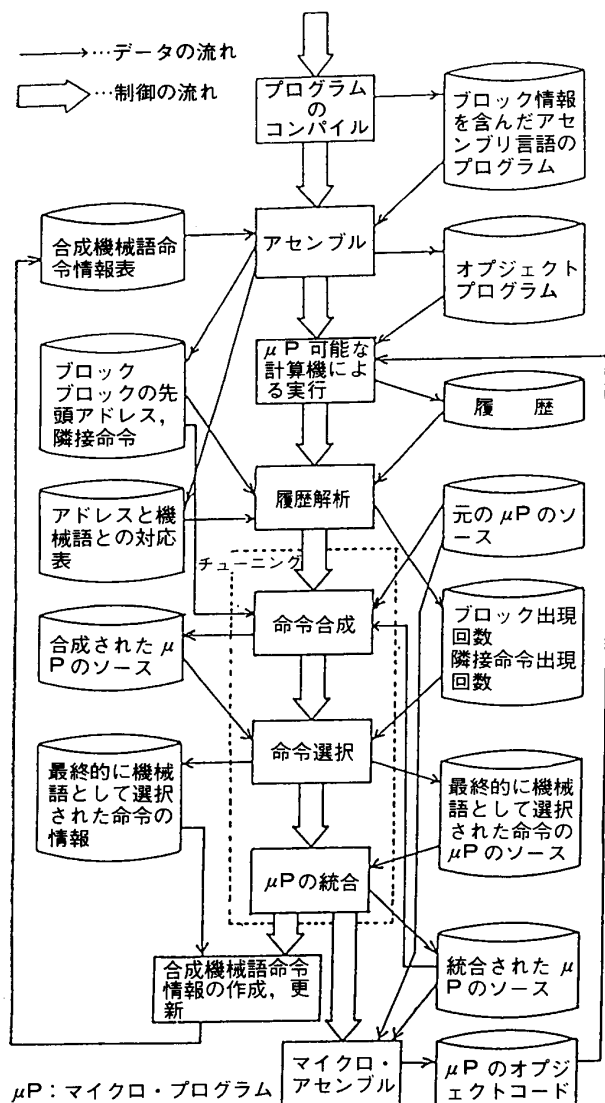


図 5 本システムの流れ

(9) 統合されたマイクロ・プログラムはマイクロ・アセンブル部でマイクロ・プログラムのオブジェクト・コードにアセンブルされ、MP の制御記憶にロードされる。

(10) 次の対象プログラムについて (1) から繰り返す。

## 5. 本システムの構成

本システムは、コンパイル部、アセンブル部、履歴解析部、命令合成部、命令選択部、マイクロ・プログラム統合部から構成されている。以下、各部について説明する。

## 5.1. コンパイル部

コンパイル部は、高水準言語によって記述されたプログラムを入力とし、ブロック情報を含んだ MP 用の初期アセンブリ言語を出力する。ブロック情報はアセンブリ言語のラベルとして出力される。

機械語の中に新たに生成された命令を真に組み込むには、多機種用のコード生成系<sup>9)</sup>を利用して、合成命令を含んだオブジェクト・コード生成を行なうことが望ましい。しかし、本システムが行なう命令合成では、コンパイル結果をアセンブリ言語で作成し、ブロック情報を出力するコンパイラがあれば一応の評価が可能のため、このような構成とした。

## 5.2. アセンブル部

コンパイル部が生成したアセンブリ言語プログラムをMP 用のオブジェクト・プログラムとして出力する。

コンパイル部が生成するアセンブリ言語は初期設定された命令に固定されている。このため、コンパイル部が出力したアセンブリ言語のプログラムを合成命令を使ったプログラムに変更する機能をもっている。

また、アセンブル部はブロックの先頭アドレス、隣接命令、ブロック、そしてアドレスと機械語との対応表を出力する。

### 5.3. 履歷解析部

履歴解析部は、MP によって実行された機械語のアドレス（実行履歴）、ブロック、ブロックの先頭アドレス、隣接命令、そしてアドレスと機械語との対応表を入力とし、ブロック出現回数、隣接命令出現回数を出力する。アドレスと機械語との対応表は、アドレスとして格納されている実行履歴から機械語を得るために使われる。

このアルゴリズムを以下に示す。

```
do while (履歴ファイルが続く);
```

アドレスを1つ取り出す;

if アドレス=ブロックの先頭アドレス

then このブロック出現回数を1増し、このアドレスから機械語を得て、隣接命令の初めの命令とする;

```
else do:
```

アドレスを1つ取り出し、このアドレスから機械語を得る；

if アドレス=ブロックの先頭アドレス

then このブロック出現回数を1増し、今得た

チューナブル・アーキテクチャ計算機システムにおけるチューニング過程について（深沢良彰・岸野 覚・門倉敏夫）

機械語は隣接命令の初めの命令であるとする；

else 今得た機械語は隣接命令の終わりの命令とし、その隣接命令出現回数を1増す；

end；

end；

#### 5.4. 命令合成部

命令合成部はブロック、隣接命令、前回のチューニングで得られたマイクロ・プログラムのソースを入力とし、すべてのブロック、すべての隣接命令に対して合成されたマイクロ・プログラムのソースを出力する。

命令の合成は、デコード部の合成、実行部の合成の順で行なわれる。

デコード部の合成においては、合成された命令のオペコードと、デコードの対象となる命令のオペコードとを比較する。この両者が一致した場合には合成された命令の実行へ制御を移すようにデコード部を合成する。しかし、デコード部が合成された段階では、合成している命令が命令選択部で合成命令として選択されるとは限らないため、この命令に対するオペコードは、決定することができない。従って、この決定は命令選択部で行なっている。

実行部の合成は次の2つの手法のどちらかによって行なわれる。

(1) 命令圧縮を行わず、命令のフェッチ、デコードを減らす。

(2) 命令圧縮を行ない、命令のフェッチ、デコード、オペランド・フェッチを減らす。

(1) はマイクロ・プログラム中のシーケンサ用命令を変更することにより実現される。(2) は合成の対象となる命令列に従って、種々の方法で行なわれる。(1) と(2) の例を以下に示す。

まず、(1) による命令合成の例を示す。対象となる命令列と合成された命令を、図6に示す。(a) はディスプレイメント DATA にインデックスレジスタ X2 の

LD R1, X2 (DATA) ..... (a)  
AR R2, R3 ..... (b)

↓  
SYN1 R1, X2 (DATA), R2, R3 ..... (c)

図6 合成対象の命令列とその合成命令〈手法(1)〉

値を加え、その値を実効アドレスとし、そのアドレスの内容をレジスタ1にロードする命令である。(b) はレジスタ2とレジスタ3の内容を加え、その結果の格納場所がレジスタ2になる命令である。(c) は本システムによって合成された命令で、(a) の演算と、(b) の演算を1命令によって行なうものである。システムはニーモニックとして、ここでは SYN1 を与えている。

(a) に対するマイクロ命令中のシーケンサ用命令を SYN1 のオペランド R2, R3 をフェッチするためのサ

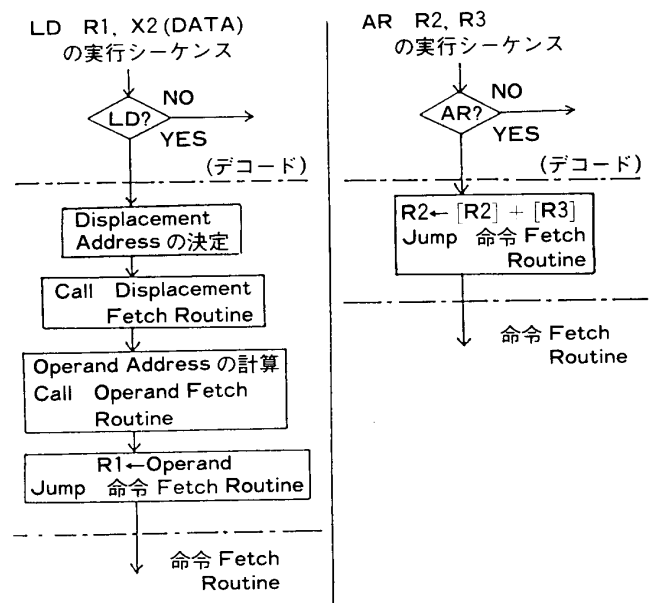


図7 LD 命令と AR 命令の実行シーケンス

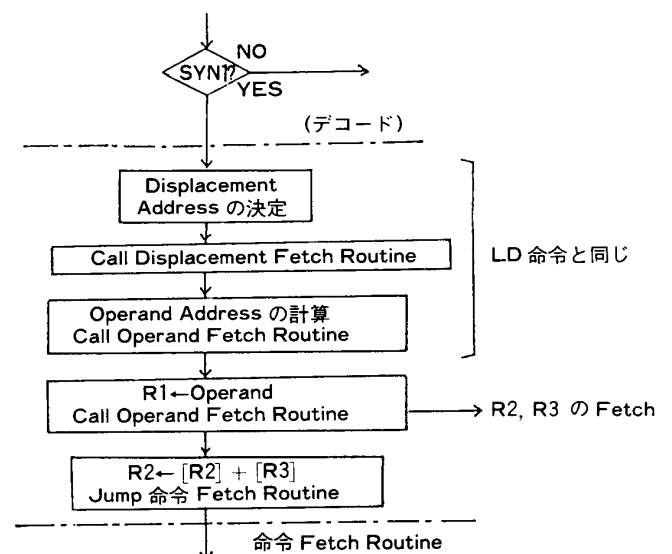


図8 手法(1)により合成された命令 SYN1 の実行シーケンス

ブルーチン・コールに変更し、この後に (b) に対するマイクロ・プログラムを結合し (c) を実行するマイクロ・プログラムが合成される。(a), (b) の実行シーケンスを図 7, (c) の実行シーケンスを図 8 に示す。

次に、(2) による命令合成の例を示す。対象となる命令列と合成された命令を図 9 に示す。対象となる命令列, (d)~(f) はレジスタ間でロード, 加算, 減算を行なう命令列であり, 各々の結果の格納場所である第 1 オペランドが同一であると仮定する。そこで, 不必要なオペランド情報の排除を行ない, オペランドを圧縮した命令を合成する。

データパス中のセクタはこのような命令を実行するためにある。セクタの機能はオペランドをバッファリングするレジスタの出力をビット単位で取り出し, ALU へ出力するものである。

(d) にあたるマイクロ命令のシーケンサ用命令がオペ

```

LR    R1, R2      .....(d)
AR    R1, R3      .....(e)
SR    R1, R4      .....(f)
      ↓
SYN2  R1, R2, R3, R4 .....(g)

```

図 9 合成対象の命令列とその合成命令〈手法 (2)〉

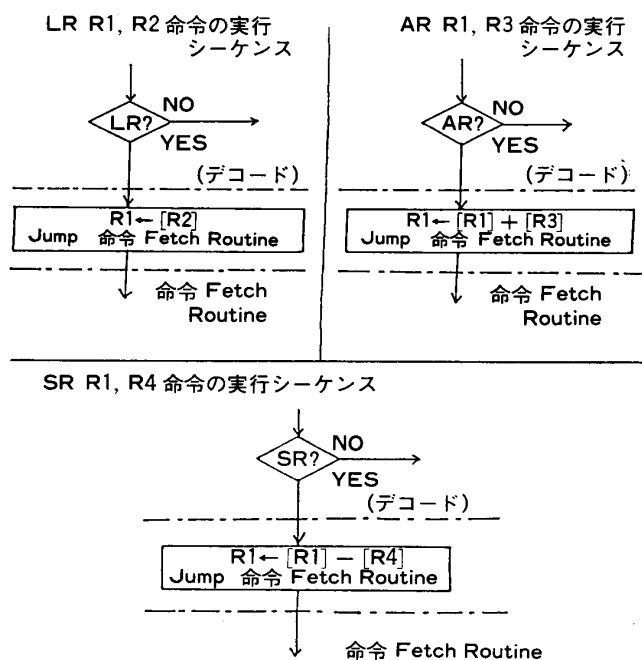


図 10 LR 命令, AR 命令, SR 命令の実行シーケンス

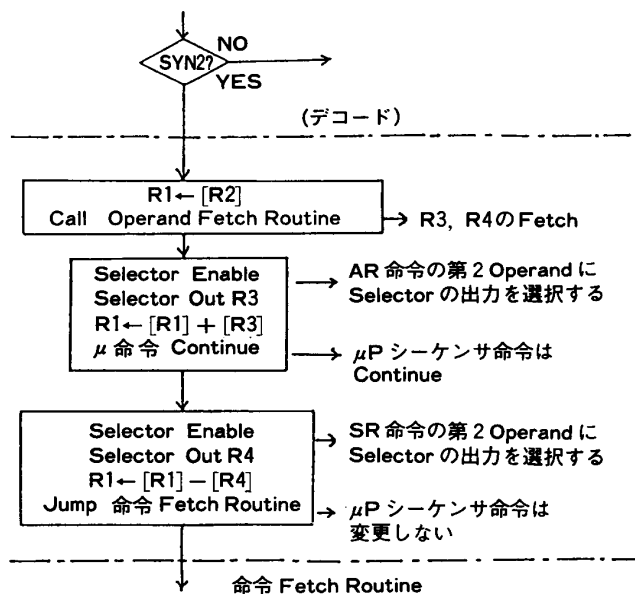


図 11 手法 (2) により合成された命令 SYN2 の実行シーケンス

ランド・フェッチのためのサブルーチン呼び出しに変更される。これにより SYN2 のオペランド R3, R4 がバッファリングレジスタにロードされる。(e) に対するマイクロ・プログラム中のセクタ制御命令は, オペランドの 1 つとして R3 を取り出すように変更される。また, シーケンサ用命令は, 次のマイクロ命令に制御を移す命令に変更される。(f) に対してはセクタ制御命令が, (e) と同様に R4 を出力するように変更される。(d), (e), (f) の各々の実行シーケンスを図 10 に, (g) の実行シーケンスを図 11 に示す。

### 5.5. 命令選択部

命令選択部はブロック出現回数, 隣接命令出現回数, そして合成されたマイクロ・プログラムのソースを入力とし, 新しく命令セットに加えるべき機械語とその情報, そしてそのためのマイクロプログラムを出力する。選択のための評価関数は次にあげる  $F_m$  を用いる。

$$F_m = K_m \cdot \Delta T_m / \Delta S_m$$

$m$ : ある隣接命令または, あるブロック

$K_m$ :  $m$  の出現回数

$\Delta T_m$ :  $m$  による実行時間の向上分

$\Delta S_m$ :  $m$  による増加制御記憶分 (時間に換算)

$\Delta T_m$  と  $\Delta S_m$  は, マイクロ・プログラムのステップ数に 1 マイクロ命令の実行時間を乗じて求めている。これは, 時間を動的に調べる事が不可能なためである。

チューナブル・アーキテクチャ計算機システムにおけるチューニング過程について（深沢良彰・岸野 寛・門倉敏夫）

そこで、静的に正確にわからない部分を近似して求めている。例えば、デコード時間に関しては、デコード用のマイクロ・プログラム量の半分の時間を換算し、マイクロ・プログラム・レベルでの条件分岐に対しては、最悪経路と最良経路の和の半分のマイクロ・プログラム量を時間に換算する。

命令選択のアルゴリズムを次に示す。

隣接命令，ブロックすべてに対して

$F_m = K_m \cdot \Delta T_m / \Delta S_m$  を計算する；

do while (空き制御記憶量 > 0)；

空き制御記憶量を補正しながら  $F_m$  が大きな値をもつ  $m$  から順に選択すべき命令の候補とする；

もし候補となった  $m$  がブロックでかつ  $m$  に含まれる隣接命令  $i$  がすでに選択すべき命令の候補なら，再び  $i$  に対する評価をやりなおす；

end；

合成命令の候補に残っている隣接命令，ブロックを合成命令として選択する；

最終的に，選択された合成命令に1バイトのオペコードをあたえる。現段階ではオペコードが1バイトの固定長となっているため，命令の総数が256をこえた場合，本アルゴリズムは終了する。

### 5.6. マイクロ・プログラム統合部

マイクロ・プログラム統合部は最終的に選択された命令のためのマイクロ・プログラムを初期命令セットのためのマイクロ・プログラムに組み込み，新しいマイクロ・プログラムのソースを作る。

## 6. シミュレーション結果

以上のアルゴリズムにより，シミュレーションで得られた結果を図12に示す。対象としたプログラムは2次方程式の解を求めるものである。また，シミュレータは，IBM 4341 システム上に PL/1 を用いて作成されている。

ここでは1回目のチューニングにおいて，ルートを計算する部分がブロックと認められ，その他いくつかの隣接命令が合成される。2回目では，いくつかの合成命令どうしも合成されている。3回目のチューニングでは，評価関数が高値を持った隣接命令が多く見つかったため実行時間比が大きく減少している。これは，命令合

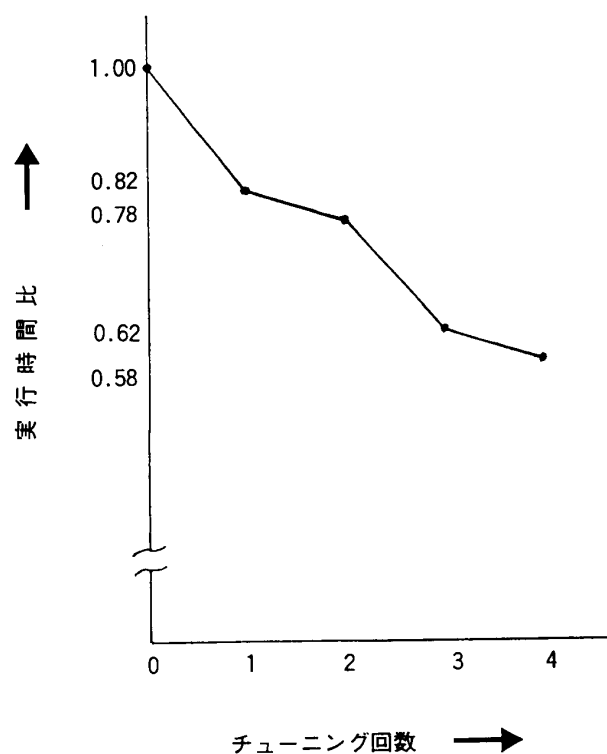


図12 チューニングの結果

成の手法(2)が有効に使用されたからである。

## 7. おわりに

本論文では，繰り返し実行される同種の問題の動的特性を機械語の実行履歴という形でとらえ，これを解析し個々の命令列をマイクロ・プログラムで1つの新しい命令に合成する。そして，合成した命令から新しい命令として適当であると評価された命令を選択することによって効率の良いチューニングを行なう方法を示した。しかし，合成されたマイクロ・プログラムの最適化を行っていないこと，本システムの評価が不十分であること，現在1バイトの固定長であるオペコード・フィールドまたは，命令全体を可変長にした場合に本システムはどのように変更され，どの程度の効果が得られるかを考察すること，などが今後の課題として残されている。

## 参考文献

- 1) Hockny, R. W. and Jesshope, C. R.: "Parallel Computers", Adam Hilger Ltd., Bristol (1981).
- 2) Rauscher, T. G. and Agrawala, A. K.: National Computer Conference, pp. 715-722 (1976).
- 3) Aho, A. V. and Ullman, J. D.: "Principles of

- Comiler Design”, Addison-Wesley Publishing Co. (1977).
- 4) Abd-Alla, A. M. and Karlgaard, D. C.: IEEE Trans. on Computers, Vol. C-23, No. 8, pp. 802-807 (1974).
  - 5) 坂村, 相磯: 電子通信学会論文誌, Nov. 1977.
  - 6) 坂村, 相磯, 飯塚: 信学技報, EC75-28.
  - 7) Sakamura, K., Morokuma, T., Aiso, H. and Iizuka, H.: National Compute Conference, pp. 499-512 (1979).
  - 8) AM 2900 Bipolar Microprocessor Family, AMD 社.
  - 9) Ganapathi, M., Fischer, C. N. and Hennessy, J. L., ACM Computing Surveys, Vol. 14, No. 4 (1982).