計算機アーキテクチャ 35－2
（1979. 9. 12）

学習機構を備えたアーキテクチャ・チューニング

坂村　健

東京大学理学部情報科学科

性能改善の程度を予測しながらアーキテクチャチューニングを行う理論と実験について述べる。

チューニングの原理は、動的頻度の高いパターンの命令をダイナミックマイクロプログラミングの技術を用いて新命令とすることにあるが、問題に適した新命令を作った場合の制御記憶の増加率と実行速度の向上率を予測して、目的に合ったチューニングを行うことに特徴がある。

理論は、バロース B1700，HP－2100 を使って実験的に確かめられ、その結果、本理論が自動アーキテクチャチューニングに限らず、ファームウエア化を定量的なデータに基づいて行う方法としても有用であることが、実証される。

1. はじめに

命令セット（ISP）レベルアーキテクチャを解くべき問題に対して自動的に適応させる機構を備えたAdaptive Computerの実現を目指して従来より行ってきたアーキテクチャチューニングの理論[1]を発展させた。新しいチューニング理論の基本的戦略は、実行時における命令セットの動的な遷移パターンを調べ頻度の高いパターンを新しい問題に適した命令として元の命令セットにダイナミックマイクロプログラム技術により付加し性能の向上を行うという既に発表した方法[1]に基づくが問題に適した命令を作った場合の制御記憶の増加量と実行速度の向上を予測してチューニングを行う点に特徴がある。すなわち作られた新しい命令を格納する制御記憶（WCS＝Writable Control Store）には量的な制限があるが、本論文で述べる方法を使う事によってWCS量にあわせたチューニングが行える。また命令パターンを発見する方法として静的な解析と動的な測定を組み合わせた新しい方法を考案しチューニングの手間を少なくしている。

本文では用語の定義から初め新しいチューニングのアルゴリズムを詳細に述べる。その後HP2100，バロースB1700を使った実験についても述べるが、その結果は提案するアルゴリズムの有用性を実証している。

2. 学習の概念を持つ自動アーキテクチャチューニングの原理

2.1. 動的な計算機の振舞いに基づく中間言語の最適設計

原理的にマイクロプログラム制御計算機は、図1に示すように階層的に構成されている。解こうとしている問題は高級言語で書かれていて、これは機械語あるいは高級言語向きに設計された中間言語の列に翻訳される。機械語，中間言語すなわち命令セットは対応するマイクロプログラムによって解釈実行される。問題をより低レベルの言語で記述する方が実行効率が良いことはよく知られている。

本論文での実行効率の改善は命令セットレベルで行われ、改善の手続きを「アーキテクチャチューニング」と呼ぶ。最大の効率を得るには全てのプログラムをマイクロプログラムで作ることは明白である。しかしプログラムの全てをファームウェア化することはコストパフォーマンス的には最良ではない。プログラムの実行の動的振舞い

を考慮し、部分的にファームウェア化すると良い。そこで部分をどのような単位で考えるか、どのような部分単位からファームウェア化するかを明確にする必要が生じる。本論文でのファームウェア化を行う単位は、命令パターンである。したがってどのような部分からファームウェア化を行うかという問題はどの命令パターンからファームウェア化を行えばよいのかという問題になる。

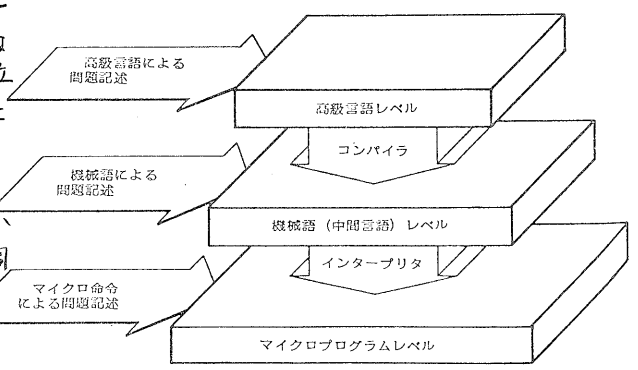


図1、マイクロプログラム制御計算機における言語レベルの階層構造

2.2 チューニングのモデルとチューニングに必要な機能

自動チューニングに関するメカニズムの簡略化したモデルを図2に示す。チューニングに関して要求される基本的機能は次の様なものである。

(1)プログラムのモニタ（モニタ）

計算機と解かれる問題の動作特性に関する詳細情報が最適化プロセスを行うのに必要である。この情報は、命令の相対頻度，実行される順序に注目した命令別の相対頻度，およびアドレスとデータの値の使用相対頻度などである。モニタはハードウェア，ソフトウェア，あるいはファームウェアで作る事ができこれらの情報を収集する。

(2)データの解析（アナライザ）

モニタから集められた情報は解析される。解析の機能は実行時間とメモリ容量を減少させる新しい命令を作るために最適化できる命令パターンの全ての可能な候補を見つけることである。これはプログラムとその実行プロフィルの解析によって行われる。

(3)計算機へのフィードバック

アナライザによって合成された新しい命令は、新しいマイクロプログラムによって解釈実行される。これによって元の命令に比べて小さいメモリ容量ですみ、実行時間を短くできる。合成されたマイクロプログラムは新しい拡張されたアーキテクチャとして以後そのプログラムが実行されるごとにWCSへロードされる。これはアーキテクチャの動的修飾であり、この時ダイナミックマイクロプログラム技術が重要な働きをする。

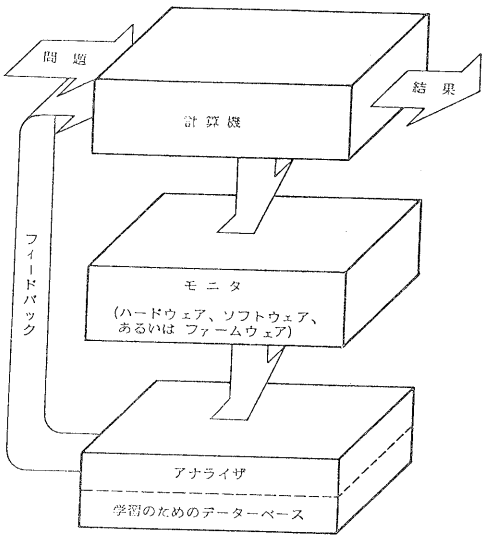


図2　自動チューニング機構のモデル

(4)チューニング動作の学習

アナライザはチューニングを行うたびにデータベースへチューニングの結果を保持する。アナライザはデータベース

の内容を参照することによって、チューニングの繰返し数を最小化することができる。

3. アルゴリズム

3-1 用語の定義

以下本論文で使われる用語について定義する。

マシン

ある中間言語(IML)の命令セットは、その命令セットIMLiに対応するマシンMiを定義する。IMLiは次の様に命令の集合で表わされる。

$$IML_i = (I_1, I_2, \cdots, I_k, \cdots, I_n)$$

ここで、I_kはマイクロ命令の列で構成されたマイクロルーチンを意味する命令である。

プログラム

プログラムは、IMLの列である。

ブロック

ブロックを定義するために、IML命令を次の2種類に分類する。

1) 分岐タイプ
次に実行する命令が必ずしも次の命令でない命令

2) 順序タイプ
次に実行する命令が必ず次の命令である命令。ブロックは次の条件を満たす命令の列である。

1. エントリポイントは先頭の命令に限る。
2. 最後の命令は分岐タイプに限る。

ブロックの長さ

ブロックBの長さ$\ell(B)$とはそのブロックを構成している命令の数をいう。

命令パターン

命令パターン(P)は、あるブロック中の連続した命令群をいう。長さ$\ell(B)$のブロックに存在する異なる命令のパターンの数は$\ell(B)\cdot(\ell(B)-1)/2$を越えない。

パターンの長さ

パターンpの長さ$\ell(P)$とは、そのパターンを構成している命令の数をいう。

パターンの荷重

パターンPの荷重$\xi(p)$は次の様に定義される。

$$\xi(p) = \sum_i^B \frac{\tau(p)\times f(p,i)}{To\times r(p)} = \frac{\tau(p)}{To\times r(p)} \sum_i^B f(p,i) = \frac{\tau(p)\times f(p)}{To\times r(p)}$$

ここで

$\tau(p)$ ：パターンpの実行時間
$f(p,i)$：ブロックiの中で実行されるパターンpの実行頻度
$f(p)$ ：全ブロックで実行されるパターンpの実行頻度
To ：プログラムの全実行時間
$r(p)$ ：パターンpの長さ

改善度

パターンPをファームウェア化した場合必要な、WCSの量をチューニングのオーバヘッドと考え、記号$\beta(P)$で表わす。改善度$\mu(P)$は、チューニングによって実行効率が改善された度合を示し、次の様に定義する。

$$\mu(p) = \frac{T(p)-t(p)}{To} = \frac{\tau(p)\times f(p)}{To} - \frac{\tau'(p)\times f(p)}{To} = \frac{\tau(p)-\tau'(p)}{To} f(p)$$

ここで

$T(p)$ ：パターンpの全実行時間
$t(p)$ ：チューニング後のパターンpの全実行時間
$\tau(p)$ ：パターンpの1回の実行時間
$\tau'(p)$ ：チューニング後のパターンp1回の実行時間
To ：プログラムの全実行時間
$f(p)$ ：全ブロック中でのパターンpの実行頻度

3-2 チューニングの予測アルゴリズム

もしWCSの容量に制限がないならば発見された命令パターンに対応する新しい命令を生て合成することにより最適なチューニングができる。しかしながら、WCSの量はシステムの価格に直接響くし、マイクロプログラムを書く手間も少なくしたい。この手間を定量的に評価するのは難しい。しかしながらマイクロプログラミングの手間がマイクロプログラムのステップ数に比例すると考えると、手間はWCSの量に比例すると考えられる。(2)
従って、最大の効果があり、かつ最小のWCS量ですむ命令パターンを選択したい。このため、ファームウェア化の効果、すなわち実行効率の改善度とWCSの増加量との比を予測する方法を命令パターンのファームウェア化に際し開発する必要が生じる。そしてもしこのような予測が可能となれば命令パターンの選択はチューニングのオーバーヘッドに基づいて行うことができる。

ここで、マイクロプログラム化された新しい命令による効率の改善度(μ)が命令パターンの荷重(ξ)およびWCSの増加量(θ)の関数と考える。すなわち

$$\mu = f(\xi, \theta) \qquad \cdots\cdots(1)$$

さらに、θは命令パターンの長さ(γ)の関数と考える。すなわち

$$\theta = g(\gamma) \qquad \cdots\cdots(2)$$

この関数fおよびgを実験的に決定する。関数fを定めるために、$\log\xi$と$\log(\mu/\theta^m)$との関数を求めた。図3に示すように、ほぼ直線的な関係がある。すなわち

$$\log(\mu/\theta^m) = n\log\xi + a \cdots(3)$$

よって、

$$\mu = A\theta^m\xi^n \qquad \cdots\cdots(4)$$

ここで、A、m、およびnは中間言語によって決定される定数である。

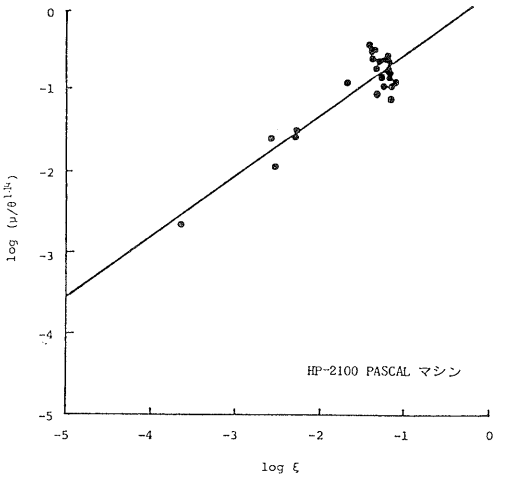


図3. $\log\xi$と$\log(\mu/\theta^m)$の関係

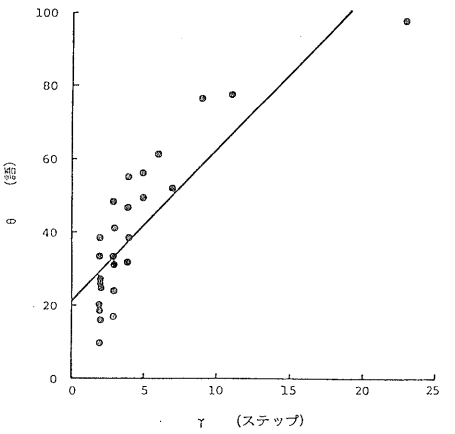


図4. γとθの関係

同様に、θとγの関係も図4に示すように一次式

$$\theta = b\gamma + c \qquad \cdots\cdots(5)$$

で表わされる。ここで、b、cは中間言語によって決る定数である。

式(4)は、命令パターンの荷重と対応するマイクロプログラムの量からファームウェア化の予測ができることを示している。そこで、この関係を「チューニング曲線」と呼ぶ。明らかに、定数、a、A、m、nは正で、荷重ξ、およびWCSの増加量θが大きくなると改善度は大きくなる。

3-3 チューニングのアルゴリズム

アルゴリズムを図5に示す。

プログラムは、静解析によって命令パターンの有限個の列で表現できる。

各々の命令パターンに重みづけを行うためにプログラムを分岐命令によってブロックに分解する。ブロックの実行頻度をハードウェアあるいはファームウェアモニタで測定すれば重みをもった命令パターンは測定されたブロックの頻度から導くことができる。

ファームウェア化する命令パターンを選択するには多くの戦略が考えられる。次の2つの方法を開発した。

(1)最大実行効率を得る方法

3-2で述べたチューニングの予測式を使って、見つけられた命令によるパフォーマンスの改善度(μ)を予測することができる。

先ずμの予測値、$\hat{\mu}$の値が最大の命令パターンを選ぶ。ある命令パターンは他と重なっていたり、一方が他方を包含していることがあるのである命令パターンを選ぶことによって他のパターンの荷重が変化することがある。従って次に再度静解析を行い命令に重みづけを行い次の命令パターンを選ぶため$\hat{\mu}$を計算しなおす。このステップを、$\hat{\mu}$の合計がある値を越えるまで繰返す。

(2)最大経済効果を得る方法

この方法は(1)で述べた方法とほとんど同様であるが、$\hat{\mu}$のかわりに、$\hat{\mu}/\hat{\theta}$を使う。ここで$\hat{\theta}$は命令パターンをファームウェア化した場合に必要とされるであろうWCSの予測値であり、これもチューニングの効果予測式から求まる。チューニングは次の条件が満されるまで繰返される。

$$\sum \hat{\mu} > s \quad \text{あるいは}$$

$$\sum \hat{\theta} > s'$$

4. 実験

4-1 実現法

上述のチューニングの手続きを自動的に行うためには以下のメカニズムが要求される。

(1)命令ブロックの荷重を定めるためのハードウェアあるいは、ファームウェアモニタ。

(2)WCSの内容を書き換えるためのダイナミックマイクロプログラム技術。

(3)高級言語のコンパイラにとってはコードが拡張されている。従ってインクリメンタルコンパイラ技術。

(4)新しい命令を合成するメカニズムを実現するための計算機。

4-2 実験システム

モニタ、アナライザ、シンセサイザから成るチューニングメカニズムを、"Automatic Performance Evaluator (APE)"と呼ぶ。

前節で述べた原理の効果を証明するために2つのAPEメカニズムを開発した。

<1>実験システム1 —HP2100+ハードウェアモニタ—

図6に実験システムの構成図を図7にチューニングとラーニングのプロセスの流れを示す。チューニングされる計算機HP2100は16ビットのデータバスを持つマイクロプログラム制御ミニコンピュータである。制御記憶は256×24ビット語の機械語をインタプリートするROMボードと、ユーザーマイクロプログラムのための同容量のWCSボード3枚から成っている。

APEのモニタとしてはCMRESS社のDYNAPROBE 7900+8000ハードウェアモニタを使っている。D7916カウント/タイムタイプハードウェアモニタおよびD8028マップ/ストアタイプモニタによってある事象の累積時間を計測し、これからの出力を解析するためにPDP-11V03を使用している。各機器間はバス接続されフィードバック

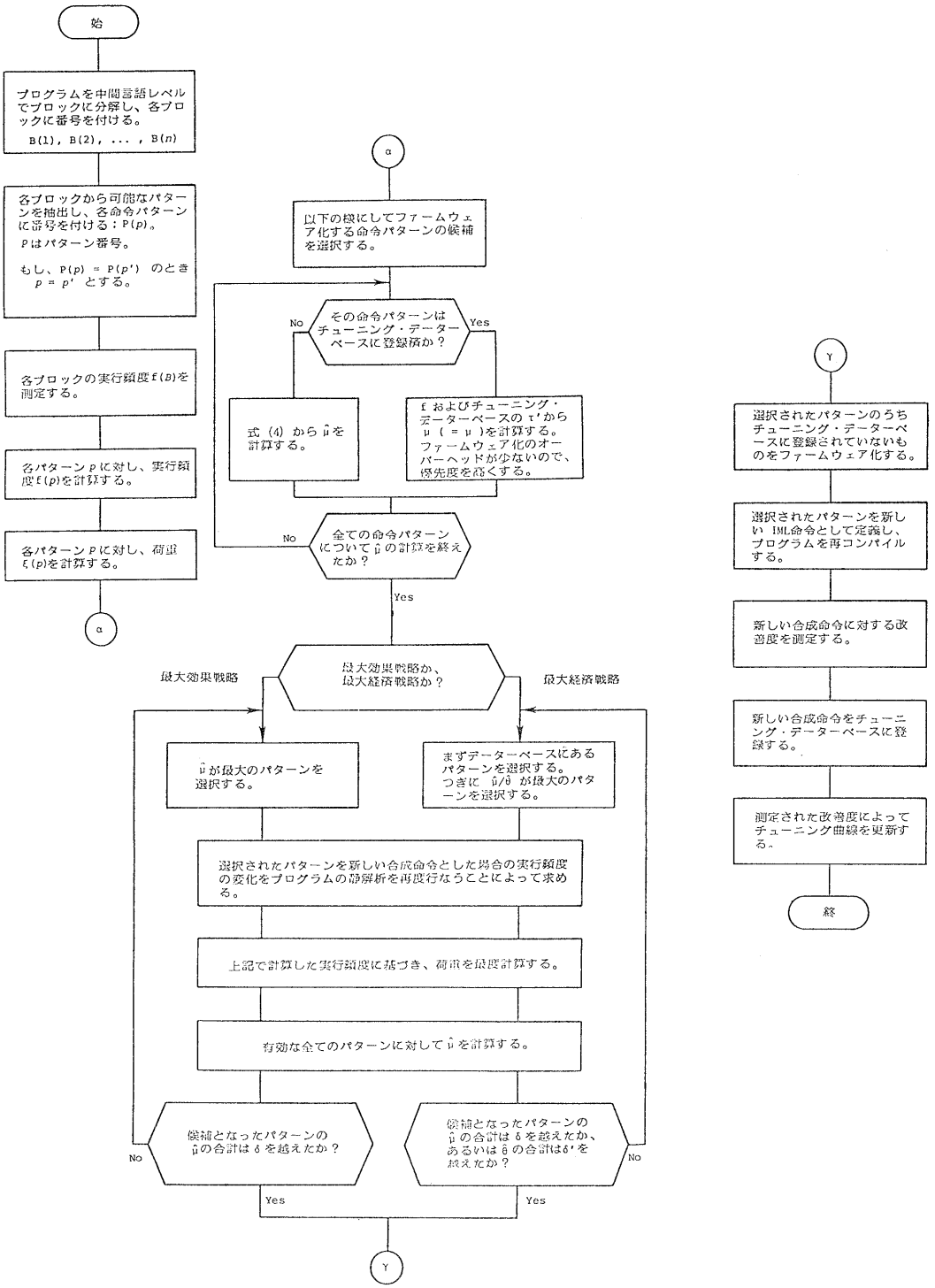


図5　チューニングのアルゴリズム

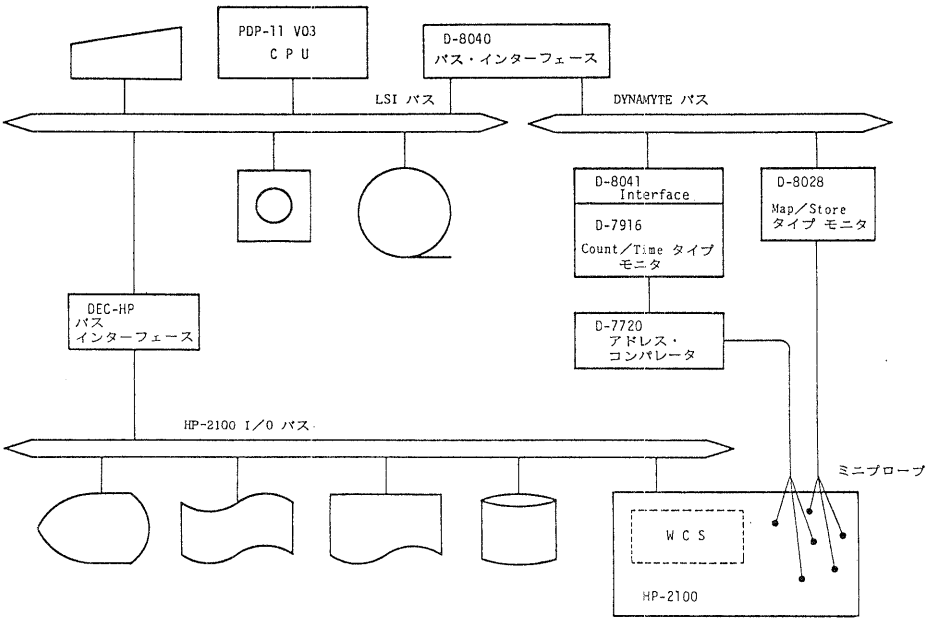


図6 実験システム構成図

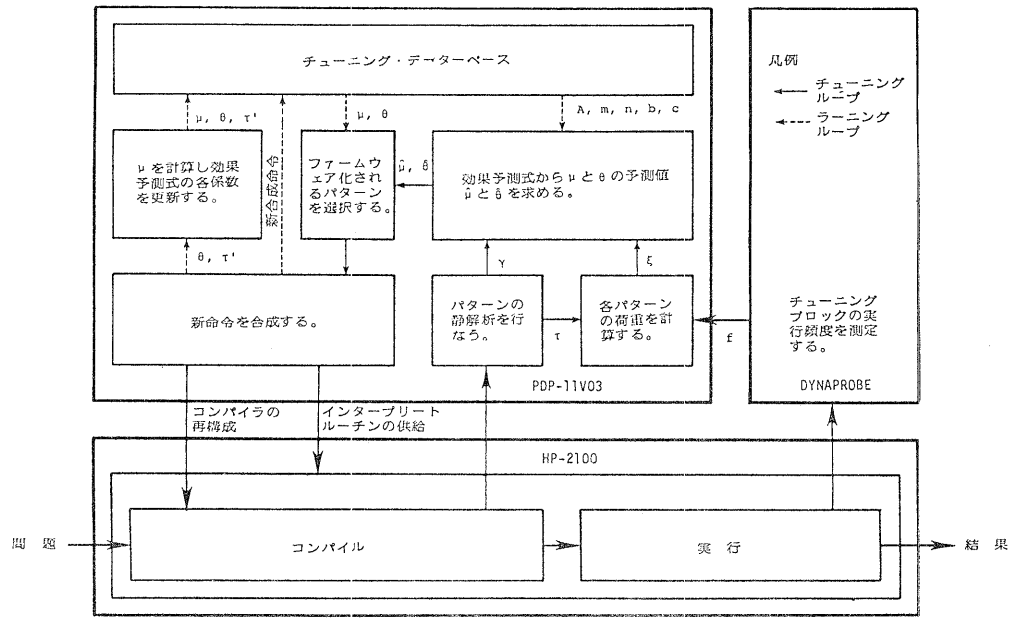


図7 実験システムのチューニング・プロセスおよびラーニング・プロセスの流れ

ループを形成する。

本システムの主な特徴の一つは、モニタ機能に対するオーバヘッドが全くないことである。これは高速のハードウェアモニタを使用し、チューニングの解析にはホスト計算機とは別にPDP11V03を使用しているためである。

<2>実験システム2 ―バロースB1700―

バロース社のB1726は、24ビットデータバスを持つマイクロプログラム可能な計算機である。この計算機は4つの汎用レジスタ、32のスクラッチパッドなど多くの内部リソースを持ちビットアドレッシング、レジスタのサブフィールドアクセスなどが可能である。マイクロプログラムを格納するWCSの容量は16ビット語で4K語である。

この計算機上にファームウェアモニタを持ったいくつかのSコードインタプリタを作成した。命令パターンの頻度を測定するファームウェアモニタは、Sコードインタープリタの命令フェッチルーチン中に埋込まれている。このためモニタリングのためのオーバーヘッドは、元来の計算機の速度に対し命令フェッチ部のみが約2倍遅くなっている。しかしながらチューニングはそれほど毎々行うわけでなく、もし行った場合でもモニタのためのマイクロプログラムはSコードインタプリタを再構成する時外してしまうので全体としてのオーバーヘッドはそれ程大きくはない。

4.3 実験に使用した中間言語

チューニングされるIMLとして次の2つのものを使用した。

(1) 60の命令から成るPASCALのためのIML

これは、K. V. Nori, V. Amman らによるもので(3)、HP2100, B-1700の両者でエミュレートした。

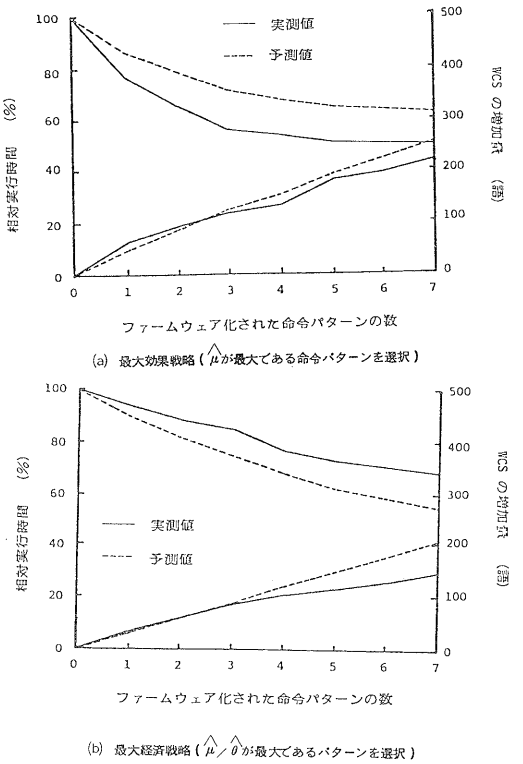


図8 チューニングの結果 (HP-2100 PASCAL マシンにおけるバブル・ソートの問題)

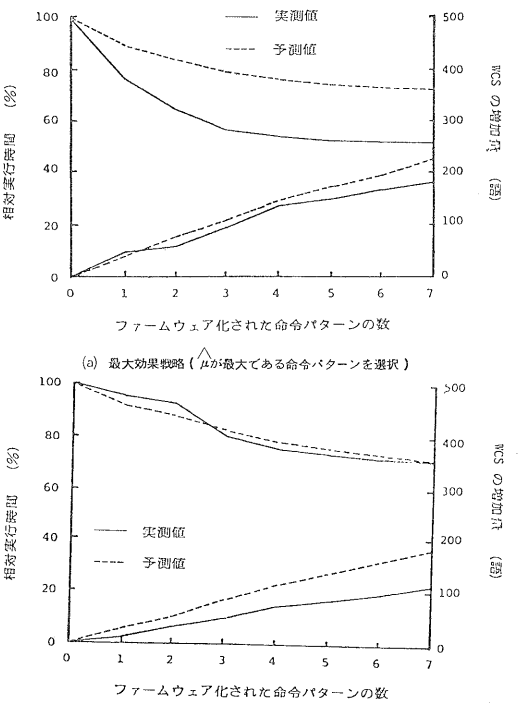
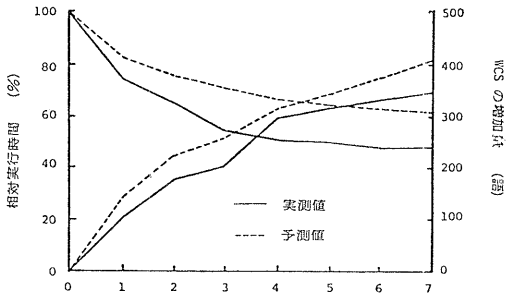
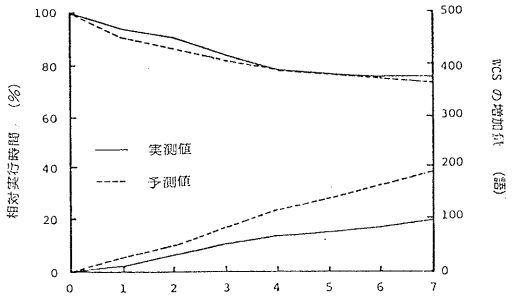


図9 チューニングの結果 (B-1700 PASCAL マシンにおけるバブル・ソートの問題)

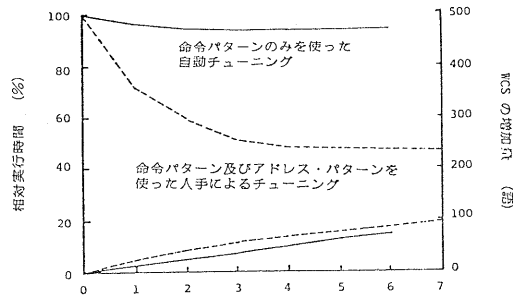


(a) 最大効果戦略（$\hat{\mu}$が最大である命令パターンを選択）



(b) 最大経済戦略（$\hat{\mu}/\hat{\theta}$が最大であるパターンを選択）

図10.
チューニングの結果（B−1700 PASCAL マシンにおけるマトリクス乗算の問題）



最大効果戦略（$\hat{\mu}$が最大である命令パターンを選択）

図11 チューニングの結果（HP−2100機械命令におけるバブル・ソートの問題）

(2) HP-2100 の FORTRAN IML すなわち HP2100上でインタプリートされる機械語の列。

5. 実験の結果と検討

(1) チューニングの結果

図8～図10にチューニングの結果を示す。図8はHP-2100によりエミュレートされるPASCALマシン上で実行されるソーティングのプログラムのチューニングの結果である。

戦略が、最大実行効率戦略の場合、プログラムの実行速度は2倍になる。一方、WCSの増加量は210ワードである。この場合、7つの命令パターンをファームウェア化している。さらにオブジェクト量は392バイトから284バイトに減少している。これは、チューニングにより効果的なコードコンパクションが行われることを示している。また最大経済効果戦略の場合、WCSの容量は130語増加し、プログラムの実行時間は30%減少した。このように、提案したアーキテクチャ自動チューニングの原理が効率向上に効果がある事が証明された。

(2) よく現われる命令パターン

表1に、チューニングの結果、現われた命令パターンのリストを示す。最大実行効率戦略で現われた命令パターンは最大経済効果戦略の場合に比べ長い命令列になっている。長い命令パターンの多くには意味を付記してある。それに反し、経済効果戦略で発見された命令パターンには意味がないものが多い。表2にその他の発見された意味のあるパターンも示す。これら命令パターンの意味を知る事は、将来の高レベル言語向き計算機のISPアーキテクチャの設計に重要な役割を果すと考えられる。

表1　しばしば現われる合成命令のリスト（HP-2100 PASCALマシン上でのバブル・ソートの問題）

最大効果戦略

No.	パターンNo.	命令パターン	意味
1	88	LAO LDO CHK DEC IXA IND	スタックに配列要素をロードする。
2	13	LDO LOD LEQ FJP	二つの値を比較し、一方が他方より小さい、あるいは等しい場合分岐しない。
3	40	LDO INC SRO UJP	ある変数を q だけ増やし分岐する。
4	70	LDO SRO	
5	41	LAO LDO CHK DEC IXA	配列のアドレスを計算をする。
6	19	LDO STO	
7	6	SRO LDC STR	

最大経済戦略

No.	パターンNo.	命令パターン	意味
1	17	DEC IXA	
2	15	LDO CHK	
3	8	LDO LOD	
4	31	LDO INC SRO	
5	10	LEQ FJP	
6	70	LDO SRO	
7	73	IND SRO	

表2　その他の意味のあるパターン

バブルソートプログラム中で発見された意味のある命令パターン

	命令パターン	意味
1.	LDO LOD LEQ FJP	if I>Work then go to ～
2.	LAO LDO CHK DEC IXA	A[I] のアドレス計算
3.	LAO LDO CHK DEC IXA IND SRO	M ← A[I]
4.	LDO INC SRO	I ← I + 1

マトリクス乗算プログラム中で発見された意味のある命令パターン

	命令パターン	意味
1.	LDO CHK DEC IXA	配列エレメントアドレスを求める基本演算
2.	LAO LDO CHK DEC IXA LDO CHK DEC IXA	A[I,J] のアドレス計算

(3) IMLの構造と計算機の構造

IML命令の構造と計算機の構造の関係がチューニングの結果に影響を与える。たとえば図11に示すHP-2100機械語のチューニングの結果では、命令パターンと同様にアドレスパターンを考慮しないと、それ程効率向上は望めない。これはHP-2100の機械語の場合、機械命令と対応するオペランドをハードウェア機構が同時にフェッチ出来るのに比べ、ユーザーマイクロプログラミングでは命令部と同時にオペランドをフェッチするのは不可能であるからである。この問題を解決するには次の2つの方法がある。

(i) 複数オペランドが取り扱えないためHP-2100の構造は、チューニングに不利である。すなわちオペランドフィールドをマイクロプログラムレベルで拡張できない。ビットアドレッシング機能をマイクロプログラムレベルで持つB-1700、およびDEC社VAX-11/780 (4)のような複数オペランドを扱える様な機能があると、この問題は比較的容易に扱える。（VAXは、ユーザーマイクロプログラムは出来ないが、）

(ii) 命令パターンとアドレッシングパターンの両方を用いて行えば、もっと良いチューニングの効果が得られる。図11の破線は、この両者を用いてチューニングを行った結果を示している。なおこれは人手で行った。しかしながらここで提案したアルゴリズムを修正することによってアドレスパターンを使ったチューニングも自動化できる。

(4) IMLとホスト計算機

PASCALおよびHP-2100の機械命令をIMLとして任意に選んだ。それにもかかわらず、かなりのパフォーマンスの改善が見られた。従って、もっとチューニングに適したIMLを用いれば"さらに良いパフォーマンスの改善が行われることが予想される。HP2100は、機械語指向で設計されているためチューニングに対する効果的な機能はない。一方B-1700はエミュレーション指向の計算機であるがハードウェア

とマスターコントロールプログラム上のソフトウェアを融合する目的でマイクロプログラムが設計されているように思われる。従っていくつかの特徴的なメカニズムはチューニングに有効利用されていない。自動チューニング指向の計算機の開発は、将来の課題である。(5)(6)。

6. おわりに。

ダイナミックマイクロプログラミング技術を用いて、ISPレベルでの計算機アーキテクチャの自動チューニングについて議論した。自動チューニング機構の簡単な説明の後、チューニング手続きの詳細なアルゴリズムを示した。提案したアルゴリズムの効果を証明する目的でHP2100とバロースB1700計算機上でいくつかの実験を行った。実験結果は元のプログラムの実行時間に対してチューニング後の実行時間が、30～60%減と、アルゴリズムの有用性を示している。

ダイナミックマイクロプログラミング技術は、プログラムの実行時に計算機アーキテクチャを動的に変更するのに大変効果がある。提案したチューニングアルゴリズムが、解く問題にとって最適な状態に計算機を変化させることができ、学習効果が大きいことが実験を通して証明された。

参考文献

(1) 坂村、相磯
「計算機アーキテクチャの自動最適化に関する考察」
電子通信学会論文誌 IECE 77/11
Vol J60-D No. 11 (1977)

(2) 坂村、諸限、飯塚、相磯
「ファームウェア化による計算機性能改善の予測」
電子通信学会論文誌 IECE 78/9
Vol J61-D No. 9. (1978)

(3) Nori, K. V. and Ammann, U.
"The PASCAL 'P' compiler: Implementation Notes"
Institut fur Informatik
Eidenossiche Technisch
Hoehschule, Zurich Report
No. 10.

(4) Strecker, W. D.
"VAX-11/780 — A virtual address extension to the DEC PDP-11 family"
proc. NCC. 47 (1978)

(5) Sakamura, K, Morokuma, T,
Aiso, H, Iizuka, H,
"Automatic tuning of computer architecture"
NCC 79, AFIPS Conference
Proceedings Vol. 48 (1979)

(6) 坂村
新しい計算機アーキテクチャー非ノイマン機能 "Adaptive Computer"
情報処理 Vol. 19. No. 12
Dec (1978)