社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－ARC－164 (3)
2005／8／3

並列事前実行における再利用表管理機構の改良

池内康樹[†] 鈴木郁真[†] 津邑公暁[†]
中島康彦[††,†††] 中島浩[†]

我々は，再利用に並列事前実行を組み合わせた，非対称な投機的マルチスレッディング機構を提案している．この並列事前実行の問題点として，実行結果を保存する再利用表に大容量 CAM が必要であり，実装が困難であることや検索オーバヘッドが大きいことが挙げられる．そのため，再利用表をより有効に活用することで，再利用表の必要容量を削減することが望まれる．本稿では，再利用表の追い出し規則を改良して有効なエントリの残存率を高くすることで，必要 CAM 容量を削減する手法を提案する．具体的には，命令区間毎の削減サイクル数を求め，削減サイクル数の大きいものの追い出しを抑止する手法を考案した．SPEC95 の perl を用いた評価では，128KB の CAM を用いた場合のサイクル削減率が 13%から 23%に向上し，従来法での 512KB の CAM と同等の結果が得られた．

An Improvement of Reuse Buffer Management for Parallel Early Computation

YASUKI IKEUCHI,[†] IKUMA SUZUKI,[†] TOMOAKI TSUMURA,[†]
YASUHIKO NAKASHIMA[††,†††] and HIROSHI NAKASHIMA[†]

We have proposed a new speculative multi-threading architecture named *parallel early computation* combined with reuse mechanism. In our earlier proposals, the architecture requires a large reuse buffer implemented by a large ternary CAM for high reuse ratio. The large capacity, however, makes it hard not only to achieve a short seek latency but also to mount the CAM onto a microprocessor chip. Therefore , it is necessary to reduce required CAM size with more effective management algorithm. This paper proposes a new management algorithm to keep useful CAM lines from replacement by dynamically evaluating the cycle reduction rate of the segment of instructions associated to the CAM lines. Our evaluation exhibits that the new algorithm improves the cycle reduction ratio of perl/SPEC95 from 13% to 23% with 128KB CAM. This performance is better than that with 512KB CAM managed by a simple LRU algorithm.

1. はじめに

命令レベル並列実行の性能限界を打破する方法として，先行命令列の実行結果の予測値に基づき後続命令列を投機的に実行する，**投機的マルチスレッディング**（Speculative Multi-Threading: 以下，**SpMT**）の研究が広く行われている．これらの多くは，通常実行スレッドと特定の予測に基づく投機実行スレッドとの間で一対一のデータ受け渡しを行うものであり，予測が不成立の場合には投機的実行結果の破棄が必要となる．また効果的に投機を行うために，再コンパイルや静的解析に基づく付加情報の埋め込み（バイナリアノテーション）を必要とするものも少なくない[1)]．

† 豊橋技術科学大学
Toyohashi University of Technology
†† 京都大学
Kyoto University
††† 科学技術振興機構さきがけ研究 21
PRESTO, JST

これに対し我々は，**再利用**を用いた SpMT である**並列事前実行**を提案している[2)]．この方法は，ある予測に基づいて事前に実行した多数の投機実行スレッドの結果を，通常実行スレッドが同一入力を検出した際に自身の過去の計算結果と同様に再利用するものである．したがって通常実行スレッドと投機実行スレッドとの関係が一対多かつ out-of-order となり，より高い性能を得ることができる．また既存のバイナリがそのまま利用可能であり，再コンパイルやアノテーションが一切不要であることも特徴の一つである．

さて，この並列事前実行によって十分な性能向上を得るためには，かなり大容量の再利用表が必要とされている．このため再利用表を構成する CAM（連想メモリ）のアクセス時間は必然的に大きなものとなり，検索や登録のためのオーバヘッドが高速化の妨げとなっている．また CAM 容量は現在の技術で 2MB 程度に達しているが，これをそのまま再利用表として用いるとすると，2 次キャッシュを大幅に上回るハードウェア量をプロセッサに追加する必要がある．また，

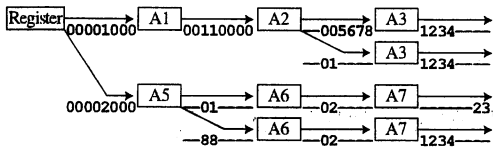

図 1　命令区間の入力参照

CAM の容量を数百 KB 程度に抑えた場合，十分に再利用の効果が得られなくなってしまう多い．これらのことから，再利用表をより有効に活用することで，より小さな再利用表で大きな性能を引き出すことが望まれる．そこで本稿では，再利用表の追い出し規則を改良し，再利用表を有効活用することで，必要 CAM 容量を削減する手法を提案する．

2. 並列事前実行

本章では我々の提案する，再利用を用いた非対称の SpMT である並列事前実行について述べる．

2.1 再 利 用

再利用とは，関数呼出しやループなどの命令区間において，その入力と出力の組を記憶しておき，再び同じ入力によりその命令区間が実行されようとした場合に，過去に記憶された出力を利用することで命令区間の実行自体を省略し，高速化を図る方法である．

再利用を行うためには，命令区間の入出力のペアを表に登録しておく必要がある．再び同じ命令区間が実行される際に，すでに同じ入力によりその命令区間が実行されている場合は，表から読み出すことで正しい出力値を求めることができる．入力セットが完全に一致していれば，実行結果は必ず同じになり，読み出した結果を検証する必要はない．ここで命令区間の入力とは，関数の引数に加え，関数およびループ内で参照される外部変数を指す．

さて，一般に命令区間内においては，ある入力値が参照され，その値によって次に参照すべき入力が決定することが多い．変数に主記憶アドレスが格納されている場合や，変数の値による条件分岐が発生する場合などがその例である．つまり，同じ命令区間であっても，参照すべき入力セットの主記憶上アドレスは，常に同じであるとは限らない．命令区間内では，一部の入力の値によって次に参照すべき入力が決定し，またその入力の値によって次に参照すべき入力が決定する，ということが繰り返される．

図 1 は，ある命令区間の入力セットのパターンをツリー構造で表現したものである．ノードは参照すべき入力アドレス，エッジはその格納値である．ここでルートからリーフまでの経路の数が，命令区間の入力セットのパターン数に対応する．この命令区間の例では，まずレジスタ上の入力が参照され，その値が 00001000 であった場合は A1，00002000 であった場合は A5 が次に参照すべき入力の主記憶上アドレスとなる．なお，参照した入力の値が異なっていても，次に参照する入力のアドレスが同じ場合もある．

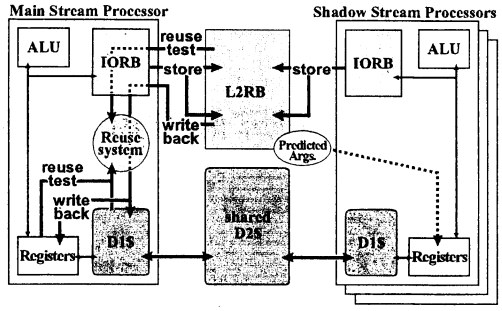

図 2　並列事前実行機構

再利用表には，このような入力セットのパターンを記憶することができる構造が必要となる．

また，この再利用表は，命令区間実行時には単純に入力の一致検索が行われるだけでなく，命令区間による出力の書き込みや，自らが書き込んだ出力の再度読み出しなど，頻繁に入出力処理を行う必要がある．ツリー構造は，多種の入力パターンを表現するには適しているものの，このような頻繁な読み書きが行われる場合はその低速さが問題となってしまう．そこで，単一の入力パターンを記録できる小規模かつ高速な再利用表 (以下，**IORB**) と，過去の出現した複数の入力パターンを格納する再利用表 (以下，**L2RB**) を用意する．命令区間実行中は IORB を用い，命令区間実行終了時に IORB から L2RB に入力セットを保存することにより，アクセス効率のよい再利用表を実現する．

2.2 並列事前実行機構

次に，並列事前実行機構の概要を**図 2** に示す．Main Stream Processor (以下，**MSP**) は，通常実行を行うプロセッサであり，可能である場合は命令区間の再利用を行う．一方，複数の Shadow Stream Processor (以下，**SSP**) は，MSP と並行して投機的実行を行う．演算器，レジスタ，IORB，一次キャッシュは各プロセッサごとに独立しており，L2RB，二次キャッシュ，および主記憶は全プロセッサで共有される．

MSP は，自身あるいは SSP が L2RB に登録したエントリを再利用する．一般的な SpMT との違いは，MSP は常に通常実行のみを行い，SSP は常に投機実行のみを行う点である．SSP は予測された入力を用いて，命令区間 (関数およびループ) の実行を MSP に先がけて行い，結果を L2RB に登録する．入力の予測が正しかった場合，MSP により通常実行が行われる時点では，命令区間は既に SSP により実行済であり，結果が L2RB に登録されているため，再利用により高速化を図ることができる．本機構では，図 2 中央に示した L2RB が，SSP と MSP の間の多対 1 のデータ引き継ぎを可能としている．

2.3 再利用表管理機構

次に，再利用表管理機構について概説する．L2RB は有限であるため，適宜エントリを削除する必要がある．再利用表管理機構では，LRU に基づいてエント

表 1 CAM の容量を変化させたときの性能変化 (Stanford)

	Bubble	FFT	Intmm	Mm	Perm	Puzzle	Queens	Quick	Towers
128KB	0.01	0.28	0.59	0.59	-0.03	0.19	0.08	0.06	0.54
2048KB	0.01	0.28	0.59	0.59	-0.03	0.68	0.08	0.06	0.57

表 2 CAM の容量を変化させたときの性能変化 (SPEC95/CINT)

	go	m88ksim	gcc	perl	compress	ijpeg	li	vortex
128KB	0.01	0.33	0.14	0.1	0.07	0.14	0.14	0.33
2048KB	0.01	0.42	0.24	0.23	0.07	0.15	0.17	0.42

リを古いものから随時削除するのに加え，L2RB の溢れが発生したときに明示的に特定のエントリを追い出すことで，これを実現している．

再利用機構はリングカウンタにより時刻管理を行っている．このカウンタは L2RB に一定数のエントリが追加されるごとにインクリメントされる．各エントリは TSID と呼ぶタイムスタンプを保持しており，L2RB に登録された時や再利用が行われたときに，現在の時刻 ID をここに保存する．また時刻 ID がインクリメントされた時点で，もっとも古い時刻 ID を TSID に持つ全てのエントリを L2RB から削除する．以下，これを **TSID パージ**と呼ぶ．

これに加え，短時間に多くの登録が発生した場合に備え，再利用エントリが枯渇した時点で明示的にエントリを削除し，空きエントリを確保する．具体的には，L2RB の各エントリに，**RFID** と呼ばれる命令区間を管理する ID が保持されており，L2RB に命令区間を登録する時点で，空きエントリが無かった場合，登録対象区間と RFID が一致するエントリを全て追い出す．以下これを **RFID パージ**と呼ぶ．RFID パージはある命令区間に対する全ての入出力セットを L2RB から削除してしまうため，サイクル数削減効果の高い命令区間を追い出してしまうこともあり，かえって性能を悪化させる可能性をはらんでいる．

3. 命令区間解析

ここでは，追い出し規則の改善手法提案のため，追い出し対象とすべきではない命令区間，および追い出し対象とすべき命令区間の特徴について Stanford と SPEC95 のベンチマークを用いて解析を行った．

3.1 対象ベンチマーク

本稿では必要 CAM 容量の削減を目的としているため，ベンチマークの中でも，CAM 容量を増大させると性能が向上するプログラム，すなわち小容量 CAM を仮定した場合に，CAM エントリの枯渇を理由に性能が低く抑えられてしまうプログラムを主な対象とする．

表 1 と**表 2** に，Stanford および SPEC95(CINT) の各プログラムにおいて，再利用表 CAM 容量を 128KB および 2MB と仮定した場合の，サイクル数削減率を示す．この結果より Stanford では Puzzle，SPEC95 では perl が，CAM 容量の増大により性能が向上することが分かる．よって以降は，主にこの 2 つのプログラムについて解析を行う．

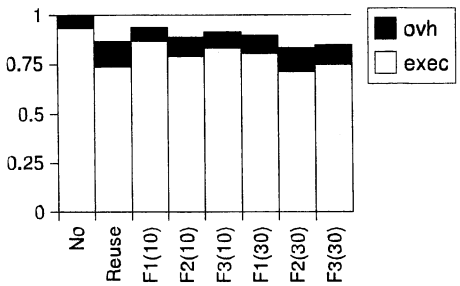

図 3 命令区間限定の影響 (perl)

3.2 解析方法

解析を行う項目は，RFID パージされた回数，TSID パージされた回数，再利用された回数，再利用されたときに削減されるサイクル数とした．これは，命令区間ごとの全体のサイクル数削減に貢献するかどうかを解析するためである．単純には，“再利用された回数 × 再利用されたときに削減されるサイクル数” によって命令区間ごとにサイクル削減数がわかる．それに加え，L2RB の追い出し状況を調べることで，命令区間の再利用機会が奪われていないかの解析も行った．

以下に，調査項目の詳細を示す．ただし，「回数」と示したものに関しては，プログラムが実行開始してから終了するまでの通算の値である．

CALL_C 命令区間の実行回数．SSP は含まない．ループの場合は 1 イタレーションで 1 回カウント．

TSID_C TSID パージの回数．L2RB の 1 エントリが追い出される毎に 1 回カウント．

RFID_C RFID パージの回数．L2RB の 1 エントリが追い出される毎に 1 回カウント．

HIT_C 再利用された回数．

RD_CYCLE 再利用による削減サイクル数の平均．

次に，命令区間を解析するにあたり，有効な評価指標を検討する．ここで，以下の三つの評価値が高いものは，L2RB から追い出すべきではない命令区間ではないかと考えられる．

- 再利用率 (HIT_C / CALL_C)
- 削減サイクル数の総和 (HIT_C×RD_CYCLE)
- 削減サイクル数 (RD_CYCLE)

再利用率が高いものは，L2RB に残すことで，再利用

表 3 spec95/perl の命令区間毎の特徴

ADR	CALL_C	TSID_C	RFID_C	HIT_C	RD_CYCLE	HIT_C×RD_CYCLE
0002ca4c	5297	27402	0	2384	216.5	516140
0001062c	10958	8402	0	6744	73.11	493045
00019458	3320	3368	0	1692	233	394236
0003c3dc	22498	272346	1979	3777	93	351261
0004bac0	5308	29155	144	738	260.23	192047
0001435c	16259	495384	3452	226	7	1582
000142d4	17935	970798	27544	159	7	1113

される可能性がより高くなると考えられる．また，削減サイクル数の総和は，全体のサイクル数削減への貢献度を表すため，残すことでより削減サイクル数が向上すると考えられる．削減サイクル数は，命令区間の粒度を表現するため，粒度が大きいものを残すことで，特に当該区間を内包する命令区間の再利用率を向上できると考えられる．

これらの評価指標のうち最も有効なものを調べるために，それぞれの指標についての上位 10 命令区間，および 30 命令区間のみを再利用対象としたときの，削減サイクル数を調べた．

図 3 に CAM 容量を 128KB としたときの評価結果を示す．グラフ内の凡例はサイクル数の内訳を示しており，exec は命令サイクル数であり，ovh は，L2RB の検索コストや，キャッシュミスなどのオーバーヘッドである．また，このグラフの項目は以下の通りである．なお，下に示した項目中にある n とはグラフ中の括弧に含まれる数字のことである．

no 再利用と SSP のいずれもなし (ovh はキャッシュミスのみ)

Reuse 再利用と SSP を用いる

F1 RD_CYCLE でソートされた命令区間のうち上位 n 命令区間のみを再利用

F2 HIT_C×RD_CYCLE でソートされた命令区間のうち上位 n 命令区間のみを再利用

F3 再利用率 (HIT_C/CALL_C) でソートされた命令区間のうち上位 n 命令区間のみを再利用

この結果より，HIT_C×RD_CYCLE でソートされた命令区間のうち上位 30 命令区間のみで実行したものがもっとも高速化されていることがわかる．

よって，ここから HIT_C×RD_CYCLE の大きい命令区間がサイクル数削減効果が大きいということがわかる．

3.3 命令区間解析

本節では，命令区間の詳細について，評価・解析を行う．なお，紙面の都合上，perl のデータのみを掲載する．**表 3** に perl の命令区間ごとの特徴を示す．この表の上 5 行に示されたデータは HIT_C×RD_CYCLE の値で降順でソートしたデータの上位 5 つのデータでありすなわち，これらの命令区間は再利用によって削減されたサイクル数の総和が大きかったものである．例えば，一番下行の命令区間 '0002ea4c' は RD_CYCLE が大きいために，全体のサイクル数削減に大きく貢献していることがわかる．また，2 行めの命令区間 '0002ea4c' は RC_CYCLE は少ないが，HIT_C が多いために，全体への貢献が大きいことがわかる．このように，これらの命令区間は，L2RB に登録されることによって削減されるサイクル数が多く，これらの命令区間を数多く登録しておくことで，再利用による効果が向上することが見込まれる．

また，下 2 行は,，再利用による削減ステップ数が少ないもののうち，L2RB に多数登録されているものを示した．これらの命令区間は数多く登録され，L2RB のエントリ数を消費しているにもかかわらず，再利用による効果が少ない．例えば，一番下行の命令区間 '000143d4' は呼び出し回数が多く，多数登録されているが，HIT_C が非常に少ない．そのため，全体へのサイクル数削減の貢献は非常に僅かに留まっている．このことから，これらの命令区間は L2RB に残しておくと，他の再利用率の高い命令区間の追い出しを無駄に増大させると考えられる．

4. 追い出し抑止の実現

次に，前章の命令区間解析をもとに，再利用表管理機構を改良する手法を提案する．

4.1 評価関数

前章から，HIT_C×RD_CYCLE の大きい命令区間の再利用表からの追い出しを抑止する処理によって，それら命令区間の再利用率を向上させ，全体の削減サイクル数を向上させることができると考えられる．

単純には HIT_C×RD_CYCLE を評価指標として，ある閾値を設け，その閾値以上の場合には追い出しを行わないと言う処理が有効であると考えられる．しかし，この指標はベンチマーク全体の合計や平均に基づくものであるため，実行中にこれを求めることはできない．そこで，一定サイクル数の期間内における命令区間毎の再利用回数と削減ステップ数を求めることで，これを近似する．これに際し，並列事前実行機構が搭載しているオーバヘッド評価機構を流用する．オーバヘッド評価機構とは，再利用により削減されるサイクル数よりも再利用表操作に要するオーバーヘッドの方が大きい命令区間は，再利用対象にしないようにするための機構である[2]．

このオーバーヘッド評価機構は，次に示す方法で削減ステップ数を計算している．ある命令区間における，最近の一定期間 T での MSP による登録回数 N^{MSP}，および SSP が登録したエントリの再利用回数 N^{SSP} を再利用機構が備える再利用頻度を記録するシフトレジスタから得る．これらの値と当該命令区間の過去の

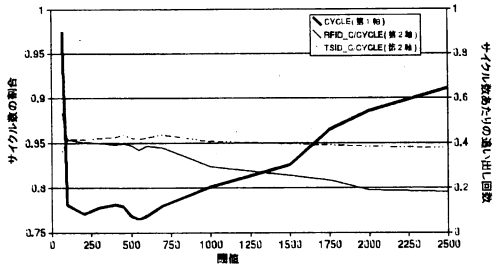

図 4 f_{rd} 閾値と性能の関係 (Puzzle)

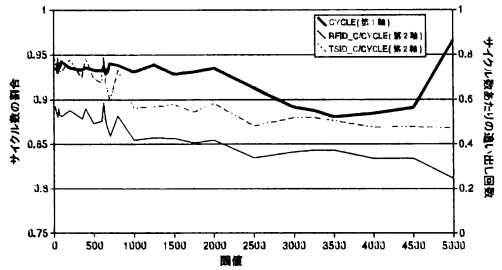

図 5 f_{rd} 閾値と性能の関係 (perl)

省略ステップ数 S から，実際に削減できたステップ数を

$$f_{rd} = (N^{MSP}+N^{SSP})\times(S-Ovh^R-Ovh^W) \quad (1)$$

として計算する．なお Ovh^R, Ovh^W はそれぞれ，過去の履歴より概算した，再利用表の検索オーバヘッドと，再利用表からキャッシュへの書き戻しオーバヘッドである．

この計算式を利用して得られた削減ステップ数は，最近の一定期間 T の間に削減されたステップ数の総和であるため，一定期間 T 内における HIT_C× RD_CYCLE の値であるといえる．そのため，ここでは f_{rd} の値を評価指標として用い，ある閾値以上の命令区間を追い出し抑止する手法を提案する．

4.2 閾値の動的算出

次に，前節で述べた f_{rd} の適切な閾値を実行中に求める方法について考える．f_{rd} の最適な閾値はプログラムによって異なるが，実行サイクル数を最小化するような値を実行中に求めることは不可能である．そこで f_{rd} に密接に関連し，かつ性能にも大きな影響を与える指標であり，かつ実行時に測定可能である L2RB の追い出し回数を利用することとした．具体的には，3 章で述べた LRU 追い出し回数である TSID_C と，命令区間の一括追い出し回数である RFID_C について，それぞれ実行サイクル数あたりの回数を予備評価し，f_{rd} の閾値設定により有効なものを選択することとした．

4.3 予備評価

f_{rd} の適切な閾値と，その L2RB 追い出し回数との関係を調べるために予備評価を行った．すなわち，

表 4 シミュレーション時のパラメータ

D1 Cache 容量	32 KBytes
ラインサイズ	32 Bytes
ウェイ数	4
レイテンシ	2 cycles
ミス ペナルティ	10 cycles
D2 Cache 容量	2 MBytes
ラインサイズ	32 Bytes
ウェイ数	4
レイテンシ	10 cycles
ミス ペナルティ	100 cycles
Register Window 数	4 sets
Window ミス ペナルティ	20 cycles/set
ロードレイテンシ	2 cycles
整数乗算 〃	8 cycles
整数除算 〃	70 cycles
浮動小数点加減乗算 〃	4 cycles
単精度浮動小数点除算 〃	16 cycles
倍精度浮動小数点除算 〃	19 cycles
IORB サイズ	32 KB
L2RB サイズ	128 KB

図 4 および図 5 に示すように，f_{rd} の閾値を変化させた場合のサイクル数の変化を測定した．なお図に示す値（左軸）は，f_{rd} による追い出し制御を行わない場合のサイクル数を 1 として正規化している．また TSID_C と RFID_C も測定し，図にはこれらを総実行サイクル数で割った値（右軸）で示している．

この 2 つの図から明らかのように，Puzzle では f_{rd} の閾値が 550，また perl では 3500 の時に，それぞれサイクル数が最小となっている．このときのサイクル数は，通常の再利用と比較して，Perl で 0.88 倍，Puzzle で 0.77 倍となっており，上に示した値付近を閾値とすることで，高速化が可能であるとわかる．

一方サイクル当たりの TSID_C の値は，どちらの図においても f_{rd} に相関した顕著な変化は見られない．しかしサイクル当たりの RFID_C の値には相関が見られ，かつ値が 0.37 付近のときの f_{rd} の閾値が，サイクル数を最小とすることが明らかになった．そこで f_{rd} の閾値の適切性を評価する指標として，“f_t=RFID_C/実行サイクル数” を採用し，f_t= 0.37 となるように f_{rd} の閾値を増減すればよいのではないかと考えた．すなわち閾値の初期値を十分大きな値とし，f_t を一定ステップごとに計測して，その値が 0.37 よりも大きければ閾値を増加させ，逆に小さければ閾値を減少させることにより，適切な閾値を動的に求めることができると予想される

5. 評　　価

以上で述べた，追い出し抑止を付加した，並列事前実行を行うシミュレータを開発し，評価を行った．

5.1 評価環境

並列事前実行機構を実装した単命令発行の SPARC-V8 シミュレータを用い，MSP および SSP のサイクルベースシミュレーションを行った．評価に用いたパ

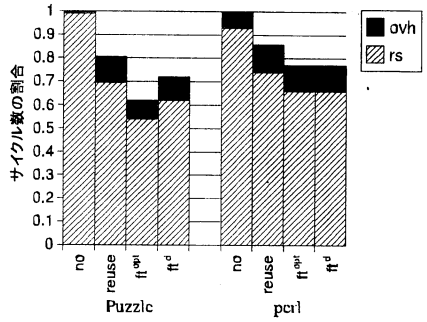

図 6 動的な f_{rd} 閾値設定の効果

ラメータを表 4 に示す．キャッシュ構成や命令レイテンシは，SPARC64-III[3] を参考にしている．ロードモジュールとしては，3 章で述べた通り，Stanford ベンチマークの Puzzle プログラムと SPEC95 ベンチマークの perl プログラムを用いた．また，閾値の動的算出のため，閾値の初期値を 5000 とし，16384 サイクルごとに f_t を評価し，0.37 よりも大きい場合には閾値を 20 加算し，小さい場合には 20 減算するようにした．

5.2 結　　果

閾値の動的算出によって実行した結果を図 6 に示す．このグラフは，左から順に

no　再利用，SSP 共に用いない．
reuse　再利用，SSP を用いる．
ft^{opt}　閾値を最適値に固定．
ft^{d}　動的に閾値を求めた

である．閾値を固定としたときに比べ，Puzzle においては性能向上は少なく留まっている．これは，閾値の探索が単純に一定期間ごとに閾値を増減させるものであったため，実行状況の変化が激しいプログラムにおいては，探索が困難であったためと考えられる．一方，perl では，閾値を固定した時とほぼ同等の性能を出すことができた．

5.3 CAM 容量変化との比較

本稿で提案した手法による，CAM 容量の削減効果について考察する．図 7 は，本稿で提案した手法を適用した場合のサイクル数 (ft^{d} 128KB) と，提案手法を用いずに CAM 容量を 128KB～2MB の範囲で変化させたときのサイクル数を，再利用や SSP を用いないときの値を 1 として正規化して示したたものである．この図から，Puzzle では従来手法の 256KB と同等の，また perl では 512KB と同等の性能が，提案手法によって得られたことが明らかになった．したがって本稿の提案手法により，CAM 容量を 1/2～1/4 に削減可能となった．

6. ま と め

本稿ではサイクル削減数の大きい命令区間の再利用率を向上させるため，命令区間の追い出しを抑止する手法を提案した．

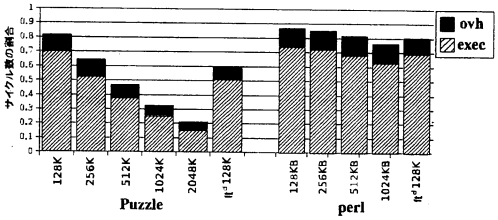

図 7 提案手法と CAM 容量増との性能比較

従来の再利用表管理機構の単純な LRU による追い出しでは，サイクル数削減効果の高い命令区間であっても，他の命令区間と同様に追い出されていた．そこで，提案手法ではサイクル数削減効果の高い命令区間の評価のために，再利用機構に搭載されているオーバヘッドフィルタによって求められた一定期間内の削減サイクル数を用いた．そして，この値が閾値以上の命令区間の追い出しを抑止する手法を提案した．また，その閾値を動的に求めるための手法も提案した．

これらの手法を実装したシミュレータを用い Stanford ベンチマークと SPEC95 ベンチマークで評価を行い，再利用率向上が得られるプログラムが存在することを示した．従来の追い出し抑止を適用しない再利用表管理機構での評価と比較した場合，Puzzle においては，従来サイクル数削減率が 19% に留まっていたのに対し，本手法を適用した再利用表管理機構では，これを 28% まで向上させることができた．これは，CAM 容量が 128KB であった場合においても，従来の 256KB 相当の性能を出すことができたといえる．また，perl においては，従来サイクル数削減率が 13% に留まっていたのに対し，本手法によりこれを 23% まで向上させることができた．これは，CAM 容量が 128KB であった場合においても，従来の 512KB 相当の性能を出すことができたといえる．

これらの結果から，追い出し規則を改良することによって，性能を向上させることができたといえる．

今後の課題としては，動的に求めている閾値の探索方法を改良し，より最適値に近い値が出るようにすることなどを検討したい．

参 考 文 献

1) Sodani, A. and Sohi, G. S.: Dynamic Instruction Reuse, *Proc. 24th International Symposium on Computer Architecture*, pp. 194–205 (1997).
2) 津邑公暁, 笠原寛壽, 清水雄歩, 中島康彦, 五島正裕, 森眞一郎, 富田眞治: 大容量汎用 3 値 CAM を用いた並列事前実行機構の効率的実現, 先進的計算基盤システムシンポジウム SACSIS2004 論文集, 情報処理学会, pp. 251–259 (2004).
3) HAL Computer Systems/Fujitsu: *SPARC64-III User's Guide* (1998).