Vol.2017-ARC-227 No.19
2017/7/27

# 自動メモ化プロセッサにおける復帰アドレス別の再利用率調査とその応用

松山 且樹[1]　藤井 政圭[1]　津邑 公暁[1]　中島 康彦[2]

**概要**：我々は，計算再利用技術に基づく高速化手法である自動メモ化プロセッサを提案している．自動メモ化プロセッサは，関数とループを計算再利用の対象としており，実行時にその入出力を再利用表に記憶しておくことで，同一入力による同一命令区間の実行を省略する．本稿では，自動メモ化プロセッサが計算再利用の対象とみなしている関数の復帰アドレスに着目し，復帰アドレス別の再利用率や削減サイクル数について調査した．その結果，関数の復帰アドレスによって再利用率が大きく異なることが確認できた．また，この調査結果に基づき，再利用率が高くなると予想されるような復帰アドレスからの呼出し時にのみ計算再利用を適用することで，自動メモ化プロセッサの性能向上を図る手法について検討した．

## 1. はじめに

これまで，さまざまなプロセッサ高速化手法が提案されてきた．ゲート遅延が支配的であった時代には，集積回路の微細化によるクロック周波数の向上により，マイクロプロセッサの高速化を実現できた．しかし，ゲート遅延に対する配線遅延の相対的な増大にともない，クロック周波数の向上が困難になってきている．こうした中で，SIMD やスーパスカラなどの細粒度な並列性に基づく高速化手法が注目されてきた．しかし，プログラム中に存在する細粒度な並列性にも限りがあることから，これらの手法にも限界がある．よって現在では，消費電力や発熱量の問題を解決しつつ，プロセッサあたりの処理能力を向上させるため，1 つの CPU に複数のコアを搭載したマルチコアプロセッサが主流となっている．

これら既存のプロセッサ高速化手法は，粒度の違いはあれど，いずれもプログラムの持つ並列性に着目したものである．これに対し我々は，計算再利用技術に基づいた高速化手法である**自動メモ化プロセッサ** [1] を提案している．並列化が処理全体の総量を変化させず複数の処理を同時実行することにより高速化を図る手法であるのに対し，計算再利用は処理自体を省略することで高速化を図る手法であり，その着眼点は根本的に異なっている．自動メモ化プロセッサは，関数およびループを再利用対象区間とみなし，実行時にその入力と出力の組を再利用表と呼ばれる表へ登録する．そして，再び同一命令区間を同一入力を用いて実行しようとした際に，再利用表に記憶した過去の出力を利用することで，その命令区間の実行を省略する．

自動メモ化プロセッサが再利用の対象区間と見なす関数は，一般にプログラム中の様々なアドレスから呼びだされる．そのため，関数が呼びだされるまでの動作により，同一の関数でも入力の規則性や周期性などの特徴が異なると考えられる．そこで本稿では，関数の復帰アドレスの違いが，関数の再利用に対しどのような影響を及ぼすかについて調査する．そしてその結果に基づき，自動メモ化プロセッサの性能向上に向けた手法を検討する．

## 2. 自動メモ化プロセッサ

本章では，本稿で取り扱う自動メモ化プロセッサの動作原理とその構成について概説する．

### 2.1 自動メモ化プロセッサの概要

計算再利用（Computation Reuse）とは，プログラムの関数やループなどの命令区間において，その入力セットと出力セットをそれぞれ実行時に組で記憶しておき，再び同じ入力によりその命令区間を実行する際に，記憶した過去の出力を再利用することで命令区間の実行自体を省略し，高速化を図る手法である．また，この手法を命令区間に適用することをメモ化（Memoization）[2] と呼ぶ．メモ化は元来，プログラミングにおける高速化のテクニックである．ただし，メモ化を適用するためには，プログラムを記述しなおす必要があり，既存のロードモジュールやバイナリをそのまま高速化することはできない．その上，ソフトウェ

[1] 名古屋工業大学
Nagoya Institute of Technology
[2] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

アによるメモ化 [3] はオーバヘッドが大きく，フィボナッチ数を求めるプログラムなどの限られたプログラムでしか性能向上が得られない．

そこで我々は，ハードウェアを用いて動的にメモ化を適用するプロセッサとして，自動メモ化プロセッサ（Auto-Memoization Processor）[1] を提案している．自動メモ化プロセッサは，プログラムの実行時に動的に関数やループを検出しメモ化を適用することで，既存のバイナリを変更することなく高速に実行できる．なお，自動メモ化プロセッサは，call 命令のターゲットから return 命令までの区間を関数，後方分岐命令のターゲットからその後方分岐命令までの区間をループとして検出する．

自動メモ化プロセッサは，一般的なプロセッサと同様に，コアの内部に ALU，レジスタ，1 次データキャッシュ（D$1）などを持ち，コアの外部に 2 次データキャッシュ（D$2）を持つ．また，自動メモ化プロセッサ独自の機構として，メモ化のために命令区間およびその入力セットと出力セットの組を記憶しておく表である再利用表（MemoTbl）をコアの外部に，メモ化制御機構（Memoize engine），および MemoTbl への書き込みバッファとして働く再利用バッファ（MemoBuf）をコアの内部に持つ．MemoTbl はサイズが大きく，コアからのアクセスコストが大きいため，入出力を検出する度に MemoTbl へアクセスするとオーバヘッドが大きくなる．そこで，このオーバヘッドを軽減するために，コアの内部に設けている MemoBuf を作業用バッファとして用いる．なお，自動メモ化プロセッサでは，レジスタおよびメモリ参照を「入力」，レジスタおよびメモリへの書き込み，さらに関数の場合はそれに加え返り値を「出力」として扱う．

自動メモ化プロセッサは再利用対象区間の開始を検知した際，MemoTbl を参照し現在の入力セットと MemoTbl に記憶している過去の入力セットとを比較する．これを再利用テストと呼ぶ．もし，現在の入力セットが MemoTbl に記憶したいずれかの入力セットと一致する場合，メモ化制御機構はその入力セットに対応する出力セットをレジスタやキャッシュに書き戻し，命令区間の実行を省略する．一方，現在の入力セットが MemoTbl に記憶したいずれの入力セットとも一致しない場合，自動メモ化プロセッサはその命令区間を通常実行しながら，その入出力を MemoBuf に登録し，実行終了時に MemoBuf の内容を MemoTbl に一括で登録することで将来の再利用に備える．

MemoBuf は複数のエントリを持ち，1 エントリが 1 入出力セットに対応する．各エントリは，どの命令区間に対応しているかを示すインデクス（FLTbl idx），その命令区間の開始アドレス（Start Addr），その命令区間の実行開始時のスタックポインタ（SP），関数の戻りアドレス，またはループの終端アドレス（retOfs），命令区間の入力セット（Read）および出力セット（Write）のためのフィールドを持つ．なお，Read フィールドおよび Write フィールドには，それぞれ複数の入出力値を保持できるようになっている．また，入れ子構造になった命令区間もメモ化の対象とするために，MemoBuf は現在実行中の命令区間に対応する各エントリをスタック構造として保持する．そのため，MemoBuf は現在使用しているエントリをポインタ（MemoBuf_top）で指しており，命令区間の検出時にそのポインタをインクリメントし，命令区間の実行終了時にデクリメントすることで，命令区間が入れ子構造の場合に対応している．

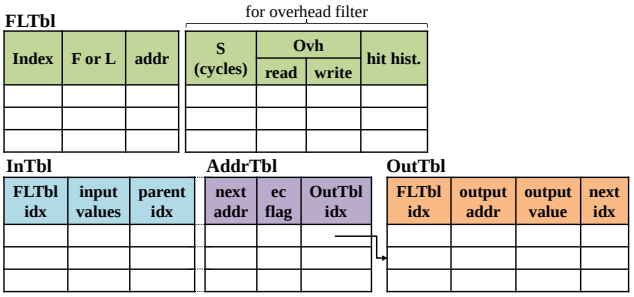



**図 1** MemoTbl の構成

ここで MemoTbl の詳細な構成を**図 1** に示す．MemoTbl は，命令区間を記憶する FLTbl，入力を記憶する InTbl，入力アドレスを記憶する AddrTbl，および出力を記憶する OutTbl の 4 つの表から構成される．FLTbl，AddrTbl，OutTbl は RAM で実装し，InTbl は高速な連想検索が可能な 3 値 CAM（Content Addressable Memory）で実装する．

FLTbl では 1 エントリが 1 命令区間に対応しており，その行番号（Index）を各命令区間の識別番号とする．また，各行にはメモ化のためのフィールドに加え，後述するオーバヘッドフィルタのためのフィールドが保持される．メモ化のためのフィールドには，関数とループを判別するフラグ（F or L）と，命令区間の開始アドレス（addr）が保持される．一方，オーバヘッドフィルタのためのフィールドには，当該命令区間のサイクル数（S），過去の再利用時に要した入力検索および出力書き戻しオーバーヘッド（Ovh read/write），過去の再利用ヒット履歴（hit hist.）が保持される．

InTbl の各エントリは FLTbl の行番号 Index に対応するインデクス（FLTbl idx）を持ち，この値を用いてどの命令区間の入力値を記憶しているかを識別する．また，命令区間の全入力パターンを木構造で管理するために，入力値（input values）に加えて親エントリのインデクス（parent idx）を持つ．この input values は，入力値が含まれるキャッシュブロックを単位として記憶するが，そのブロック内の入力値以外の部分はドントケア値とすることで，再利用テスト時の一致比較に影響を与えないようにする．なお，レジスタの値が入力値となる場合も同様に，複数レジスタの入力値をまとめて 1 つの入力エントリに記憶する．

AddrTbl は InTbl と同数のエントリを持ち，各エントリは 1 対 1 に対応する．AddrTbl の各エントリは入力値検索のために，次に参照すべきアドレス（next addr）を持つ．また，入力セットの終端エントリか否かを保持するフラグ（ec flag）を持ち，そのエントリが終端エントリである場合，出力を記憶している表である OutTbl のエントリを指すインデクス（OutTbl idx）も持つ．

OutTbl の各エントリは FLTbl のインデクス（FLTbl idx）に加えて，命令区間の出力先のアドレス（output addr），および出力値（output values）を持つ．また，出力セットの各エントリをリスト構造で管理するため，次に参照すべきエントリのインデクス（next idx）を持つ．

### 2.2 オーバヘッドフィルタ機構

自動メモ化プロセッサは，計算再利用を適用可能な命令区間の実行を省略することで高速化を図るが，その際には MemoTbl を検索するコスト，および入力が一致したエントリに対応する出力を MemoTbl からレジスタやキャッシュに書き戻すコストがオーバヘッドとして発生する．命令区間の中にはこれらのオーバヘッドが大きく，計算再利用を適用することで却って性能が悪化してしまうものも存在する．そのため，自動メモ化プロセッサはそのような命令区間に対する計算再利用を中止する機構を持つ．これをオーバヘッドフィルタと呼ぶ．FLTbl では，各命令区間に対し一定期間における再利用の成否状況をシフトレジスタ（図 1 中 hit hist.）を用いて記録し，それぞれの命令区間の計算再利用に対する適否の算出に用いる．シフトレジスタでは再利用 1 試行分の成功，失敗の結果が 1 ビットで記憶され，再利用に成功した場合は 1 が，再利用に失敗した場合は 0 がセットされる．また，新たに再利用テストの結果を記憶する際には，シフトレジスタを右に 1 ビットシフトしてから，左端に新たな 1 ビットの情報をセットする．シフトレジスタはこのように動作することで，ビット幅と同じ回数分の直近の過去の再利用の成否状況を記憶できる．

例えば，ある命令区間について，最近の一定回数 $T$ の再利用テストにおける再利用成功回数 $M$ は，幅 $T$ ビットの上記シフトレジスタから得られる．この値と，その命令区間に対応する FLTbl エントリに記憶されている過去の省略サイクル数 $S$ から，実際に削減できたサイクル数を

$$M \cdot (S - Ovh^R - Ovh^W) \qquad (1)$$

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当該命令区間の MemoTbl 検索オーバヘッド，および MemoTbl からキャッシュなどへの書き戻しオーバヘッドである．

また，再利用テストに失敗した場合でも，MemoTbl の検索オーバヘッドは存在する．このオーバヘッドは，

$$(T - M) \cdot Ovh^R \qquad (2)$$

として計算できる．

ここで，発生したオーバヘッド（2）よりも，削減できたサイクル数（1）が大きいような命令区間は，計算再利用の効果が得られると考えられる．式（1）から式（2）を引いたものを ***Gain*** とすると，

$$Gain = M \cdot (S - Ovh^W) - T \cdot Ovh^R \qquad (3)$$

となり，この $Gain$ が正値であれば，計算再利用の効果があると判断する．そして，そう判断された命令区間に対してのみ MemoTbl への登録および計算再利用の適用を行う．

**表 1** 再利用表，およびキャッシュのパラメータ

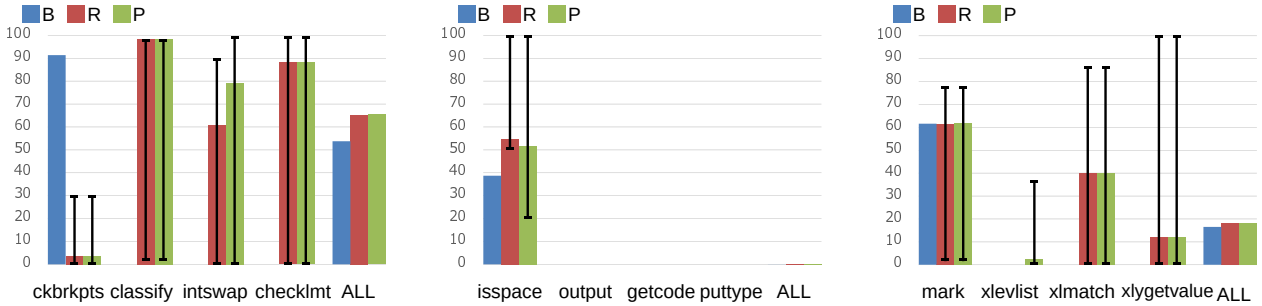
| | |
|---|---|
| Comparison (Register and CAM) | 9 cycles/32 Bytes |
| Comparison (Cache and CAM) | 10 cycles/32 Bytes |
| Write back (MemoTbl to Register) | 2 cycle/32 Bytes |
| Write back (MemoTbl to Cache) | 1 cycle/32 Bytes |
| D1 cache | 32 KBytes |
| line size | 32 Bytes |
| ways | 4 ways |
| latency | 2 cycles |
| miss penalty | 10 cycles |
| D2 cache | 2 MBytes |
| line size | 32 Bytes |
| ways | 4 ways |
| latency | 10 cycles |
| miss penalty | 100 cycles |
| Register windows | 4 sets |
| miss penalty | 20 cycles/set |

## 3. 復帰アドレス別の傾向調査

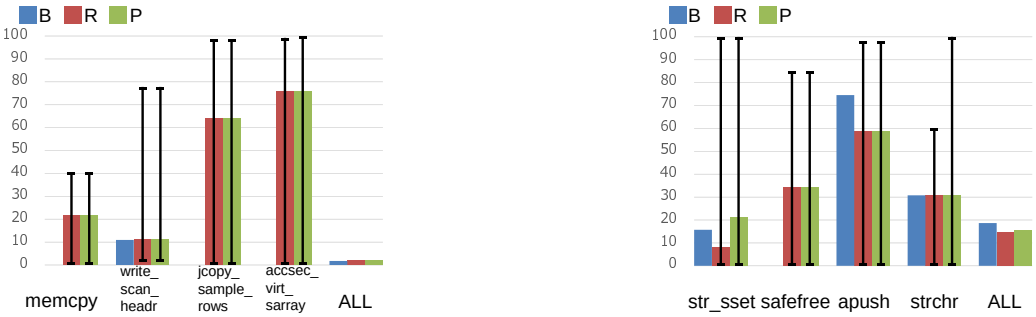
自動メモ化プロセッサは関数，およびループを再利用対象区間とみなすが，1 章でも述べたように，プログラム中の関数は，復帰アドレスによって入力の規則性や周期性などの特徴が異なると考えられる．そこで，入力の特徴の違いが関数の再利用に対して及ぼす影響を確認するために復帰アドレス別にオーバヘッドフィルタを管理した場合の再利用率について調査し，その調査結果について考察する．

### 3.1 評価環境

評価には，汎用ベンチマークプログラムである SPEC CPU95 と，計算再利用のための機構を実装した単命令発行の SPARC V8 シミュレータを用いた．評価に用いたパラメータを**表 1** に示す．キャッシュのパラメータや命令レイテンシは SPARC64-Ⅲ[4] を参考とした．評価項目として，関数全体の再利用率に加え，復帰アドレス別の再利用率を計測した．この際，計算再利用の適否を命令区間の復帰アドレスごとに判断するため，命令区間の復帰アドレスごとにオーバヘッドフィルタを適用した．また，入力の特徴の違いが復帰アドレスではなく呼出し元関数によって異なるか否かについても調査するために，呼出し元関数別の再利




(a) 124.m88ksim　(b) 129.compress　(c) 130.li




(d) 132.ijpeg　(e) 134.perl

**図 2**　関数・関数全体の再利用率およびその平均

**表 2**　(e) 134.perl における各関数の o_cycle と r_ovh，r_cycle

| 関数名 | | o_cycle | r_ovh | r_cycle |
|---|---|---|---|---|
| str_sset | (**B**) | 696424 | 989855 | -293431 |
| | (**R**) | 408563 | 463813 | -55250 |
| | (**P**) | 718299 | 455952 | 262347 |
| safefree | (**B**) | 0 | 662 | -662 |
| | (**R**) | 312704 | 223353 | 89351 |
| | (**P**) | 312704 | 223353 | 89351 |
| apush | (**B**) | 342225 | 325479 | 16746 |
| | (**R**) | 269505 | 246539 | 22966 |
| | (**P**) | 269505 | 246539 | 22966 |
| strchr | (**B**) | 756 | 386 | 370 |
| | (**R**) | 756 | 404 | 352 |
| | (**P**) | 756 | 440 | 316 |
| 全体 | (**B**) | 12485353 | 5183688 | 7301665 |
| | (**R**) | 14395471 | 4189166 | 10206305 |
| | (**P**) | 15009292 | 4158375 | 10850917 |

用率も計測した．なお，本評価では，再利用率の理論値を計測するために再利用表のサイズを無限大と仮定し，またループは再利用対象外とした．

### 3.2　評価結果

評価結果を**図 2** に示す．図 2 のグラフの太いバーは，各ベンチマークプログラムの関数全体，および再利用率に特徴がある関数の中から選出した 4 つの関数それぞれの平均再利用率を，細いバーは呼出し元関数ごと，または復帰アドレスごとの再利用率のうち，最大値から最小値までの値域を示している．なお，平均再利用率は同一関数を再利用した回数の総和を実行した回数の総和で割ることにより算出している．平均再利用率は 3 本のバーで示しており，

(**B**) 命令区間別に再利用の適否を判定した場合（青色）

(**R**) 呼出し元関数別に再利用の適否を判定した場合（赤色）

(**P**) 復帰アドレス別に再利用の適否を判定した場合（緑色）

にそれぞれ対応している．各プログラムの 4 つの関数は，計測した再利用率に基づいて選出した．具体的には，(**B**) の再利用率が最も高い関数，および (**B**) の再利用率が最も低い関数，(**P**) の再利用率が最も高い関数，再利用率に差がない関数を選出した．また**表 2** は，図 2 (e) 134.perl の選出した各関数における，再利用によって削減された実行サイクル数の合計（o_cycle），検索オーバヘッドと書き戻しオーバヘッドの合計（r_ovh），およびこれら二つの値の差 (r_cycle) を，再利用率の適否判定単位別に示している．評価の結果，計算再利用の適否判定単位の違いにより平均再利用率に差がある関数が存在することがわかる．次に，再利用率の値域に着目すると，(**R**) と (**P**) の結果が大きく異なっている場合があることがわかる．これらのことから呼出し元関数，または復帰アドレスにより，関数が過去の入力セットと同一の入力セットを伴って呼びだされる頻度が異なっていることがわかる．

### 3.3　復帰アドレスの違いがもたらす効果

図 2 (e) の str_sset 関数のグラフと表 2 の str_sset 関数の項目のそれぞれ (**B**) と (**P**) に着目すると，(**B**) と比較して (**P**) は，再利用率と r_cycle の値が大きいことがわかる．ここで，str_sset 関数の実行回数，検索回数，再利用回数

表 3 （e）134.perl の str_sset 関数における実行回数等

| | 復帰アドレスまたは呼出し元関数 | 実行 | 検索 | 再利用 |
|---|---|---|---|---|
| (**B**) | | 64493 | 29607 | 10144 |
| (**R**) | 1d25c | 5919 | 4129 | 2065 |
| | 245a0 | 750 | 53 | 0 |
| | 265c0 | 18181 | 6269 | 1710 |
| | 27eb8 | 1 | 1 | 0 |
| | 29fa4 | 2019 | 2019 | 2017 |
| | 2e9e0 | 13329 | 55 | 0 |
| | 40708 | 1 | 1 | 0 |
| | 41bf4 | 24291 | 43 | 0 |
| | 420ec | 3 | 3 | 1 |
| | 合計 | 64494 | 12573 | 5793 |
| (**P**) | 1d284 | 5919 | 4129 | 2065 |
| | 2469c | 750 | 53 | 0 |
| | 2693c | 12064 | 21 | 0 |
| | 269cc | 2041 | 30 | 0 |
| | 26a00 | 4076 | 4076 | 4074 |
| | 27f40 | 1 | 1 | 0 |
| | 2a71c | 2019 | 2019 | 2017 |
| | 2eb4c | 6120 | 5661 | 5275 |
| | 2efdc | 7213 | 55 | 0 |
| | 40734 | 1 | 1 | 0 |
| | 41c10 | 24291 | 43 | 0 |
| | 47094 | 3 | 3 | 1 |
| | 合計 | 64498 | 16092 | 13432 |

を**表 3** に示す．なお，(**B**) と (**R**) では str_sset 関数の呼出し元関数，またはさらに上位の関数を再利用している回数が (**P**) より多いため，(**P**) よりも実行回数の合計が少なくなっている．(**B**) における検索回数が約 3 万回であることから，プログラム開始後約 3 万回呼び出され，その時点で再利用の効果がないと判定されたことがわかる．(**B**) は，命令区間ごとに再利用の適否を判定するため，一度計算再利用に不適と判定されると，その関数に対する全ての呼出しが再利用の適用対象から除外される．一方 (**P**) では，復帰アドレスごとに計算再利用の適否を判定するため，ある復帰アドレスからの呼出しが再利用に不適であると判定された場合でも，他の復帰アドレスからの呼出しには再利用を適用できる．アドレス 1d284，26a00，2a71c，2eb4c のように再利用成功率が高いアドレスから呼びだされた場合にのみ計算再利用を適用したことで，(**B**) では再利用テストが行われなかった呼出しに対しても再利用テストを行え，(**P**) では (**B**) と比較して，o_cycle と再利用率とが大きくなったと考えられる．一方で，再利用成功率の低いアドレスから呼びだされる場合を早期に再利用対象から除外でき，(**B**) と比較して r_ovh が小さくなったと考えられる．結果として，削減サイクル数 r_cycle が向上した．続いて，同じく図 2（e）の str_sset 関数のグラフと表 2 の str_sset 関数の項目の (**R**) と (**P**) に着目すると，(**R**) は再利用率，および o_cycle の値が (**P**) よりも小さいことがわかる．ここで，表 3 の (**P**) に着目すると，復帰アドレスが 2eb4c の場合には再利用に成功している一方で，復帰アドレスが 2efdc の場合には早期に再利用テストが行われなくなったことがわかる．また，表 3 の (**P**) と (**R**) に着目すると，呼出し元関数の先頭アドレスが 2e9e0 の場合と復帰アドレスが 2efdc の場合とで検索回数が同じであることがわかる．これらのことから，復帰アドレス 2eb4c と 2efdc は先頭アドレスが 2e9e0 である親関数内に含まれており，それらのうち，先に 2efdc から str_sset 関数が呼び出され，繰り返し再利用に失敗したことで，(**R**) では，アドレス 2eb4c と 2efdc からの呼出し全てが再利用対象から除外されたと考えられる．一方 (**P**) では，アドレス 2efdc から呼び出された場合については再利用適用対象となり，かつその再利用テストに成功したため，(**R**) は (**P**) と比較して再利用率，および o_cycle が抑えられてしまったと考えられる．

さらに，同じく図 2（e）の str_sset 関数のグラフと表 2 の str_sset 関数の項目の (**B**) と (**R**) に着目すると，(**R**) では，(**B**) と比較して再利用率が小さくなっていることがわかる．これは，(**R**) では呼出し元関数ごとに計算再利用の適否を判定するため，再利用成功率の高いアドレス 2eb4c から呼びだされた場合に対しても，計算再利用を中止してしまったためだと考えられる．しかし同時に，検索回数が少なくなったことで，再利用テストによる r_ovh が小さくなったため，結果として r_cycle は大きくなった．

これらのことから，呼出し元ごとの再利用成功率の違いを自動メモ化プロセッサの高速化に活用できると考えられるが，図 2（e）の strchr のように，呼出し元別に管理した場合でも再利用率が変化しない関数では，復帰アドレスや呼出し元関数の数に比例して，（**B**）よりも r_ovh が増加してしまう．これは，オーバヘッドフィルタを呼出し元別に適用するようにしたことで，呼出し元それぞれに対して計算再利用が不適であると判定されるまでに必要となる再利用テストの失敗回数が，総計で（**B**）よりも多くなってしまうためである．そのため，再利用成功率が復帰アドレスや呼出し元関数によって変化しない関数を多く含む場合には，呼出し元ごとに計算再利用の適否を判定することでかえって性能が低下してしまうと考えられる．

## 4. 性能向上に向けた手法の検討

従来の自動メモ化プロセッサでは命令区間ごとに計算再利用の適否を判定するため，復帰アドレスごとの再利用成功率は考慮されていない．そのため，前節で述べたように無駄な r_ovh が発生してしまう場合や，再利用成功率が高い復帰アドレスからの呼び出し時にも計算再利用の効果がないと判定してしまう場合が存在する．そのような場合には，復帰アドレスごとに計算再利用の適否を判定することが高速化につながると考えられる．その場合，関数の復帰

表 4　各ベンチマークにおける
区別して保持する必要のある再利用ヒット履歴の数

| ベンチマーク | (**B**) | (**R**) | (**P**) |
|---|---|---|---|
| 124.m88ksim | 84 | 121 | 173 |
| 129.compress | 22 | 27 | 39 |
| 130.li | 104 | 289 | 349 |
| 132.ijpeg | 149 | 255 | 356 |
| 134.perl | 154 | 403 | 629 |

アドレスごとに再利用ヒット履歴を保持する必要があるため，記憶すべき情報量が増加し，必要な再利用表サイズの増加が懸念される．一方，呼出し元関数ごとに計算再利用の適否を判定する場合は，復帰アドレスごとに判定する場合と比較して，記憶する情報量を低く抑えられる．

再利用表のサイズは有限であるため，記憶する必要のある情報量が多くなるほど，エントリ溢れに伴って再利用ヒット率が低下する可能性が高く，適否判定を復帰アドレスごとに行うことで，呼出し元関数ごとの場合よりも性能低下する可能性がある．ここで，各ベンチマークプログラムを実行した際に，再利用の適否判定の対象となる単位の数，すなわち，区別して保持する必要のある再利用ヒット履歴の数を**表 4** に示す．表 4 から，(**P**) では全ての関数と復帰アドレスの組の最大数が 629 であるのに対し，(**R**) では全ての関数とその呼出し元関数の組の最大数が 403 であることがわかる．

以降では，再利用表の有効活用を重視し，呼出し元関数ごとに計算再利用の適否を判定する方法をベースに検討する．呼出し元関数ごとの判定のためには，各関数がどの関数から呼び出されたのかを管理する必要がある．よって，関数を登録するための表である FLTbl に，呼出し元関数のアドレスを格納するフィールドを追加する．InTbl の各エントリは FLTbl の行番号 Index を持つことで，どの関数の入力値を記憶しているかを識別する．そのため，単純に FLTbl に呼出し元関数のアドレスを格納するフィールドを追加し，FLTbl の各エントリを関数とその呼出し元関数の組と対応させてしまうと，InTbl は記憶している入力を関数だけでなく呼出し元関数とも関連付けてしまう．これにより，再利用テストの際には，同一の関数から呼び出された場合の過去の入力としか比較できなくなる．そこで FLTbl を，1 エントリを 1 関数と対応させる表（FLTbl1）と，呼出し元関数ごとに再利用ヒット履歴を保持するための表（FLTbl2）に階層化する．そして，InTbl は FLTbl1 の行番号 Index を持つことで，関数の入力をその呼出し元に関わらず，再利用テスト時の比較対象とすることができる．なお，FLTbl1 および FLTbl2 は，FLTbl 同様 RAM で実装することを想定している．ここで，FLTbl1 および FLTbl2 の構成例を**図 3** に示す．FLTbl1 は，その行番号（FL1 Index）を各関数の識別番号とする．また，各行には従来の FLTbl が持つメモ化のためのフィールドに加え，

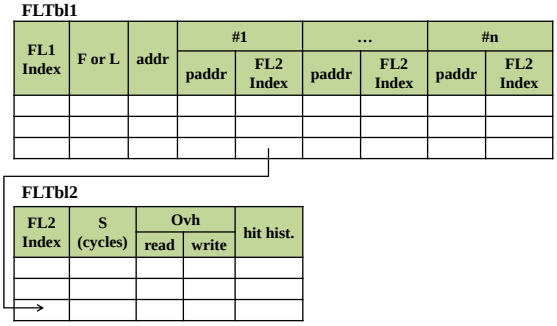



図 3　階層化された FLTbl の構成例

FLTbl2 のエントリと対応する，呼出し元関数を識別するための呼出し元関数の先頭アドレス（paddr）と，FLTbl2 の対応するエントリへのインデクス（FLTbl2 Index）の組を複数記憶するためのフィールドを持つ．FLTbl2 の各エントリは，呼出し元関数の先頭アドレスと関数の先頭アドレスの組に対応する．また，FLTbl2 は従来の FLTbl が持つオーバヘッドフィルタのためのフィールドを持つ．これを利用することで，呼出し元関数ごとに計算再利用の適否を判定する．

## 5. おわりに

本稿では，自動メモ化プロセッサが再利用の対象区間と見なす関数は，復帰アドレスによって入力の規則性や周期性などの特徴が異なるという予測に基づき，復帰アドレス別の再利用率について調査した．結果，復帰アドレスによって再利用率が大きく異なることが確認できた．また，復帰アドレスごとに計算再利用の適否を判定し，再利用成功率が高いと判定される復帰アドレスからの呼出時に限って再利用テストを行うことでより多くの実行サイクル数が削減できることを確認した．一方で，再利用成功率が低い復帰アドレスからの呼出しを計算再利用の対象から早期に除外することで，再利用テスト失敗に伴う再利用表検索オーバヘッドを抑制できることも確認した．今後は調査結果についてより深く考察したうえで，自動メモ化プロセッサのさらなる性能向上に向けた改良案について検討していきたいと考えている．

### 参考文献

[1] Tsumura, T., Suzuki, I., Ikeuchi, Y., Matsuo, H., Nakashima, H. and Nakashima, Y.: Design and Evaluation of an Auto-Memoization Processor, *Proc. Parallel and Distributed Computing and Networks*, pp. 245–250 (2007).
[2] Norvig, P.: *Paradigms of Artificial Intelligence Programming*, Morgan Kaufmann (1992).
[3] Huang, J. and Lilja, D. J.: Exploiting Basic Block Value Locality with Block Reuse, *Proc. 5th Int'l Symp. on High-Performance Computer Architecture (HPCA-5)*, pp. 106–114 (1999).
[4] HAL Computer Systems/Fujitsu: *SPARC64-III User's Guide* (1998).