卒業研究論文

自動メモ化プロセッサにおける
オーバーヘッドフィルタの改良

指導教員　　松尾 啓志 教授
　　津邑 公暁 准教授

名古屋工業大学 工学部 情報工学科
平成 19 年度入学　19115033 番

小田 遼亮

平成 23 年 2 月 8 日

自動メモ化プロセッサにおけるオーバーヘッドフィルタの改良

小田 遼亮

内容梗概

CPU における消費電力や発熱量の増大により，動作周波数の向上が難しくなってきている現在，これらの問題点の解決策としてマルチコアプロセッサが提案されている．マルチコアプロセッサとは，1 つの CPU に複数コアを搭載した，並列性に基づく高速化手法を用いたプロセッサである．一方，これとは全く異なる概念である計算再利用技術に基づく高速化手法を用いた自動メモ化プロセッサが提案されている．計算再利用とは，一度実行した事がある命令区間の実行自体を省略する事により高速化を図る手法である．自動メモ化プロセッサは，並列事前実行と呼ばれる投機的な実行を行うコアを搭載する事で，更なる高速化を図っている．

自動メモ化プロセッサは，再利用対象区間の実行時に入出力の情報を再利用表と呼ばれる表へ登録する．そして，再度同じ区間を実行する時，再利用表へ登録しておいた入力と現在の入力の一致比較を行う．入力が一致した場合，再利用表へ登録しておいた出力を書き戻す事で実行を省略する．この入力一致比較や書き戻しはオーバーヘッドとなり，このオーバーヘッドが大きいと再利用を適用する事でかえって実行時間が増加してしまう場合がある．そこで，自動メモ化プロセッサは，再利用による効果が得られないと判定した区間の再利用表への登録を中止する，オーバーヘッドフィルタと呼ばれる機構を備えている．しかし，現在この判定の基準として過去の再利用の成功履歴は考慮されていない．再利用表の容量は有限であるため，このような再利用に成功する確率が低い区間の情報が再利用表へ登録されると，再利用に成功する確率が高い区間が再利用表から追い出される可能性が高まり，実行速度の低下に繋がる．

本研究の目標は，再利用表への不必要なエントリの登録を抑制することで再用表を有効活用し再利用効率を上げることで，自動メモ化プロセッサのさらなる高速化を図ることである．そこで，オーバーヘッドフィルタにおいて過去の再利用の成功を考慮した登録の中止を行う手法を提案する．

提案手法の有効性を検証するため，従来の自動メモ化プロセッサに提案手法のモデルを実装し，SPEC CPU95 ベンチマークでシミュレーションによる評価を行った．その結果，メモ化を行わない場合に対し，従来手法では最大で 24.0%，平均で 5.1%のサイクル数の削減だったところ，提案手法では最大で 30.5%，平均で 8.3%まで向上することが確認できた．

自動メモ化プロセッサにおけるオーバーヘッドフィルタの改良

目次

1 はじめに

これまで，配線遅延の相対的な増大に伴い，高いクロック周波数だけではプロセッサの性能向上が難しくなってきた．それにより，スーパースカラや SIMD 等の命令レベル並列性 (Instruction-Level Parallelism: ILP) に基づく高速化手法が注目された．また，クロック周波数を上げる事により電力消費や発熱量は急激に増大する．そこで，スレッドレベル並列性 (Thread-Level Parallelism: TLP) に基づいた高速化手法である，マルチコアプロセッサ技術が注目されている．マルチコアプロセッサでは，クロック周波数を低く抑えたプロセッサを複数持つことで，電力消費や発熱量の増大を抑えつつ処理の高速化を図る．これらの高速化手法は，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．

一方で，プログラムにおける値の局所性に着目した計算再利用と呼ばれる手法が存在する．並列化が，処理を同時実行する事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事により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．このように，計算再利用は並列化とは直交する概念であるため，並列化が有効でないプログラムに対しても高速化を図る事ができる可能性があり，並列化とも併用可能であるという利点が存在する．

計算再利用には，ハードウェアによるものとソフトウェアによるもの，またその両方によるもの等，様々なものが存在する．専用のハードウェアを用いることによりバイナリの変更無しで，既存のプログラムを実行できる区間再利用のモデルに自動メモ化プロセッサ [1] が存在する．自動メモ化プロセッサは実行した関数やループの入出力を再利用表と呼ばれる表に記憶しておく．そして，再び同じ関数やループが実行される際にその入力値と過去の入力値とを比較し，一致すれば過去の出力を利用することで実行を省略する．しかし，この比較や出力の利用にはオーバーヘッドが生じるため，再利用を適用する事によりかえって性能が悪化してしまう場合がある．そこで自動メモ化プロセッサは，性能の悪化を招くような再利用を中止するために，オーバーヘッドフィルタと呼ばれる機構を備えている．本研究では，このオーバーヘッドフィルタにおいて，過去の再利用の成功を考慮し，登録の中止とその再開を行う手法を提案する．

以下，2 章では本研究が扱う自動メモ化プロセッサの動作モデルを述べる．3 章ではオーバーヘッド評価機構の改良を提案し，4 章でその実装について説明する．5 章で本提案手法の評価を行い，最後の 6 章で結論を述べる．

2　研究背景

本章では，本研究で取り扱う自動メモ化プロセッサについて，その高速化の方針，アーキテクチャの構成，動作を概説する．

2.1　自動メモ化プロセッサ

メモ化 (Memoization)[2] とは，関数やループといったプログラム中の命令区間に対してその入出力を保存しておく事で高速化を図る手法である．このメモ化による高速化をハードウェアを用いて動的に行う事により，既存のバイナリを変更する事無く実行可能なプロセッサとして自動メモ化プロセッサが提案されている．

2.1.1　再利用機構における入出力

計算再利用では，入力が一致すれば出力が必ず正しい事を保証しなければならない．自動メモ化プロセッサでは関数とループを再利用対象区間としているので，これらの区間における入出力が何かを明確にする必要がある．関数の入力になりうる値とは，大域変数，及びその関数を呼び出す関数の局所変数である．また出力は大域変数，及び返り値である．これに加えてループでは，そのループを含む関数の局所変数が使用される可能性があるため，その局所変数も入出力となる．

2.1.2　再利用機構の構成

自動メモ化プロセッサのアーキテクチャを**図 1** に示す．自動メモ化プロセッサは，コアの内部に一般的な CPU コアが持つ ALU，レジスタ (Reg)，1 次データキャッシュ (D$1) を持ち，コアの外部に共有の 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が独自に持つ機構として，コアの内部に MemoBuf と再利用機構を管理するための構造 (Reuse_System) を持ち，コアの外部に MemoTbl を持つ．MemoTbl とは命令区間及びその入出力を記憶するための表であり，計算再利用を行うために必要となる．また，コアが命令区間の入出力を MemoTbl に登録する時，サイズの大きい MemoTbl に対して毎回参照を行うとオーバーヘッドが大きくなってしまう．そこで，このオーバーヘッドを軽減するため，作業用の小さなバッファである MemoBuf をコアの内部に設けている．MemoBuf と MemoTbl の詳細な構成を**図 2** に示す．

まず MemoBuf について説明する．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，該当する命令区間を記憶する RFindex，命令区間の実行開始時のスタックポインタ SP，関数の戻り値とループの終端アドレス

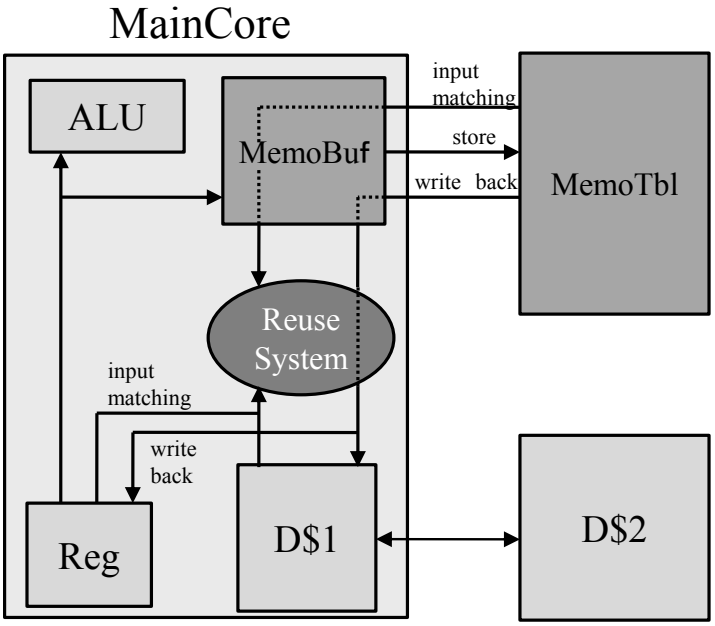

図 1: 自動メモ化プロセッサのアーキテクチャ

を記憶する FOfs，命令区間の入力セット Read，及び出力セット Write のフィールドからなる．命令区間の実行中に検出された入出力は，命令区間の実行を続けながら Read フィールドと Write フィールドに記憶されていく．そして，各命令区間の実行終了時に MemoBuf に記憶しておいた再利用エントリを MemoTbl 本体に一括で登録する．

ここで，MemoBuf に複数のエントリが存在するのは，再利用コアが実行中に呼び出した関数やループのネスト構造を保持するためである．MemoBuf のどのエントリを使用しているかを判断するためにポインタ memobuf_top を用いる．MemoBuf はスタックのように扱われ，階層の低い方から順に番号が振られている．エントリは番号の値の小さい方から順に使用され，関数呼び出し (call) や多重ループによってネストが増加すると，それに対応して memobuf_top がインクリメントされる．逆に関数から戻った場合 (return) や内側のループの実行終了時に，memobuf_top をデクリメントし階層を下げる．このようにして，MemoBuf は，コアが現在実行している関数やループのネスト構造を保持する．また，MemoBuf のエントリ数は有限であるため，一度に登録可能な命令区間の数に限りがある．そこで，登録可能な数よりも多くネストを検出した場合には，外側の命令区間から順次登録を中止し，より内側の命令区間を登録対象に加えることにしている．

次に，MemoTbl の各構成要素について説明する．まず，RF は命令区間を記憶する表であり，関数とループを判別するフラグ (type) を持ち，命令区間の開始アドレス

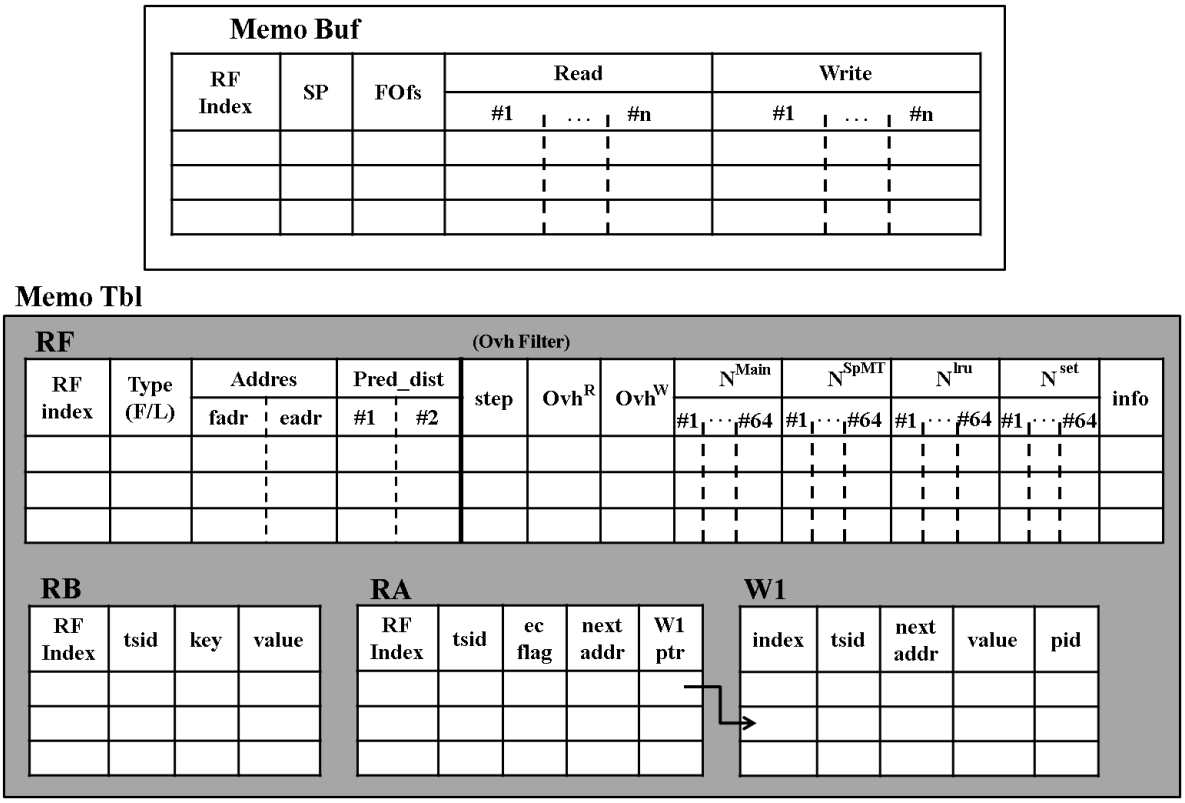

図 2: MemoBuf と MemoTbl の詳細な構造

(fadr) と終端アドレス (eadr) を記憶する．ただし，関数の場合は終端アドレスの代わりに戻りアドレスを記憶している．また，該当する命令区間を記憶する RFindex を持つ．RFindex を RF が一括管理し，他の各要素が RFindex を持つことで，命令区間ごとのエントリの管理ができる．更に，後述する並列事前実行やオーバーヘッドフィルタにおいて使用される値も持っている．RB は命令区間の入力値を記憶するための表であり，次入力エントリを得るためのインデクス (key) と入力値 (value) を記憶する．また，これらの値から次入力となるエントリを検索し特定する必要があるため，連想検索が可能な CAM(Content Addresable Memory) で実装されると仮定している．更に，CAM で実装する事で入力一致比較時の MemoTbl の検索オーバーヘッドを小さく抑えることができる．RA は入力アドレスのセットを記憶する表であり，次に参照すべき入力アドレス (nextaddr) を記憶する．RB と RA は同数のエントリを持ち，各エントリは 1 対 1 に対応する．さらに，RA はそのエントリが入力値セットの終端であることを示すフラグ (ecflag) を持つ．W1 は出力値を記憶する表であり，次に参照すべき出力アドレス (nextaddr) と出力値 (value) を記憶する．それに加えて，どのコアが登録を

行ったかという情報 (pid) も持つ．また，RA の終端エントリがその出力を記憶している W1 のエントリを指すポインタ (W1ptr) を持つ．MemoTbl と入力一致比較を行い，入力が完全に一致した場合は W1 のエントリを指すポインタ (W1ptr) により当該入力に対応する出力を読み出し，レジスタやメモリへ書き戻すことで，命令区間の実行を省略することができる．これらに加え RB，RA，W1 は次節で述べるパージのためにそれぞれタイムスタンプ (tsid) を持つ．

2.2 パージアルゴリズム

MemoTbl の容量は有限であるため，適宜エントリを削除する必要がある．そこで，自動メモ化プロセッサでは，Least Recently Used(LRU) に基づく手法と，明示的に特定のエントリを削除する手法の 2 つを用いている．

2.2.1 TSID パージ

TSID パージは LRU に基づく追い出し手法である．このような追い出しを行うために，自動メモ化プロセッサはリングカウンタによる時刻管理を行っている．このカウンタは RB，RA，W1 へ一定数の登録があるごとにインクリメントされる．RB，RA，W1 の各エントリは図 2 に示すように，tsid と呼ばれるタイムスタンプを保持しており，登録が行われた時や再利用が行われた時に，現在のリングカウンタの値をここに保存する．リングカウンタが更新される度に，リングカウンタが更新された後と同一の tsid を持つエントリを RB，RA，W1 から削除し，その後リングカウンタを更新する．リングカウンタの更新は，メインコアが MemoTbl への登録を行うタイミングで行われる．

2.2.2 RFID パージ

RFID パージは，再利用エントリが枯渇した時点で空きエントリを確保するために，特定のエントリを明示的に指定して削除する追い出し手法である．エントリの明示的な指定には命令区間を記憶している RFindex を用い，指定した RFindex を持つエントリを MemoTbl から全て削除する．再利用エントリの枯渇には，RF が枯渇する場合と RB，RA，W1 のいずれかが枯渇する場合が存在する．RF は TSID パージによる追い出しを行わないため，一定量の登録があるとエントリが枯渇する．この場合空きエントリを確保するために，2.4 節で述べるオーバーヘッドフィルタによりパージする区間を選択し，その区間に対し RFID パージを行う．その後，確保されたエントリへ登録を行う．一方，並列事前実行コアによる登録が多く発生した場合，メインコアによる TSID パージが起こらず，RB，RA，W1 のいずれかが枯渇する事がある．この場合空

きエントリを確保するために，現在登録中のエントリと同一の RFindex を持つエントリを RB，RA，W1 からパージする．RB，RA，W1 が枯渇した場合の追い出しには，メインコアによる登録エントリを保護する目的がある．ここで RFID パージは，再利用に成功する確率が高い区間や使用頻度の高い区間を追い出してしまうことがあり，性能を悪化させる可能性がある．

2.3 並列事前実行

自動メモ化プロセッサでは，関数とループを再利用対象区間とする．これらの区間を再利用する為には，過去に全く同一の入力セットにより実行している必要がある．よって，入力の一つとしてイタレータ変数を持つようなループイタレーションでは，過去に同一の入力セットによる実行がなく，再利用による効果が全く得られない．そこで，過去の命令区間の入力に基づき，同一命令区間を将来実行する際に用いられる入力値を予測し，該当区間をメインコアとは別のコアで投機的に実行しておく並列事前実行が提案されている．このコアは並列事前実行コアと呼ばれ，前もって実行しておいた結果を MemoTbl へと登録しておく．もし入力値予測が正しかったならば，メインコアで実行しようとする区間が既に MemoTbl に登録されており，実行を省略する事ができる．また，入力値予測が誤っていた場合，MemoTbl の検索コストはかかるものの，投機実行に起因するオーバーヘッドは発生しない．

図 3 に並列事前実行機構を備えた自動メモ化プロセッサのアーキテクチャを示す．並列事前実行コアは複数備える事ができる．また，ALU，レジスタ (Reg)，1 次データキャッシュ(D$1)，MemoBuf は各コアが独立で持ち，MemoTbl と 2 次データキャッシュ (D$2) 及び主記憶は全てのコアで共有されるものとする．

MemoBuf をコアごとに持っているので，各命令区間の実行終了時にそれぞれのコアは入出力情報を MemoBuf から MemoTbl へ独立して登録することができる．また，MemoTbl は共有されているので，メインコアは自身や並列事前実行コアによって登録された MemoTbl のエントリを用いて再利用を行うことができる．

並列事前実行を行うためには，過去の再利用エントリの登録情報から，将来の入力値を予測して，並列事前実行コアへと渡す必要がある．そこで，入力を予測して並列事前実行コアに入力として与えるための小さなハードウェア (Input Pred) を MemoTbl に設けている．また，入力の予測にはストライド予測 [3] を用いており，最近の 2 組の入力の差分であるストライドに基づいて予測を行う．そのため，RF は各命令区間に対してストライド予測に用いるこの 2 組の入力 (Pred_dist) を保持している．

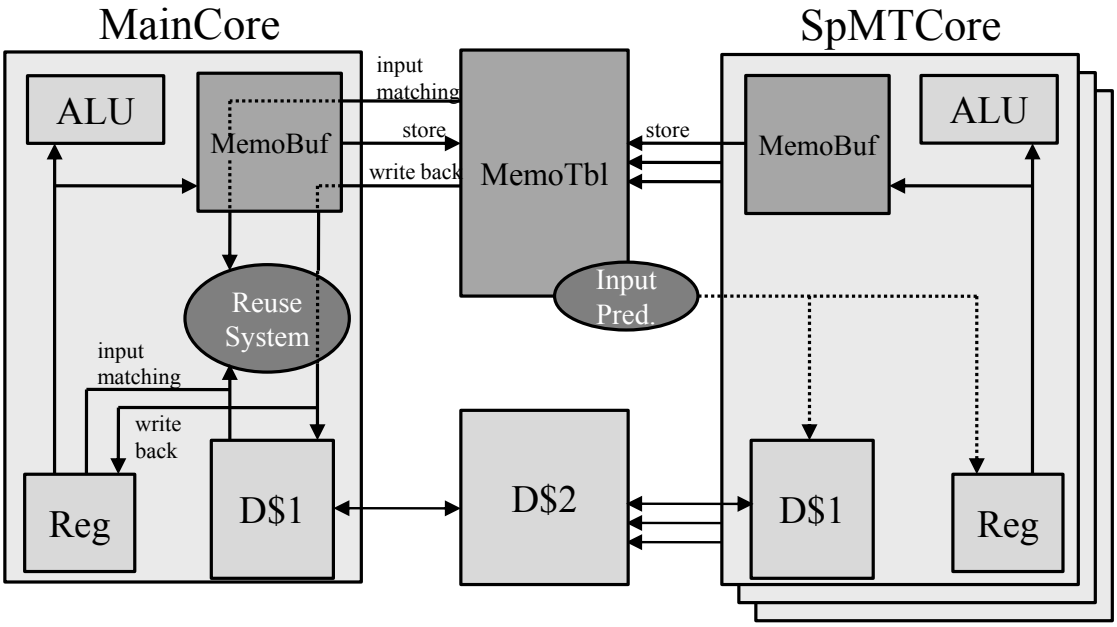

図 3: 並列事前実行機構を備えた自動メモ化プロセッサのアーキテクチャ

それでは，並列事前実行コアを用いた場合の動作を**図 4** を用いて説明する．例として，並列事前実行コアを 3 基とし，過去の入力履歴からストライドが 1 であると計算されているとする．ここで，ある命令区間に対してメインコアが入力値 4 で通常実行しているとすると，それに平行して並列事前実行コアではストライド予測を用いて入力値 5，6，7 でそれぞれ実行を行う．

効率的な並列事前実行の実現のためには，図 4 の (a) に示すように，メインコアが入力値 4 の実行を終えて入力値 5，6，7 の実行に移ろうとした時には既に並列事前実行コアでのそれぞれの入力に対する処理が終えられていなければならない．各入力に対する操作は通常同程度の時間で処理が終了すると考えられるが，分岐の変化やキャッシュミス等により，処理が遅延してしまう場合がある．並列事前実行の処理に遅延が発生すると図 4 の (b) に示すように，再利用を行う事ができず，メインコアが並列事前実行コアと同一の入力値を用いた実行をしてしまう．この問題は，並列事前実行を早めに開始する事により，回避可能である．そこで，自動メモ化プロセッサでは，現在メインコアで実行中の命令よりもある程度先まで並列事前実行コアへ入力の割り当てを行っている．

また，メインコアと並列事前実行コアでは 2 次データキャッシュや主記憶を全てのコアで共有している．ここで，各コアが主記憶に書き込む内容は異なっており，主記憶

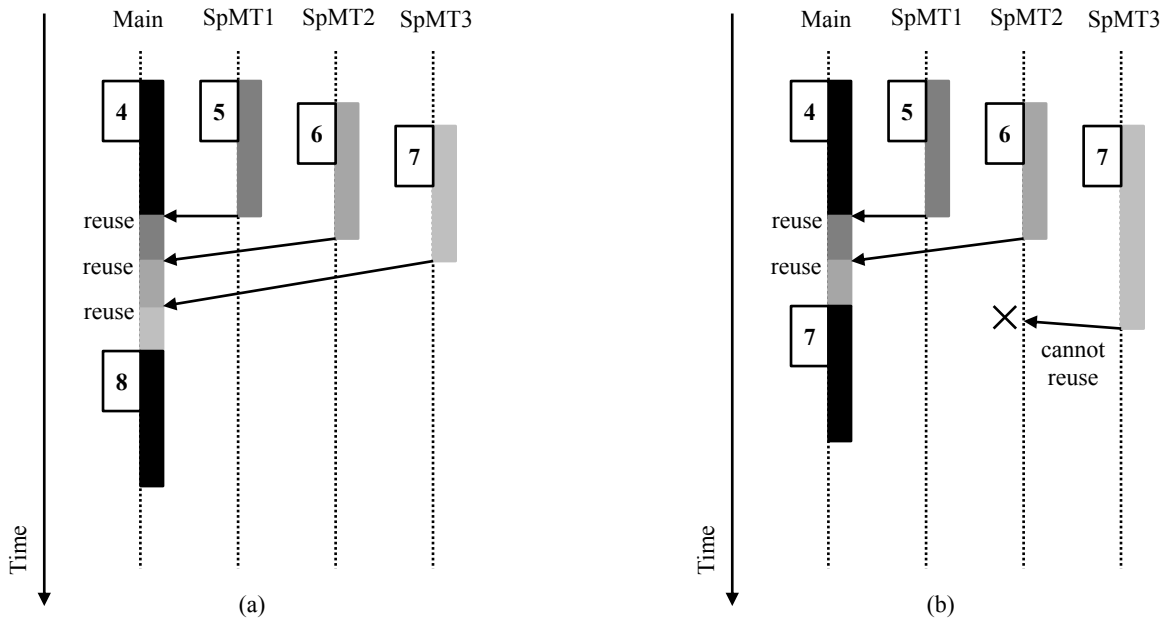

図 4: 並列事前実行の流れ

一貫性を保つ必要がある．そこで，並列事前実行コアは，MemoTbl への登録対象となるような主記憶参照には，各コアごとに設けられた MemoBuf を使用する．また，その他の局所的な参照には各コアごとに設けた局所メモリを用いる事により，キャッシュ及び主記憶への書き込みを行わないようにしている．ここで，メインコアが主記憶に書き込みを行った場合には，対応する並列事前実行コアのキャッシュラインが無効化される．

2.4 再利用オーバーヘッドとオーバーヘッド評価機構

自動メモ化プロセッサは，再利用対象区間の実行時にその入出力を MemoTbl に登録する．そして，再度同じ区間を実行する時，現在の入力値と記憶されている入力値が一致するエントリを MemoTbl から検索する．検索が成功した場合，MemoTbl へ登録しておいた出力をレジスタやキャッシュに書き戻す事で実行を省略している．この検索や書き戻しにはオーバーヘッドが生じる．検索時に生じるオーバーヘッドは，再利用の成功・失敗に関わらず発生する．また，書き戻し時に生じるオーバーヘッドは，再利用が成功した際に発生する．これらのオーバーヘッドをあわせて再利用オーバーヘッドと呼ぶ．命令区間によっては，再利用オーバーヘッドが大きく，計算再利用を行わずに実際に命令を実行した方が早く実行を終えることができる場合も存在する．その場合，計算再利用により性能が悪化するだけでなく，不必要な入出力を MemoTbl に

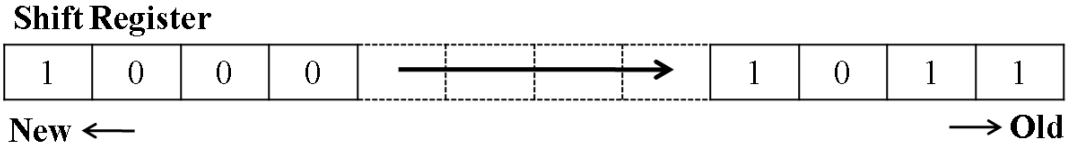

図 5: シフトレジスタの構造

登録していることになり，MemoTbl が有効活用されなくなる．そこで，自動メモ化プロセッサでは，MemoTbl への無駄なアクセスを抑制する再利用オーバーヘッド評価機構を備えている．再利用オーバーヘッド評価機構では，再利用オーバーヘッドと，計算再利用により高速化できる実行サイクル数とを見積もり，計算再利用により効果が得られると判断した命令区間に対してのみ再利用を適用する．具体的には，命令区間の再利用により削減できるサイクル数と，その再利用に必要となるオーバーヘッドについて概算を行う小さなハードウェアを MemoTbl に付加する．この機構を**オーバーヘッドフィルタ**と呼ぶ.

オーバーヘッドフィルタは命令区間ごとの判定を行うために，図 2 で示したように RF の各エントリにこれらのサイクル数を保持している．また，再利用に成功する確率等の過去の履歴を用いる事により，更にサイクル数の増加を抑制できると考えられる．そこで，動的に変化する過去の履歴を把握するために，一定期間における登録及び再利用の状況をシフトレジスタを用いて記憶している．このシフトレジスタの構造を**図 5** に示す．シフトレジスタでは履歴として記憶する情報を 1 ビットで表す．新たに情報を履歴として記憶する時には，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．これにより，左端に最新の情報を記憶し，右に行く程古い情報を記憶する．シフトレジスタはこのように動作し，ビット数と同じ回数分の状況を記憶できる．既存のオーバーヘッドフィルタでは 64 ビットのシフトレジスタを用いて，過去 64 回分の状況を記憶しており，シフトレジスタの値を元に再利用オーバーヘッドの算出を行う．また，1 ビットの info フラグを用いる事で，過去に再利用した事があるかどうかを記憶している．

それでは，オーバーヘッドフィルタの動作について説明する．オーバーヘッドフィルタは MemoTbl への登録，及び MemoTbl の検索を中止するかどうかの判定を行う．またそれらに加え，並列事前実行を行う区間の選択と，パージする RF エントリの選択も行う．

MemoTbl への登録時，命令区間の再利用により削減できるサイクル数を C，その再

利用の検索時に発生するオーバーヘッドを Ovh^R，書き戻し時に発生するオーバーヘッドを Ovh^W とする．これらの値から削減サイクル数を

$$C - Ovh^R - Ovh^W \tag{1}$$

として計算し，この削減予測サイクル数 (1) が正の時にのみ登録を行う．このように，登録時には過去の再利用における RF エントリの使用頻度等の履歴は用いず，サイクル数のみを用いて判定を行う．ここで，登録による効果が得られないと判断された区間は，RFID パージを行わないようにすることで，今後その命令区間の再利用を行わない．一方，MemoTbl の検索を中止する判定基準には再利用に成功する確率が考慮される．ここで，再利用に成功する確率を考慮するため，図 2 に示した N^{Main} と N^{SpMT} を用いる．N^{Main} は過去 N 回の検索が起こる間の，メインコアが登録したエントリによる，再利用の成功状況を記憶しているシフトレジスタである．N^{Main} では，検索が行われた時，メインコアが登録したエントリにより再利用が成功すると 1 を記憶し，そうでなければ 0 を記憶する事で再利用の成功状況を記憶している．この N^{Main} より算出した，過去 N 回の検索の間にメインコアが登録したエントリにより再利用が成功した回数を M とする．同様に，N^{SpMT} は並列事前実行コアが登録したエントリよる再利用の成功状況を記憶しているシフトレジスタであり，この N^{SpMT} より算出した，過去 N 回の検索の間に並列事前実行コアが登録したエントリにより再利用が成功した回数を S とする．これらの値と，(1) で示した削減予測サイクル数より，再利用が行われた時に削減できるサイクル数を

$$(M + S) \times (C - Ovh^R - Ovh^W) \tag{2}$$

として計算する．また，再利用が行われなかった場合にも検索オーバーヘッドは発生するので，その再利用が行われなかった場合のオーバーヘッドを

$$(N - M - S) \times Ovh^R \tag{3}$$

として計算する．ここで発生したオーバーヘッド (3) よりも，削減されたサイクル数 (2) の方が小さい場合には，再利用の適用によりサイクル数が増加してしまっていると判断できる．そのような場合に，MemoTbl の検索を中止する事によって速度の低下を防ぐ．ここで，再利用に成功する確率が低い期間と再利用に成功する確率が高い期間があるような命令区間は，検索を中止する事によって，再利用に成功する確率が高い期間にも再利用が適用できなくなってしまう可能性がある．そこで，自動メモ化プロ

セッサは，$M + S = 0$ である場合，過去に再利用が行われた事がある区間であるならば，S を N で初期化する事により検索を再開する機構を備えている．

ところで，前述の並列事前実行では，最も高速化の効果が大きいと考えられる区間に対してこれを適用する事が重要である．そこで，並列事前実行の対象とする区間の選択にも，オーバーヘッドフィルタは用いられる．ここで，図 2 に示した N^{set} はメインコアによるエントリの登録状況を記憶している 64 ビットのシフトレジスタの値である．N^{set} は一定期間毎にメインコアによる登録があれば 1 を記憶し，なければ 0 を記憶する．これにより，最近に登録が行われている程 N^{set} は大きな値となる．この値を用いて並列事前実行を行う区間の優先度を

$$N^{set} \times S \times (C - Ovh^R - Ovh^W) \tag{4}$$

として計算する．式 (4) では，最近に登録が行われており，並列事前実行コアが登録したエントリによる再利用に成功する確率が高く，削減予測サイクル数が大きいような区間の優先度が高くなり，この優先度が高い区間が並列事前実行の対象として選ばれる．また，この値が一定の閾値より大きい区間が存在しない場合には，効率的な並列事前実行ができないと判断し，並列事前実行は行われない．ここで，並列事前実行による再利用の成功にも局所性があると考えられるので，一定期間ごとに並列事前実行を再開する．これにより，成功する確率の低い登録を中止しつつ，登録を中止する事によって起こる性能悪化を最小限に抑えている．また，このように並列事前実行を中止する事で，MemoTbl への不必要な登録の削減も行っている．

さて，RF の容量は有限であるため適宜エントリの追い出しを行わなければならない．そこで，追い出しを行う RF エントリの選択にも，オーバーヘッドフィルタが用いられる．ここで，図 2 に示した N^{lru} はメインコアによる登録状況を記憶している 64 ビットのシフトレジスタの値である．N^{lru} は一定期間毎に再利用の成功やメインコアによる登録があれば 1 を記憶し，なければ 0 を記憶する．これにより，最近に再利用の成功やメインコアによる登録がある程 N^{lru} は大きな値となる．この値を用いて並列事前実行を行う区間の優先度を

$$N^{lru} \times (M + S) \times (C - Ovh^R - Ovh^W) \tag{5}$$

として計算する．式 (5) では，最近に登録や再利用の成功が無く，再利用に成功する確率が低く，削減予測ステップが小さいような区間の優先度が低くなり，この優先度が低い区間がパージ対象区間として選ばれる．このように選ばれた区間に対し，2.2.2 項で述べた RFID パージを用いて追い出しを行う．

3 提案

自動メモ化プロセッサが持つ MemoTbl の容量は有限である．そのため，再利用による効果が得られない区間の登録を中止することで，MemoTbl を有効活用するべきである．そこで，本研究では MemoTbl への登録時に過去の再利用の成功履歴を考慮する事により，これを実現する手法を提案する．また，それに伴い既存の追い出し手法では問題が生じるので，新たな手法を提案する．更に，効果の正確な予測のために，新たな予測削減サイクル数の更新方法を提案する．

3.1 過去の履歴を用いた登録中止

既存の自動メモ化プロセッサは 2.4 節で述べたように，オーバーヘッドフィルタにより MemoTbl への登録を中止する際の判定基準として，再利用区間に対する削減可能サイクル数，検索オーバーヘッド，書き戻しオーバーヘッドを用いる．このように，オーバーヘッドフィルタで登録を中止する基準に過去の履歴は用いられない．一方で，MemoTbl の検索を中止する基準には過去の履歴が用いられており，再利用を開始してから一定回数の検索が行われた時に，再利用が一度も成功しなかった区間は，2.2.2 項で述べた RFID パージが行われるまで検索が行われない．このような指標の違いにより，検索が行われる事がないエントリが MemoTbl へ登録される場合がある．そこで，登録時にも過去の履歴を用いる手法を提案する．この手法では，RF に区間を登録してから一定回数の検索が行われるまでの間に再利用が一度も成功していなかった場合，その区間への登録を行わないようにする．それにより，既存のモデルでは行われてしまう MemoTbl への不必要なエントリの登録を大幅に削減する事ができる．

3.2 コアの種別を考慮した登録中止機構

3.1 節では，RF に区間を登録してから一定回数の検索が行われるまでの間に再利用が一度も成功していないような区間の登録を中止する手法を提案した．この手法ではどのコアが登録を行った事により再利用が成功したかは考慮されていない．しかし，入力の一つとしてイタレータ変数を持つようなループイタレーションなど，並列事前実行コアの登録したエントリによる再利用は成功する場合でも，メインコアの登録したエントリによる再利用は成功しない場合があると考えられる．そこで，メインコアが登録したエントリによる再利用成功が無く，並列事前実行コアの登録したエントリによる再利用成功確率の高い場合が実際に存在する事を確かめるために，SPEC CPU95

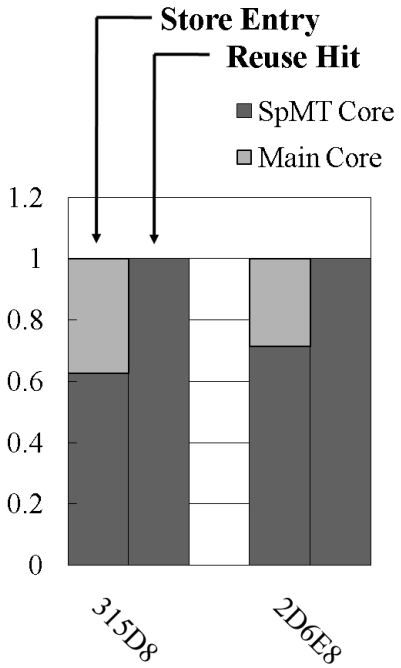

図 6: メインコアと並列事前実行コアの登録による再利用成功確率

ベンチマークプログラムの中で，再利用による効果が大きい 124.m88ksim について，再利用区間ごとに登録数と再利用の成功数を計測した．その結果より，実際にその特徴を持っていた再利用区間を**図 6** に示す．

図 6 で示されるグラフは，左から順に登録を行ったコアの比率，どのコアが登録したエントリにより再利用が成功したかの比率を示したものである．また，横軸の値は再利用区間の開始アドレスを 16 進数で表している．このグラフから分かるように，メインコアによる登録では再利用が全く成功していないような再利用区間が存在する．このような場合，並列事前実行による登録は中止するべきではないが，メインコアによる登録は中止するべきである．そこで，RF に区間を登録してから一定回数の検索が行われるまでの間に，メインコアが登録したエントリによる再利用の成功が一度もなかった場合，その区間の登録を中止する手法を提案する．しかし，このような登録の中止を行うと，メインコアと並列事前実行コアの両方の登録による再利用の成功確率が高い命令区間において，登録してからの一定期間中に並列事前実行コアの登録による成功だけが起こってしまった場合，成功確率の高いメインコアによる登録を中止してしまう．そこで，この手法により中止された登録は一定回数の登録の中止があるごとに再開する機構を備えるようにする．これにより，登録を中止する事によって起こ

る性能悪化を最小限に抑える．

さて，ストライド予測に用いる入力の履歴はMemoTblへエントリの登録があった場合にのみ更新されるので，メインコアによる登録を中止するとその分の入力の履歴が更新されなくなる．これにより，メインコアによる登録を中止する事でストライド予測が当たらなくなり，並列事前実行コアによる登録も中止されてしまう．しかし，メインコアによる登録を中止しても並列事前実行は動作させるべきである．そこで，提案手法により登録を中止する時，ストライド予測に用いる入力の履歴の更新を行う事でこの問題を解決する．

また，既存手法ではメインコアがMemoTblへとエントリを登録するタイミングでTSIDパージを行うので，メインコアによる登録を中止するとTSIDパージが行われる回数が減少する．これにより，TSIDパージが行われないまま並列事前実行コアによる登録が続けられ，RB，RA，W1エントリが溢れる可能性が高くなるので，RFIDパージが行われる回数が増加する．RB，RA，W1エントリが枯渇した時に行われるRFIDパージは再利用の効果が高いエントリを追い出してしまう可能性があり，このようなRFIDパージの増加は性能の悪化を引き起こす．そこで提案手法により登録を中止する時，TSIDパージを行う事でこの問題を解決する．

それでは，コアの種別を考慮した登録中止の適用による効果について具体的な例を用いて説明する．**図7**に示すプログラムでは，提案手法を適用する事により，再利用率を向上させる事ができると考えられる．このプログラムについて簡易なモデルで提案手法の働きを見る事にする．簡易モデルでは，RBの最大エントリ数が100ライン，*functionA*の登録に使用するRBエントリ数が10ライン，*loopA*の1イタレーションの登録に使用するRBエントリ数が5ラインであるとし，並列事前実行コアは3基であるとする．また，*loopA*は並列事前実行の登録による再利用は成功するが，メインコアが登録したエントリによる再利用は成功しないとあらかじめわかっているとする．加えて，簡単のために並列事前実行はメインコアによりi=0のイタレーションが実行されはじめた時点で実行可能とする．ここで，効率的な並列事前実行が行われている場合，図4の(a)に示すように，メインコアによる登録が1回行われるごとに，並列事前実行コアによる登録は3回行われる．

プログラムが実行されると，まず*functionA*が実行され，その入出力がMemoTblへと登録される．続いて*loopA*が20回実行されると，既存手法では*loopA*のエントリの登録に$5 \times 20 = 100$ライン使用されるので，それまでに登録されていたラインが全て追い出される事となる．これにより，既存手法ではループが終了した後の*functionA*

```
int main(void){
     :
  functionA(100);
  for(i=0;i<20;i++) {
      :
    /* loop A */
      :
  }
  functionA(100);
     :
  return(0);
}
```

図 7: サンプルプログラム

の再利用は成功しない．一方，提案手法では *loopA* のメインコアによる登録が行われないので，1/4 の登録が行われなくなり，MemoTbl には $5 \times 20 \times 3/4 = 75$ ライン分のエントリが登録される．これにより 25 ライン分の空きができるので，*functionA* による登録の追い出しは行われない．ここで，*functionA* は前回と同一の引数を受け取っており，再利用に成功する．このように，登録エントリ数が減少すると，追い出されるエントリ数も減少するので，再利用率が向上すると考えられる．また，提案手法を適用した事によって追い出されなくなったエントリによる再利用が成功せず，提案手法により速度が向上しない場合もある．しかし，不要なエントリの登録が減少する事による速度の低下や，再利用率の低下は起こらない．これにより，メインコアが登録したエントリによる再利用が成功しないという予測が正しい限り，提案手法の適用による速度低下は起こらない．

3.3 RFID パージの改良

既存の自動メモ化プロセッサでは 2.4 節で述べたように，再利用による効果が得られないと判断して登録を中止した区間に対しては，RFID パージを行わない．また，3.1 節と 3.2 節で述べた提案手法を適用すると，登録を中止する区間が大幅に増える事になる．これらより，RFID パージを行わない区間が大幅に増え，登録を中止した区間

でRFが埋めつくされる可能性が高くなる．RFが登録を中止した再利用区間で埋めつくされると，それ以降再利用が全く適用できなくなるという問題が発生する．そこで，再利用による効果が得られないと判断したエントリに対してRFIDパージを行うようにすることを提案する．

既存手法では2.4節で述べた式(5)の値が最も小さい区間を追い出しエントリとして選んでいた．これにより，使用頻度が低く再利用に成功する確率の低いエントリを追い出す事ができていた．しかし，削減予測サイクル数が負である区間に対して式(5)を適用すると，使用頻度が高い区間ほど式(5)の値が小さくなってしまうので，使用頻度の高い区間が優先して追い出されてしまう．また，削減予測サイクル数が正である区間よりも，負である区間の方が式(5)の値が小さいので優先して追い出される．ここで，RFから追い出されたエントリは過去の履歴や削減予測サイクル数の情報が無くなってしまう．予測削減サイクル数の情報が無い区間が実行されると，オーバーヘッドフィルタによる登録の中止が行われない．このように，削減予測サイクル数が負である区間を記憶していないと，その区間をMemoTblへ登録してしまう．この問題を解決するために，保護すべきRFエントリの優先度を

$$N^{lru} \times (N^{Main} + N^{SpMT}) \times |C - Ovh^R - Ovh^W| \tag{6}$$

として計算する．式(6)では，前述の削減予測サイクルが負である事により生じる問題を，削減予測サイクル数に対して絶対値をとる事で解決する．これにより，使用頻度や再利用に成功する確率が低いエントリが優先して追い出されるようになる．また，負である区間と正である区間が平等に追い出されるようになる．しかし，オーバーヘッドフィルタが登録を中止している区間は$(N^{Main} + N^{SpMT})$が0となり，式(6)の値が使用頻度や削減予測サイクルに関わらず0となってしまうので，使用頻度や削減予測サイクル数が考慮されなくなる．そこで，オーバーヘッドフィルタが登録を中止している区間においても使用頻度を考慮するために，再利用に成功する確率に最低値を与え，追い出しを行うRFエントリの優先度を

$$N^{lru} \times (N^{Main} + N^{SpMT} + 1) \times |C - Ovh^R - Ovh^W| \tag{7}$$

として計算する．式(7)では，$(N^{Main} + N^{SpMT})$が0である場合にも使用頻度が考慮されるようになる．

3.4 平滑化

これまで，登録を中止すべき区間に着目した手法を提案してきた．一方，本節では登録を中止すべきではない区間に着目する．現在の自動メモ化プロセッサでは，登録を中止する際の指標として用いる，削減できるサイクル数，その再利用に必要となる検索オーバーヘッド，書き戻しオーバーヘッドは，一番最近に計測した値を用いる．このように一番最近に計測した値のみを指標とすると，一度でも削減予測サイクル数が負となると，以降の再利用が中止されてしまう事になる．しかし，再利用によりサイクル数が削減できる事が多い再利用区間については，再利用を中止するべきではない．そこで，削減できるサイクル数，その再利用に必要となる検索オーバーヘッド，書き戻しオーバーヘッドの値の更新に平滑化を用いる手法を提案する．平滑化とは，値の急激な変動を抑制する手法である．削減できるサイクル数の更新に平滑化を用いる場合，過去の削減できるサイクル数を C^{old}，新規に計測した削減できるサイクル数を C^{now} とすると，新しい削減できるサイクル数 C^{new} を

$$C^{new} = (\alpha C^{old} + C^{now})/(1+\alpha) \ \ (0 \leq \alpha) \tag{8}$$

として計算する．ここで α は新規に計測した値の反映しにくさであり，この値が大きい程新規に計測した値が反映しにくくなる．再利用に必要となる検索オーバーヘッド，書き戻しオーバーヘッドについても式 (8) と同様に計算する．ここで，条件分岐によりサイクル数が大きく変わるような関数を例にして平滑化が有効に働く場合について説明する．

図 8 に示すプログラムにおいて *functionB* は，引数 i を条件分岐用の変数として用いている．この時，i が 0 であれば処理 A が行われ，そうでなければ処理 B が行われる．ここで処理 A を再利用した時に削減できるサイクル数が 100 サイクルであり，処理 B を再利用した時に削減できるサイクル数が -10 サイクルであったとする．また，説明のために処理 A と処理 B を除いた部分の予測削減サイクルは考えない．更に，値の更新しやすさ α を 1 とする．プログラム実行中に *functionB* が i=0 で呼ばれ続けていると，その予測削減サイクル数は正であり再利用が行われ続ける．しかし，既存手法では一度でも引数 i が 0 以外の値で呼び出されると，予測削減サイクルが -10 サイクルとなりそれ以降の再利用が中止されてしまう．一方，値の更新に平滑化を用いた場合，引数 i が 0 以外の値で呼び出されると $(1 \times 100 + (-10))/(1+1) = 45$ より，予測削減サイクルは 45 サイクルとなるので，以降の再利用を続ける事ができる．

```
int functionB(int i){
  if(i == 0) {

     /* 処理A */
     /* 削減サイクル：100 */



  }
  else {
     /* 処理B */
     /* 削減サイクル：-10 */
  }
  return i;
}
```

図 8: サンプルプログラム 2

4 実装

MemoTbl への登録時に，メインコアが登録したエントリによるヒット率が低い場合，メインコアによる登録のみを中止できるように，自動メモ化プロセッサを拡張する．本章では提案する機構の実装について述べる．

4.1 過去の履歴を用いた登録中止

3.1 節で述べた，RF に区間を登録してから一定回数の検索が行われるまでの間に再利用が一度も成功していなかった場合，その区間の登録を中止する手法を実現する．具体的には，$N^{Main} + N^{SpMT} = 0$ である場合，過去に再利用が行われた事がある区間であり $info$ フラグの値が 1 であるならば，その区間の登録を行う．一方，過去に再利用が行われた事がない区間であり $info$ フラグの値が 0 であるならば登録を中止する．これにより，不必要なエントリの登録を削減する．

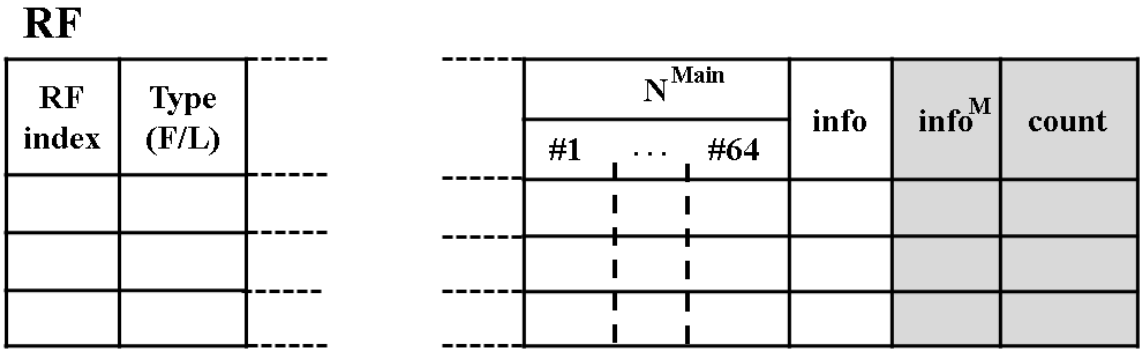

図 9: 拡張後の RF

4.2 コアの種別を考慮した登録中止機構

4.2.1 登録中止

ここでは，3.2 節で述べた一定回数を n 回として説明する．ある再利用区間が RF に登録されてから，その再利用区間に対して n 回の検索が行われるまでの間に，メインコアが登録したエントリによる再利用成功が一度も起こっていなかった場合メインコアによる登録を中止する．

そこで，ある再利用区間が RF に登録されてからその再利用区間に対して n 回の検索が行われた事を知るために，各 RF ラインごとにカウンタ ($count$) を設ける．ここで，後述する拡張のために 4.1 節で述べた並列事前実行コア用のシフトレジスタは用いない．$count$ では，再利用区間に対する検索があるごとにその値をインクリメントする事で検索回数をカウントする．ここで，$count$ は中止の判定が行われるまでの一定回数である n 回のカウントができればよい．そのため，$count$ には $\lceil \log_2 n \rceil$ ビットレジスタを用いることにする．

また，メインコアが登録したエントリによる再利用の成功があったかどうかを知るために，その成功を検出して記憶しておく必要がある．既存手法において，どのコアが再利用エントリの登録を行ったのかという情報は，W1 に記憶されているので，この値を用いる事でメインコアが登録したエントリによる再利用の成功を検出できる．しかし，検出した成功を記憶しておく事ができないので，**図 9** に示すように，各 RF ラインごとにメインコア用の，1 ビットの再利用成功フラグ ($info^M$) を追加する．メインコアが登録したエントリによる再利用の成功があった場合，このビットに 1 をセットする事でそれを記憶しておく．

追加した $info^M$ と $count$ を用いる事で，ある再利用区間が RF に登録されてからそ

の再利用区間に対して検索が一定回数起こった時，メインコアの登録したエントリによる再利用が一度も成功していなければ登録を中止する機構を実現する．具体的には，あるRFエントリの$count$の全ビットが1であり，$info^M$が0であった時，今後その再利用区間のエントリのメインコアによる登録を行わなくする．

また，3.2節において述べたように，メインコアによる登録を中止する事により起こる問題が存在する．そこでコアの種別を考慮した登録中止機構による中止が行われた場合，再利用区間への入力履歴であるPred_distの値を更新し，TSIDパージを行うようにする．

4.2.2 登録再開

メインコアによる登録を中止した場合に，再利用率がかえって低下してしまう場合が存在する．そこで，コアの種別を考慮した登録中止機構により，登録が中止された再利用区間の登録を，一定期間ごとに再開する機構を実装する．この機構は，登録中止機構と同様に再利用区間へ一定回数の検索が行われた時に動作させる．そのために，先程RFに追加した$count$を拡張する．拡張後の$count$も先程と同様に，再利用区間への検索があるごとに，その値をインクリメントして，検索回数を保持する．

登録の中止は4.2.1項で述べた方法と同様にn回の検索を行った時に，メインコアが登録したエントリによる再利用成功が一度も起こっていなければ行われる．その場合，登録を中止してから更に一定回数の検索が行われた時，登録の再開を行うようにする．ここで，本提案では登録を中止してから登録を再開するまでの一定回数を$n \times 3$回とする．登録を中止してから更に$n \times 3$回分の検索が行われた事を知るために，$count$の値を$n \times 4$回までカウントできるように拡張する．そのため，拡張した$count$には$\lceil log_2(n \times 4) \rceil$ビットレジスタを用いることにする．

拡張した$count$と$info^M$を用いる事で，ある再利用区間がRFに登録されてからその再利用区間に対して検索が一定回数起こった時，メインコアの登録したエントリによる再利用の成功が一度も起こっていなければ登録を中止し，登録を中止してから更に一定回数の検索が行われた時，登録の再開を行う．このために，$count$の値がnよりも小さい間は登録を行い，nよりも大きい間は登録を中止する．具体的には，あるRFエントリの$count$がnである時，メインコアによる再利用登録が成功した事がある区間であり$info^M$が1であれば，$count$を0で初期化する事で登録を継続する．一方，メインコアの登録したエントリによる再利用が成功した事がない区間であり$info^M$が0であれば，$count$をインクリメントし続ける事で，$count$の値がnよりも大きくして登録を中止する．登録を中止した場合，登録を中止してから$n \times 3$回分の検索が行わ

れる事で $count$ の値が $n \times 4$ となった時に，$count$ を 0 で初期化する事で登録を再開する．本研究ではこの n の値を 6 として，RF ごとに 8 ビットのレジスタを付加する事でこの機構を実現した．

4.3 RFID パージエントリ選択機構

既存手法ではオーバーヘッドフィルタで登録が中止される区間は記憶されていたが，既存手法の問題点の修正と提案手法の実装を行った事により，再利用中止区間が大幅に増え再利用が適用できなくなる可能性が高くなる．そこで既存の RFID 溢れによるパージアルゴリズムを変更する必要がある．

まず，今後再利用が適用できなくなる場合に対処するために，負の値を持つ区間に対しても RFID パージを行うように変更した．ここで，削減予測ステップ数が負となる事がある区間の追い出しを行うと，削減ステップ数が負の区間を再利用してしまう事が増えてしまう場合がある．

そこで，2.4 節において述べた，使用頻度である N^{lru} の更新方法を変更する．既存手法において使用頻度は，検索成功時と登録時にしかセットされていなかった．これは，既存手法ではオーバーヘッドフィルタにより再利用が中止された区間は追い出しが行われなかったので，オーバーヘッドフィルタにより検索が中止された場合に，N^{lru} を更新する必要は無かったからである．しかし，提案手法では，削減予測サイクル数が負の値を持つ再利用区間であっても，使用頻度が高い区間の追い出しは行わないように N^{lru} の更新を行うようにする．

4.4 平滑化機構

平滑化機構では，3.4 節で述べたように登録を中止する際の指標として用いる削減できるサイクル数，その再利用に必要となる検索オーバーヘッド，書き戻しオーバーヘッドの更新に平滑化を用いる．これらの値は，メインコアによる登録時と，再利用成功時に更新される．本研究では，平滑化に用いる式 (8) 中の α の値を 1 とする．ここで，α の値を 1 とする事により式 (8) の除算が 2 による除算となり，算術 1bit シフトで実現でき高速に計算可能となる．これにより，中止すべきではない区間についてはその登録を続ける事ができる．

表 1: 評価環境

D1 Cache 容量	32KBytes
ラインサイズ	32Bytes
ウェイ数	4ways
レイテンシ	2cycles
Cache ミスペナルティ	10cycles
共有 D2 Cache 容量	2MBytes
ラインサイズ	32Bytes
ウェイ数	4ways
レイテンシ	10cycles
Cache ミスペナルティ	100cycles
Register Window 数	4sets
Window ミスペナルティ	20cycles/set
MemoTbl サイズ	32bytes × 4K 行 (128KByte)
レジスタ ⇔ CAM	9cycles/32bytes
メモリ ⇔ CAM	10cycles/32bytes
CAM ⇒ レジスタ or メモリ	1cycles/32bytes

5 評価

本提案手法を実現するため，既存の自動メモ化プロセッサに対して追加実装を行った．また，提案手法の有効性を示すため，ベンチマークプログラムを用いて評価と考察を行った．

5.1 評価環境

評価には，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．なお，キャッシュや命令レイテンシは SPARC64-III[4] を，MemoTbl 内の RB に用いる CAM の構成はルネサステクノロジ社の R8A20410BG[5] を参考にしている．また，評価対象のプログラムには，汎用ベンチマークプログラムである SPEC CPU95 ベンチマークを用いた．

再利用テストのコストとして，レジスタと CAM を 32 バイト比較するのに 9 サイクル，メモリと CAM を 32 バイト比較するのに 10 サイクルを要するものとする．現在

の一般的なアーキテクチャでは，CPU コア内部のクロック速度は，外部のメモリパスのクロック速度の 10 倍程度で動作している．そのため，レジスタやメモリと，CPU コア外部にある MemoTbl との比較に要するサイクル数は現実的なコストとなる．

5.2 評価結果

評価結果のグラフは，左から順に

(N) メモ化なし

(M) メモ化あり

(P1) 過去の履歴を用いた登録中止 (RFID パージ変更無)

(C1) 過去の履歴を用いた登録中止+コアの種別を考慮した登録中止 (RFID パージ変更無)

(P2) 過去の履歴を用いた登録中止

(C2) 過去の履歴を用いた登録中止+コアの種別を考慮した登録中止

となる．グラフは各モデルの総実行サイクル数を表しており，それぞれメモ化なし (N) を 1 として正規化した．自動メモ化プロセッサには，再利用を行うメインコア 1 つと並列事前実行コア 3 つが割り当てられている．SPEC CPU95 ベンチマークの結果を **図 10** に示す．

図 10 の凡例は順に，exec は命令の実行に要したサイクル数，read_ovh は MemoTbl の比較に要したサイクル数 (検索コスト)，write_ovh は MemoTbl の出力をレジスタやメモリに書き込む際に要したサイクル数 (書き戻しコスト)，cache1_mis 及び cache2_mis は 1 次と 2 次データキャッシュミスにより要したサイクル数，window_mis は SPARC アーキテクチャ特有のレジスタウインドウミスによって要したサイクル数である．read_ovh と write_ovh を合わせて再利用オーバヘッドとなる．

図 10 の SPEC CPU95 ベンチマークにおける評価結果を説明する．提案手法に RFID パージの改良を加えなかった (P1) と (C1) を既存手法である (M) と比較すると，特に 130.li と 147.vortex において速度が低下している．また 132.ijpeg では検索コストが大幅に増加している．これは 2.2.2 節で述べたように，RF が多くの登録停止エントリにより埋めつくされてしまい，有効な再利用が適用できなかったからだと考えられる．一方，124.m88ksim においては速度が向上している．このように高速化が図れたのは RF に余裕があったからだと考えられる．

続いて提案手法である (P2) と (C2) を，RFID パージの改良を加えなかった (P1) と (C1)，また既存手法 (M) と比較する．(P2) と (C2) は概ね全てのプログラムにおいて，

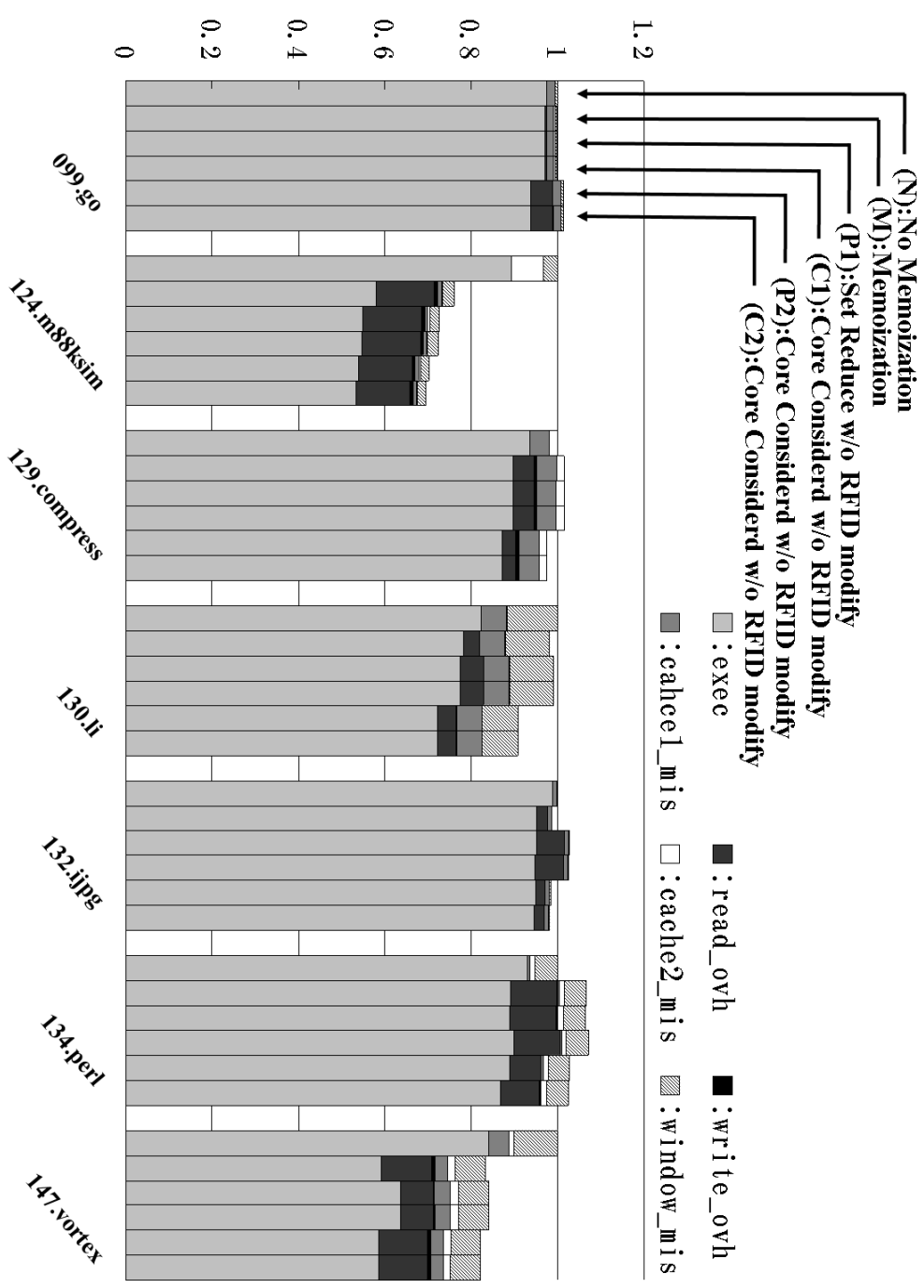

図 10: 評価結果

(P1) と (C1) や (M) よりも高速化できている．しかし 099.go では (P2) と (C2) の両方の性能が悪化している．

更に過去の履歴を用いた登録の中止を行った手法である (P2) とそれに加えてコアを用いた登録中止を行う手法である (C2) を比較すると，124.m88ksim，132.ijpeg で少し高速化している．また，その他の結果においても (P2) を適用した事による悪化はほぼ起こっていない．

結果をまとめると，SPEC CPU95 ベンチマークでメモ化なし (N) に比べ，従来手法では最大で 24.0%，平均で 5.1%のサイクル数の削減だったのに対し，提案手法である (P2) では最大で 30.5%，平均で 8.3%のサイクル数を削減できた．

5.3 考察

提案手法 (C2) と (P2) では RFID パージの改良を加えた事によって，多くの場合で (C1) や (P1) よりもサイクル数が削減できた．これは，不要だと判断した RF エントリを追い出す事により，他の命令区間のために RF エントリを使う事ができるようになったからである．しかし 099.go において (P2) や (C2) では実行サイクル数が削減できているが，検索コストが大きいためにかえって実行時間が増加してしまっている．これは，099.go に再利用を適用してしまう事によって速度が低下してしまう命令区間が多いからだと考えられる．

また，(C2) と (P2) を比べると 124.m88ksim と 132.ijpeg において (P2) が (C2) よりもサイクル数を削減でき，高速化している．ここで，(M) に (C2) を適用した事による登録の停止回数の増加に対する，(C2) に (P2) を適用した事による登録の停止回数の増加を計測したところ，124.m88ksim では，(P2) を適用した事による登録の停止回数の増加は 41.8%，132.ijpeg では 46.8%であり，多くの登録を停止できていた．これにより，MemoTbl を有効活用する事ができ，効率的な再利用が適用され，高速化を図る事ができたと考えられる．しかし，099.go，129.compress，130.li，147.vortex は，(P2) の適用による変化がほぼ見られない．これらの区間に対しても，同様に登録停止回数の増加を計測したところ，130.li と 147.vortex での (P2) の適用による登録停止回数の増加は 2%以下であった．これにより，サイクル数にほぼ変化が無かったと考えられる．また，129.compress での (P2) による登録停止回数の増加は 29.1%と比較的大きな値であったが，再利用に成功する確率はほぼ変化しておらず，総実行サイクル数にほぼ変化が無かった．このように，登録数を削減しても再利用の成功確率にあまり変化が見られない場合も存在する．また，(C2) と比べ (P2) で 0.1%以上性能が悪化するものは見られなかった．これは 3.2 節で述べたように，(P2) が登録の再開機構を備えているため，性能の悪化が抑えられているからだと考えられる．

(C2) による登録の停止回数に対する (P2) による登録の停止回数の増加を計測したところ，124.m88ksim では (C2) による停止の 41.8%，132.ijpeg では 46.8%であり，多くの登録を停止できていた．これにより，MemoTbl を有効活用する事ができ，効率的な再利用が適用され，高速化を図る事ができたと考えられる．しかし，099.go，129.compress，130.li，147.vortex は，(P2) の適用による変化がほぼ見られない．これらの区間に対して，(C2) による登録の停止回数に対する (P2) による登録の停止回数の増加を計測したところ，130.li と 147.vortex での増加は 2%以下であった．これにより，サイクル数にほぼ変化が無かったと考えられる．また，129.compress での増加は 29.1%と比較的大きな値であったが，再利用に成功する確率はほぼ変化しておらず，総実行サイクル数にほぼ変化が無かった．このように，登録数を削減しても再利用の成功確率にあまり変化が見られない場合も存在する．また，(C2) と比べ (P2) で 0.1%以上性能が悪化するものは見られなかった．これは 3.2 節で述べたように，(P2) が登録の再開機構を備えているため，性能の悪化が抑えられているからだと考えられる．

6　終わりに

本研究では，従来の自動メモ化プロセッサの更なる高速化手法として，コア毎に登録の中止を行えるようにする，オーバーヘッドフィルタの改良を提案した．提案手法の有効性を確認するため，SPEC CPU95 ベンチマークを用いて評価を行った．その結果，通常通りに命令を実行するのと比較して，従来手法では最大で 24.0%，平均で 5.1%のサイクル数の削減だったのに対し，提案手法である (P2) では最大で 30.5%，平均で 8.3%のサイクル数の削減となり、有効性を確認できた．

本研究の今後の課題としては，登録を中止した区間でも削減予測サイクル数を考慮する事が挙げられる．これにより，中止されている登録でも効果が得られるような期間には，再利用を行う事ができるようになる可能性がある．これにより，再利用プロセッサの更なる高速化を図る事ができると考えられる．

謝辞

本研究の為に，多大な御尽力を頂き，御指導を賜わった名古屋工業大学の松尾啓志教授，津邑公暁准教授，齊藤彰一准教授，松井浩助教に深く感謝致します．また，本研究へ多くの助言，協力をして頂いた松尾・津邑研究室及び齊藤研究室の方々にも感謝致します．

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] Wang, K. and Franklin, M.: Highly Accurate Data Value Prediction Using Hybrid Predictors, *30th MICRO*, pp. 281–290 (1997).

[4] HAL Computer Systems/Fujitsu: *SPARC64-III User's Guide* (1998).

[5] ルネサステクノロジ: ニュースリリース: R8A20410BG (2009).