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

2004−ARC−157 (23)
2004−HPC− 97 (23)
2004／3／2

2次キャッシュを用いた再利用および並列事前実行機構における高速化手法

笠原寛壽[†] 清水雄歩[†] 津邑公暁[†]
中島康彦[††] 五島正裕[†]
森眞一郎[†] 富田眞治[†]

再利用を用いた並列事前実行機構において使用される再利用表を，現実的な幅を持つCAMにより構成してきた．しかしながら，CAM自体の検索回数が増加しオーバーヘッドが大きくなるため，再利用の実行によりプログラムが性能低下する場合が存在する．本稿では，再利用表検索にかかるコストを評価する機構を追加し，この問題を解決する．予備実行との比較評価を行った結果，CAMに長レイテンシを仮定した場合においても，Stanfordでは予備実行の1%に対して約20−30%，SPEC95では4%に対し7−10%という，良好な平均サイクル数削減率が得られた．また，共有2次キャッシュを仮定し，再利用機構が効果的なプリフェッチ機構としても作用することを示した．

A Speedup Technique for Region Reuse and Parallel Early Computation with Second Level Cache

Hirohisa Kasahara,[†] Yuho Shimizu,[†] Tomoaki Tsumura,[†]
Yasuhiko Nakashima,[††] Masahiro Goshima,[†] Shin-ichiro Mori[†]
and Shinji Tomita[†]

We have implemented reuse buffer for region reuse with general-perpose CAM. However, the times of reference to CAM increase.Since reuse cost becomes large, a program may carry out a performance fall by execution of reuse. This paper proposes a structure which estimates the cost concerning reuse buffer reference. We show the average ratio of eliminated cycles ranges from about 20% to 30% with Stanford benchmark, and from 7% to 10% with SPEC95 benchmark. These results are better than the one with pre-execution: 1% with Stanford and 4% with SPEC95. We also show that reuse structure with second level cache functions well as a structure for pre-fetching.

1. はじめに

現在，投機的マルチスレッディングについて多くの研究が進められている．投機的マルチスレッディングとは，複数の予測値に基づいて，複数のプロセッサでスレッドを実行し高速化を図る手法である．この手法について広く研究されるようになった背景には，スーパースケーラの効果の頭打ちがあげられる．スーパースケーラでは，一般的なプログラムでは依存関係無しに並列実行できる命令の数はせいぜい4−5程度だと言われており，ウインドウサイズを拡大し，load発行を増やすだけでは，これ以上の性能向上は見込めなくなってきている．

投機的マルチスレッディングにおいて，スレッド間のデータの引継ぎはバトンリレー方式が主流であり，1対1のプロセッサ間に引継ぎデータ構造が設けられ，特定の引継ぎポイントを必要とするものである．しかしながら，リレーの際に，複数のマルチスレッディングの実行結果を利用可能とすれば，より高い性能を得られると考えられる．この多対1の引き継ぎデータ構造のモデルとして，我々は区間再利用および並列事前実行による投機的マルチスレッディングを提案している．

区間再利用 (以下，再利用) とは，関数やループなどの命令区間に対して，入力および結果を再利用表と呼ぶ表に保存しておくことにより，再度同一の入力でその命令区間が呼ばれた場合に，命令を実行することなく保存された結果を用いて高速化を図る手法である．また並列事前実行とは，投機スレッドが予測される入力値に基づいて実行自体を事前に行うことによって，結果を前もって再利用表に登録する手法である．これにより，本来，再利用効果が現れない入力値が単調に変化する場合においても，再利用の効果を上げ，高速化を図ることができる．従来の予備実行のように，loadを投機実行することによりキャッシュへのプ

† 京都大学
Kyoto University
†† 京都大学 / 科学技術振興事業団さきがけ研究21
Kyoto University / PRESTO, JST

リフェッチを行う手法とは，本質的に異なる．予測失敗時にペナルティが発生しないという特長がある．

本稿では，再利用時に生じるオーバーヘッドを考慮し，再利用を実行するか否かを判断する評価機構を追加することにより，更なる高速化を図る．また，新たに2次キャッシュを持つハードウェア機構を実現し，キャッシュミスの削減効果を期待する．以下では，まず，我々が提案している再利用と並列事前実行による投機的マルチスレッディングについて説明する．それから，再利用時に生じるオーバーヘッドを考慮し，再利用を実行するか否かを判断する評価機構について述べ，評価を行う．

2. 再利用および並列事前実行

本章では，区間再利用および並列事前実行について概要を述べる．再利用機構を実現するためには，命令区間の入出力を特定できる必要がある．本稿では，SPARC ABIの規定に基づき，入出力を特定できる区間を自動的に識別する機構を想定する．

2.1 再利用

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，過去の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．入出力のセットを記憶するために，再利用表を用いる．再利用表は，命令区間を記憶する表RF，命令区間の入力値セットを記憶する表RB，入力アドレスのセットを記憶する表RA，そして出力値を記憶する表W1から成る．

呼出し命令が発行されると，以下の手順により呼出す区間の再利用を試みる．まず，命令区間の先頭アドレスをRFから検索し，RFに登録済みの命令区間であれば，RB，RAからレジスタ引数および主記憶値の一致比較を行う．これを，内容を比較すべきアドレスがなくなるまで繰り返し，全ての入力が一致した場合，W1に登録されている出力を書き戻すことにより，命令区間の実行を省略する．入力が一致せずに，再利用できなかった場合には，その命令区間を実行し，その入出力情報を新たに表に追加登録する．

2.2 並列事前実行

再利用を行うプロセッサ（Main Stream Processor：以下MSPと略する）とは別に，命令区間の事前実行によりRB，RAを有効化するプロセッサ（Shadow Stream Processor：以下SSPと略する）を複数個設けることにより，さらなる高速化を図る．演算器，レジスタ，1次キャッシュは各プロセッサごとに独立しており，再利用表，2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．図1に並列事前実行機構の概要を示す．MSPは，自身あるいはSSPが再利用表に登録したエントリを再利用する．

MSPおよび各SSPが命令区間の入出力を再利用表に登録する際，巨大な再利用表を毎回直接参照するとオーバヘッドが大きくなる．このため各MSP/SSPに，作業用の小さい再利用表RWを用意し，区間実行の終了時に一括してRWの内容を再利用表本体に登録する(store)．MSPは，命令区間実行開始時に，再利用表を参照し(reuse test)，入力セットが完全に一致するエントリが見つかった場合，当該エントリに対応する出力をW1から1次キャッシュおよびレジスタに書き戻す(write back)．これにより，命令区間の実行が省略される．SSPは，再利用表のこれまでの入力セットから予測される入力セットを受けとり，投機実行を行う．投機実行の結果得られた入出力セットは，MSPと同様に命令区間の実行終了時にRWから再利用表本体に(store)される．

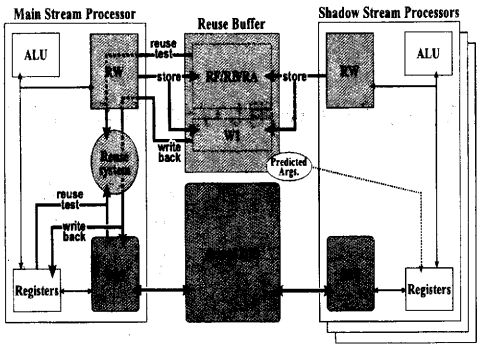

図1 並列事前実行機構

2.3 汎用CAMによる再利用表構成

RB，RAにおいて命令区間の入力を格納する部分は，複数のエントリを一度に比較可能とするため，CAMを用いて構成する．しかし，命令区間の入力となる主記憶アドレスは多様であり，それらを保持するために，再利用表には2Kwordという非常に大きな幅を仮定する必要があった．これは実在のCAMと比較すると非現実的である．そこで我々は，汎用CAMをより幅の広いCAMとして利用する手法[1]をベースとして，汎用CAMにより再利用表を実現している．汎用のCAMにおいて基本となっているシングルマッチによる構成を可能にするのと同時に，再利用表の幅を超えるような入力セットにも対応し，少ない入力セットしか登録されない場合や，同じ内容の部分が複数エントリに現れる(冗長性のある)場合などに生じる記憶域の無駄を省いている．この手法を以下に説明する．

前節で述べたように，各MSP/SSPには作業用の小さな再利用表RWを持たせ，命令区間実行時に再利用表本体に登録を行う．この際，命令区間の入力を多数記憶できるよう，RWにはある程度の幅が必要である．これに対し再利用表本体は，CAMで構成することを考えるとあまり大きな幅を仮定することはできない．図2に，RW，RB，RA，W1の構成の概略図を示す．MSPおよびSSPが命令区間実行中に用いるRWは，1行が1入力セットに対応する．各行は，

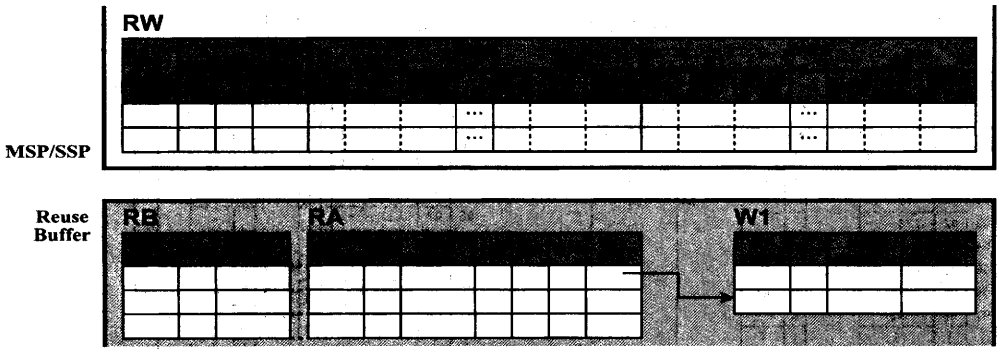

図 2 再利用表の構成

当該命令区間が記憶されている RF 内のインデクス (RF index)，命令区間実行開始時のスタックポインタの値 (SP)，関数/ループの別 (F/L)，ループ終了時の分岐方向 (taken)，そして命令区間の入出力値セット (Read, Write) から成る．入出力の格納域は 1word の入出力が格納できる単位に区切られており，それぞれ，当該アドレスに書き換えがあったか否かを記憶するフラグ (flag)，参照すべき次入出力の主記憶アドレス (next addr.)，そして入出力値およびマスク (Mask, Val.) から成る．

命令区間の実行が終了すると，生成された入出力セットの RW エントリを，再利用表本体 RB，RA，W1 に格納する．以下ではこれらについて説明する．

RB は入力値を記憶するための表，RA は次に参照すべき入力の主記憶アドレスを記憶するための表である．なお RB と RA の各行は 1 対 1 に対応する．RB の各行は，当該エントリの親エントリを指すインデクス (key) と，入力値およびそのマスクを格納する部分 (Mask, Val.) から成る．そして RA の同じ行が，その入力の次に参照すべき主記憶アドレス (next addr.)，およびそのアドレスに対する書き換えが発生したか否かを記憶するフラグ (flag) を保持する．なお，RA 内の UP，Alt.，DN は，主記憶値テストが必要ないエントリに関して連想検索自体を省略するための拡張である．書き込み (更新) が行われていない主記憶から成るものが，テストの必要ないエントリである．UP は親チャンクのインデクス，Alt. は次に検索すべきアドレス，DN は次に検索すべきインデクスを示している．主記憶値テストが必要ないエントリに関しては，DN および Alt. をキーとして次の検索を行う．これによって，主記憶テストの必要がない途中の入力セットを迂回して検索を続けることができる．

実際に RB，RA への格納例を，図 3 に示す．この上部に示している構造図は，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．再利用登録時以降，命令区間が参照する入力アドレスに対して書き換えが発生していない場合は，再利用テスト時にその入力値を比較する必要がない．よって，入力アドレスそれぞれに対し書き換えがあったか否かを flag を用いて記憶しておくことで，比較の必要があるアドレスに対してのみテストを行うことが可能となり，再利用テストのオーバヘッドが軽減できる．フラグ操作の詳細については，文献[2] を参照されたい．さて，本稿ではこの flag を終端フラグとしても用いることとする．これにより，可変長の入力セットを再利用表によって記憶することが可能となる．

また RA の各行は，出力を記憶する表 W1 の行を指すポインタを含む．これによって，入力が完全に一致した場合に，それに対応する出力が格納されている W1 の行を知ることができ，その内容を 1 次キャッシュおよびレジスタに書き戻すことで命令区間の実行を省略する．

ここで，前節で挙げた幅広 CAM による構成時の課題について考える．まず固定幅の問題は，上述のとおり RA 内の flag を終端フラグとして兼用することで，可変長データが格納可能となり，解決される．また，冗長性およびマルチマッチの課題についても，ツリー構造上で共通の幹の部分は RB および RA 上でも 1 つのエントリとして格納されるので，やはり解決されることが分かる．またこのことより，幅広 CAM を前提とした場合に再利用表を構成するには膨大なハードウェア量が必要であったのに対し，汎用 CAM を用いることによって必要ハードウェア量も大幅に抑えられると考えられる．

再利用テストは，RA から読み出したアドレスを用いてキャッシュを参照し，それにより読み出した値を RB で検索する，という手順を繰り返すことにより行う．以下では図 4 の例を用いて，基本的な検索時の動作を説明する．

まず，ツリーのルートを表すインデクス FF およびレジスタの値で RB を検索する．例ではレジスタの値は 00001000 である．RB のインデクス 00 の行がマッチし，RA の同行が参照される．次に参照すべき主記憶アドレスは A1 であるものの，flag がオフであるため値を読み出す必要はなく，マッチしたインデクス 00 を用いて RB を検索する．次にインデクス 01 の行が

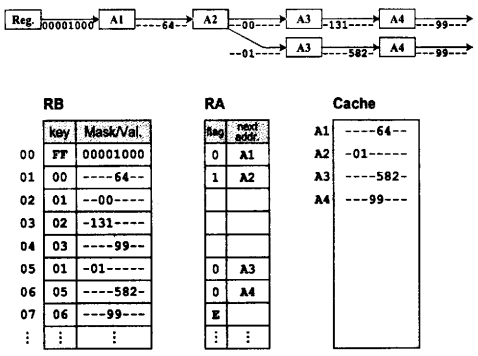

図 3 入力セットの RB/RA への格納

マッチする．同様に RA を参照すると，flag がオンであるため，主記憶のアドレス A2 を参照し，インデクスおよび読みだした値の双方，すなわち 01:--01---- をキーとして RB の検索を行う．同様に RB の検索を繰り返し，RA に終端フラグが現れると，全ての入力値が一致したことになる．当該 RA 行に格納されている W1 へのポインタにより，W1 から出力値を読みだす．

3. 再利用オーバーヘッドの評価機構

過去に実行された命令区間が再び呼出される場合，再利用表の中から同一入力のエントリを検索する．汎用 CAM による再利用表の実現によって，CAM 自体の検索回数が増えるため，比較する項目 (レジスタおよび主記憶の値) が多い場合，大きなオーバーヘッドが生じる．これにより，再利用を用いると反って遅くなる可能性も大きくなる．再利用を行うことによって高速化されない場合を認識し，事前に再利用を中止する機構を追加することにより，再利用の効率を上げ高速化を図る．本章では，その評価方法を示し，再利用オーバーヘッド評価機構の動作方法を述べる．

3.1 オーバーヘッド評価方法

再利用不可であった場合，エントリを検索するためにかかったサイクル数 (以下，検索コスト) が無駄となる．ある命令区間の数回の実行において，無駄になる検索コストの合計が，再利用によって削減できるサイクル数の合計よりも大きくなる場合，その数回分での再利用は効果がないことになる．つまり，区間ごとに過去規定回数分の呼出しに対して再利用を行ったか否かを保持しておき，その回数分内での無駄な検索コストが削減サイクル数を上回るならば，再利用を行わないこととする．検索コストや削減サイクル数の計算には，区間が実行される際，ステップ数や，入力値の検索および出力の書き込みに要するサイクル数を算出しておき，最近の実行におけるこれらの値を使用する．過去の規定回の呼出しにおいて，無駄な検索コストは，(再利用しなかった回数) × (同一入力値の検索に要するサイクル数) であり，削減サイクル数は，(再利用した回数) × (1 度の再利用により削減されるサイクル数) となる．1 度の再利用により削減されるサイクル数は，ステップ数から検索および書き込みサイクル数を引いたほどである．

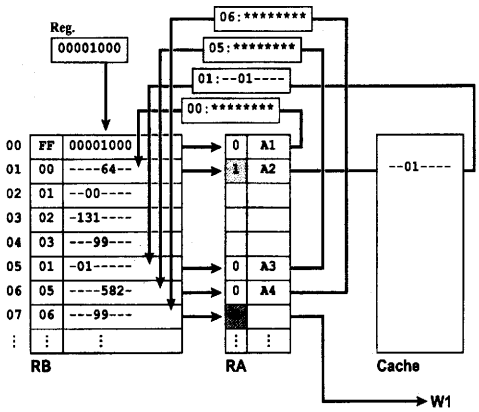

図 4 再利用テストの手順

この手法では，再利用しないケースが続いた場合，無駄な検索コストの値が大きくなるため，再利用効果が現れるにも関わらず，再利用のための検索を中止することがある．このため，長く再利用しない状況が続いたときには，過去に再利用した回数の値を変えてやることによって，再利用効果の出現に対応する．

3.2 オーバーヘッド評価機構の動作

過去に実行された区間が再び呼出されると，まず，無駄になった検索コストと削減サイクル数を計算する．これらを比較し，検索コストのほうが大きい場合，再利用を中止する．一方，検索コストが小さい場合は同一入力のエントリを検索し，見つかると再利用される．エントリが見つからなかった場合は，その命令区間を実行する．命令区間の実行を再利用表に登録するときは，実行した命令ステップ数，同一入力値を検索する際にかかるサイクル数および，出力値を書き込む際にかかるサイクル数を算出しておく．ここで，命令区間における実行ステップ数が，検索および書き込みに要するサイクル数の合計よりも小さい場合は，そもそも再利用による効果が得られないため，再利用表へ登録しないこととする．再利用表へ登録されない区間は，SSP で実行されることもない．

4. 評　　価

前章に述べたオーバーヘッドを評価する手法を用いて，再利用および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．

4.1 評価環境

評価には，再利用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

表 1　シミュレーション時のパラメータ

ラインサイズ	32 Byte
1 次 cache 容量	32 KByte
1 次 cache ウェイ数	4
1 次 cache ミスペナルティ	10 cycles
2 次 cache 容量	2 MByte
2 次 cache ウェイ数	4
2 次 cache ミスペナルティ	100 cycles
Register-Window	4 set
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
RW 深さ	6
RF エントリ数	256
SSP	3

用いた各パラメータを表 1 に示す．命令レイテンシは，SPARC64-III[3] を参考にしている．

ハードウェア構成に関しては，1 次キャッシュを 32KB：4way とし，共有 2 次キャッシュは 2MB：4way とした．また，1 次キャッシュ，共有 2 次キャッシュ，主記憶のレイテンシはそれぞれ，2 サイクル，10 サイクル，100 サイクルと仮定した．RW は 32Byte 幅 × 256 エントリ が 4 セット から成り，1 次キャッシュと同サイズの 32KB とし，レイテンシも 1 次キャッシュと同じと仮定した．一方，RB も 2 次キャッシュと同サイズの 2MB とした．

MSP が D1/D2 に対し store を行うと，SSP 内の D1 で invalidate が発生し，後続命令をそれぞれ 10 サイクル/100 サイクルだけ stall させる．以後 SSP には，D1 ミスとして観測される．また，MSP/SSP で D2 に対し load が発生した場合，それより 100 サイクル後以降は MSP/SSP 内の D1 上で hit すると仮定した．

4.2 Stanford による評価

Stanford ベンチマークを，gcc-3.0.2 (-msupersparc -O2) によりコンパイルし，スタティックリンクにより生成したロードモジュールを用いた．オーバヘッド評価機構を用いない場合を図 5 に示す．なお，再利用表 CAM のレイテンシは，レジスタとの比較に 32Byte/1 サイクル，キャッシュとの比較に 32Byte/2 サイクル (以下，$1\tau/2\tau$ と記す) を仮定している．

各ベンチマークのグラフは，左から順に，(M)SSP と再利用のいずれもなし，(P) 予備実行によるプリフェッチ，(W) 幅広 CAM (幅 2Kword × 深さ 256 × 登録可能区間 32 = 64MB)，(Gs) 汎用 CAM (幅 256bit × 深さ 4K = 128KB)，(Gm) 汎用 CAM (幅 256bit × 深さ 64K = 2MB)，(Gl) 汎用 CAM (幅 256bit × 深さ 256K = 8MB) を用いた場合に要したサイクル数であり，それぞれ (M) を 1 とした正規化を行っている．なお，(W) では，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る．test(r)，test(m) はそれぞれ，再利用表とレジスタ，再利用表とキャッシュの比較に要したサイクル数である．write は，命令区間の出力を再利用表からレジスタおよびキャッシュへ書き戻すのに要したサイクル数である．また，D1$，D2$，および window は，それぞれキャッシュミスペナルティとレジスタウィンドウミスによるペナルティ

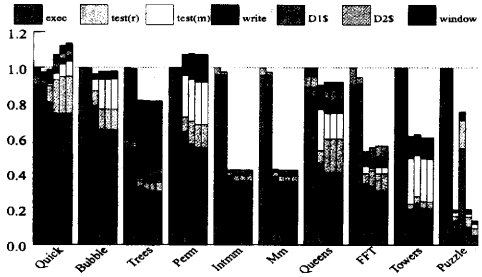

図 5　MSP の実行サイクル数 (Stanford，コスト評価なし)

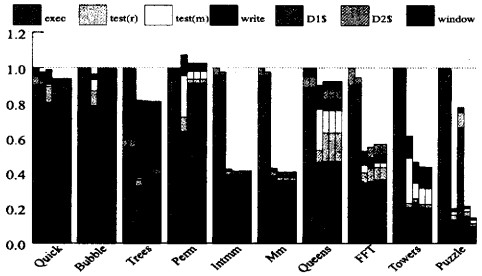

図 6　MSP の実行サイクル数 (Stanford，コスト評価あり)

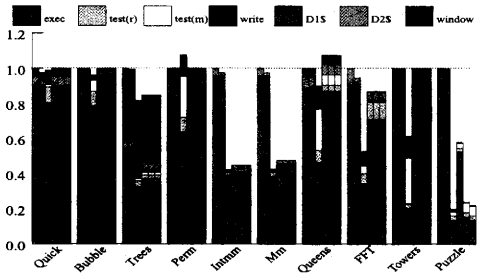

図 7　MSP の実行サイクル数 (Stanford，コスト評価あり，長レイテンシ)

を表している．

(Gs) は従来仮定してきた幅広 CAM の 1/512 のハードウェア量で実現できるにもかかわらず，Puzzle 以外の全てのベンチマークで (W) と同等もしくはそれ以上の高速化が得られている．また，従来の 1/32，1/8 のハードウェア量で実現できる (Gm)，(Gl) では，Bubble，Queens，FFT では僅かに (W) に劣るものの，Puzzle も含む全てのベンチマークで (W) と同等以上の性能が得られた．

次に，オーバヘッド評価機構を用いた場合を図 6 に示す．Bubble では僅かに悪化しているものの，Quick，Perm，Towers において高速化が見られる．Towers においては，一度も再利用できない命令区間について検索を行っていた無駄が解消されている．ステップ数の小さい命令区間を再利用する Quick，Perm において

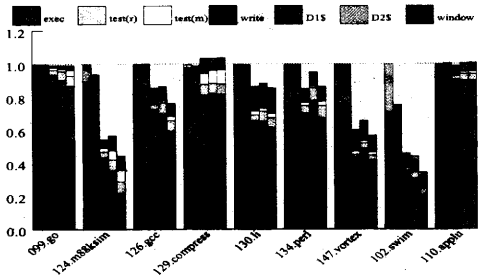

図 8 MSP の実行サイクル数 (SPEC95, コスト評価なし)

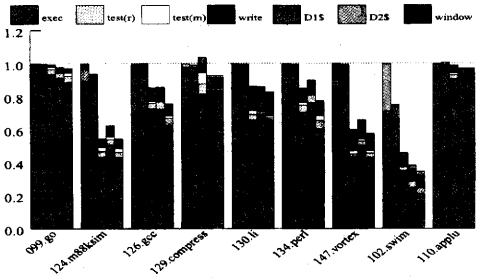

図 9 MSP の実行サイクル数 (SPEC95, コスト評価あり)

は，再利用を用いることにより逆に性能低下を起こしてしまう場合を回避することができていることが分かる．平均サイクル削減率を調べたところ，(P) では 1% に留まったのに対し，(Gs)，(Gm)，(Gl) ではそれぞれ 32%，40%，42% という高い値となり，予備実行よりも有意に良好な結果が得られた．

次に，より現実的な環境として汎用 CAM のレイテンシを大きく仮定した場合の測定を行った．具体的には，再利用表とレジスタとの比較に 32Byte/9 サイクル，キャッシュとの比較に 32Byte/10 サイクル (以下，$9\tau/10\tau$ と記す) を仮定した．この結果を図 7 に示す．レイテンシが非常に大きくなっているにもかかわらず，FFT，Towers 以外では大きな性能低下は見られない．特に，再利用表のサイズが大きくなるに従い発生するはずの性能低下を回避できており，オーバヘッド評価機構の有効性が見てとれる．平均サイクル数削減率においても，(Gs)，(Gm)，(Gl) はそれぞれ 21%，27%，28% となり，良好な結果となった．

4.3 SPEC95 による評価

次に，SPEC95 による結果を示す．Stanford ベンチマーク同様の方法で生成したロードモジュールを用いた．まず前節と同じく，再利用表のレイテンシを，$1\tau/2\tau$ と仮定し，オーバヘッド評価機構を用いない場合および用いた場合の結果をそれぞれ図 8，図 9 に示す．それぞれのベンチマークのグラフは，左から (M)，(P)，(W)，(Gs)，(Gm)，による結果である．

129.compress などから分かるように，やはり Stanford と同様に，再利用の適用による性能低下を回避することができている．平均サイクル数削減率は，(P) の 4% に対し，(Gs)，(Gm) は 23%，29% という値となり，Stanford 同様良好な結果となった．次に，再利用表のレイテンシを，$9\tau/10\tau$ と仮定した場合に，オーバヘッド評価機構を用いた結果を図 10 に示す．全体的に性能は低下し，(W) には大きく劣るものの，オーバヘッド増大による速度低下を抑えることができている．平均サイクル数削減率は，(Gs)，(Gm) それぞれ 7%，10%となり，(P) と同等以上の結果が得られた．

また，予備実行には及ばないものの，128.m88ksim や 102.swim などにおいて，2 次キャッシュミスが大幅に削減されている．このように，Stanford の結果にはほとんど表れていなかったが，本再利用機構が共有

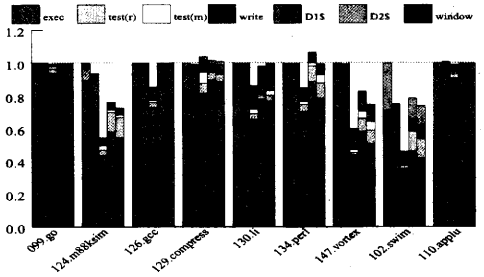

図 10 MSP の実行サイクル数 (SPEC95, コスト評価あり，長レイテンシ)

2 次キャッシュに対して，予備実行と同様に効果的なプリフェッチ機構としても働いていることが分かる．

5. おわりに

汎用 CAM による再利用表の実現により検索コストが大きくなるため，コストが再利用による削減ステップを上回るような場合に再利用自体を中止する手法を提案し，Stanford および SPEC95 により評価を行い，再利用の適用によって却って性能が低下するような場合を回避することができることを示した．

再利用表においては理想的なレイテンシと長レイテンシを仮定したが，いずれの場合においても予備実行より有意に良好な結果が得られることが分かった．また，本機構により 2 次キャッシュミスが削減されることを示し，予備実行と同様に効果的なプリフェッチ機構としても作用していることを示した．

参考文献

1) MUSIC SEMICONDUCTORS: Using The MU9C1965A LANCAM MP For Data Wider 128 Bits,1998.
2) 津邑公暁, 中島康彦, 五島正裕, 森眞一郎, 富田眞治: 並列事前実行機構における主記憶値テストの高速化, 情報処理学会論文誌: コンピューティングシステム,Vol.45,No.SIG 1(ACS 4)pp.31-42,2004.
3) HAL Computer Systems/Fujitsu: SPARC64-III User's Guide,1998