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

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[†]

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実現に必要となる再利用表を，従来，非常に幅広の CAM を用いて構成することを仮定してきたが，本稿ではより現実的な大容量の汎用 3 値 CAM を用いて実現する方法を提案する。幅の狭い汎用 CAM に再利用エントリを効率的に格納する手法について述べ，さらに，再利用テストのオーバヘッド増大を抑えるために，コスト評価機構を追加することで，再利用の適用によるプログラムの性能低下を回避できることを示す。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，共有二次キャッシュを仮定することで，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。

An Efficient Imprementation of Parallel Early Computation with Large General-Purpose Ternary CAM

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

We have proposed an asymmetrical speculative multi-threading with reuse and parallel early computaion. This paper proposes an imprementation of reuse buffer for parallel early computaion with large general-purpose ternary CAM. We show an efficient way of mapping reuse input/output set onto ternary CAM. Futhermore, we propose a structure which estimates the reuse costs. It helps the reuse occasionally reducing program performance. 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 the reuse structure functions well as a structure for pre-fetching.

1. はじめに

近年，処理速度の向上により，主記憶からのデータ転送速度が相対的に遅くなってきている。このことから，投機的マルチスレッド (Speculative Multi-Threading: 以下，**SpMT**) 機構を利用して load 命令を事前に実行し，効果的なプリフェッチ機構として利用する研究 (Pre-execution: 以下，**予備実行**) が多く行われている[1)~4)]。

さて従来の SpMT では，投機実行スレッドと通常実行スレッドの間は，1 対 1 でデータの引き継ぎが行われる。これに対し，データの引き継ぎを多対 1 とし，実行スレッドおよび複数の投機スレッドによる実行結果を out-of-order に利用可能とすれば，より高い性能が得られると考えられる。我々はこの 1 モデルとして，再利用および並列事前実行による非対称な SpMT を提案している[5)]。

再利用 (Reuse) とは，関数呼び出しやループなどの命令区間に対して，実行時にその入出力の組を再利用表に記憶しておき，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，記憶されている出力を利用することで命令区間の実行自体を省略する高速化手法である。再利用の特長は，入力値さえ一致すれば実行結果を検証する必要がない点，および再利用の対象とする命令区間が増えても必要となる機構の複雑さが増大しない点にある。

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

また**並列事前実行** (Parallel Early Computation) は，再利用表に登録されている命令区間に対してこれから出現するであろう入力を予測し，通常スレッドの実行と並行して，投機スレッドがその予測された入力値に基づいて実行自体を事前に行い，その結果を再利用表に登録する手法である。これにより，入力が単調に変化する場合など，過去の実行結果の単純な再利用では効果がない命令区間に対しても，再利用の効果を上げ，高速化を図ることができる。

この並列事前実行機構では，再利用表が投機実行スレッドと通常実行スレッドの多対 1 のデータ引き継ぎ構造として働き，全体として，効果的な SpMT として機能する。また，並列事前実行で予測対象となるのは，あくまで命令区間に対する入力である。このため予測がはずれた場合でも，通常実行スレッドから投機の失敗は観測されず，投機実行のキャンセルおよび再実行のためのオーバヘッドが発生しないという利点がある。従来の投機実行や予備実行とは本質的に異なる手法である。

さて，これまで我々は再利用表を実現するにあたり，再利用対象となる命令区間の多様な入力を記憶するために，幅 2K word におよぶ非常に巨大な CAM (Content-Addressable Memory) を仮定してきた。これにより，再利用表の構成および操作が容易となり，また検索コストも小さく抑えることができた。しかし一方で今日，二次キャッシュサイズは大きなものでも 2MB 程度であり，また CAM に関しては，大規模なものでも幅が 256bit 程度である。よって並列事前実行機構をより現実的なものとするには，これらのキャッシュと同程度の大きさの，幅の狭い汎用 CAM で再利用表を構成する必要があると考える。

そこで本稿では，汎用の 3 値 CAM を用いて効率的に再利用表を実現する方法について述べる。汎用 CAM の幅は大きくないため，命令区間の入出力セットを折畳んで格納するなどの工夫が必要となる。これに伴い検索オーバヘッドが大きくなり，再利用全体においてオーバヘッドの増大が予想される。この解決策として，既に提案している主記憶値テストの高速化手法[5] に加え，オーバヘッドのコスト評価機構を新たに追加することで，全体オーバヘッドの抑制を図る。また，最近の記憶階層に倣い，並列事前実行機構に共有二次キャッシュを設け，キャッシュミスの削減効果の評価を行う。本稿で提案する手法をシミュレータ上に実装し，予備実行とあわせて評価・比較を行った。

以下 2 章では，再利用および並列事前実行機構の概要について述べ，3 章で従来仮定してきた幅広 CAM による構成の問題点を明らかにする。その後 4 章で汎用 CAM による再利用表の実現方法を示し，機構のハードウェアモデルについて述べる。5 章では再利用オーバヘッドのコスト評価機構について述べ，6 章でシミュレーションによる評価結果を示す。最後に，7 章でまとめを行う。

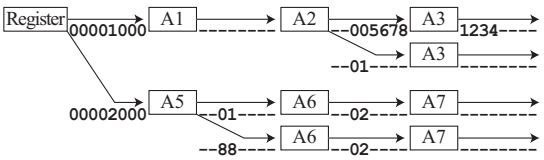

図 1 命令区間の入力参照

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。再び同じ命令区間を実行する必要が起こったとき，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。

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。

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

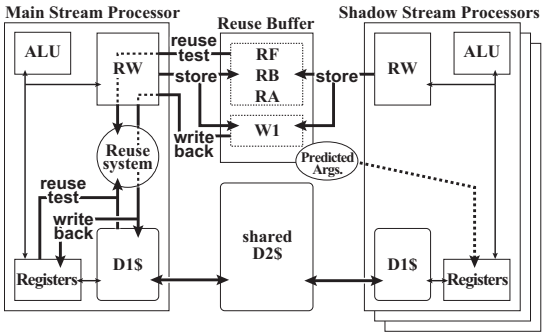

図 2 並列事前実行機構

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，一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，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 が再利用表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，結果を再利用表に登録する。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，結果が再利用表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 中央に示した再利用表が，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。

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。また，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 から一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。

3. 従来方式の問題点

これまで我々は，命令区間の入力パターンを保持する再利用表の最も単純な構造として幅広の CAM を仮定し，CAM の 1 行を 1 入力セットに対応させてきた[5]。これにより表の構成および動作モデルは簡潔なものとなった。しかし命令区間の入力となる主記憶アドレスは 2.1 節で述べたように多様であり，それらを保持するために，再利用表には 2K word 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。また，このほかにも，以下のような様々な課題があった。

固定幅：再利用表には 2K word という十分な幅を想定しているが，1 行の長さは再利用表の幅に等しくなるため，それを超えるような入力セットは登録できないという問題がある。また，逆に入力セットを多く登録できるよう十分な幅を再利用表にとった場合，少ない入力セットしか登録されない行に関しては，記憶域の大きな無駄が生じる。

冗長性：ある命令区間において，多くの入力のうち途中までの入力セットは全く同じで，ごく一部のみが異なっている場合を考える。1 行を 1 つの入力セットに対応させているため，このような場合も再利用表では，途中まで全く入力の項目が同じであるような 2 つの行を使用することになり，入力セットの共通部分が重複して登録されることになる。これは記憶域の使用効率という点で，好ましくない。

マルチマッチ前提：再利用対象となるのは，最終的に全入力が一致した 1 行のみであるが，各入力に対し順に一致テストを行っている検索途中の時点においては，再利用表上でマルチマッチが存在している。これに対し汎用の CAM はシングルマッチが基本であり，マルチマッチのまま構成しようとするのは困難である。

4. 汎用 CAM による再利用表の実現

本章では，汎用の CAM を用いて，再利用表を構成する具体的手法について述べ，前章で挙げた問題点が解決されることを示す。

4.1 ハードウェアモデル

2 章で述べたように，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。ネットワークルータ等に用いられる汎用 3 値 CAM は，大きなものでも幅が 256bit 程度である。よって，RW の長いエントリを折畳んで格納する必要がある。文献 6) には，汎用 CAM をより幅の広い CAM として利用できる方法が示されており，以下ではこの手法をベースとして，汎用 CAM により再利用表を実現する方法について述べる。

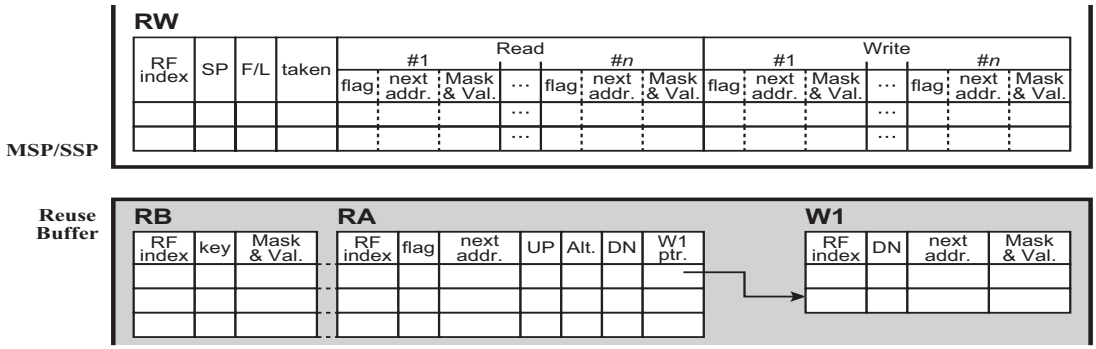

図 3 再利用表の構成

図 3 に，RW，RB，RA，W1 の構成の概略を示す。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，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 (Val., Mask) 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。すなわち RB は，図 1 で示したツリーの各エッジを格納し，RA が各ノードを格納する。このうち連想検索の対象となる RB を CAM で構成し，RA の各行は RB の各行と一対一に対応させるる。

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 (Val., Mask) 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，再利用テストを軽減するための拡張であり，これについては 4.4 節で述べる。実際に RB，RA への格納例を，**図 4** に示す。

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 5) 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。

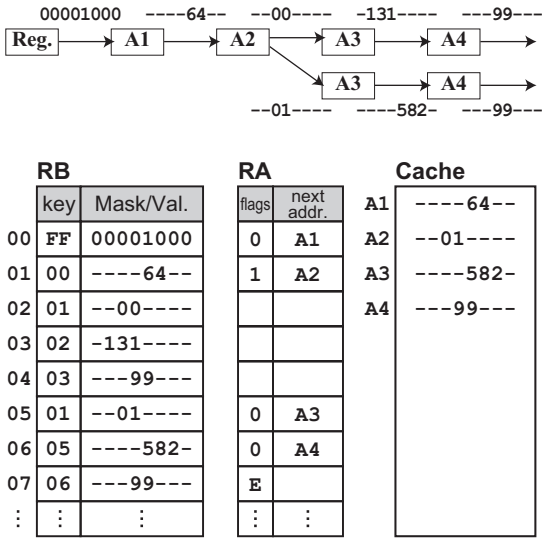

図 4 入力セットの RB/RA への格納

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 (W1 ptr.) 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，その内容を一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

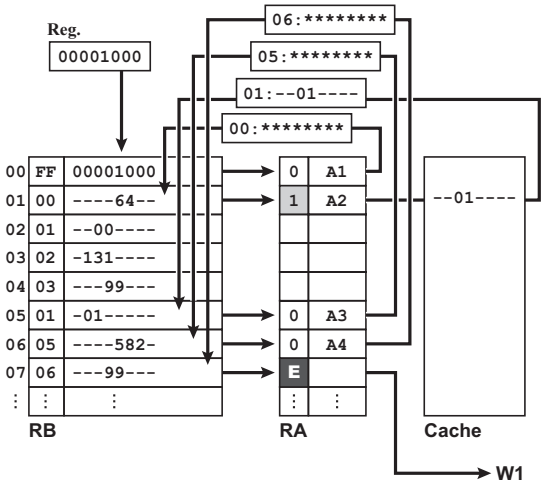

図 5 再利用テストの手順

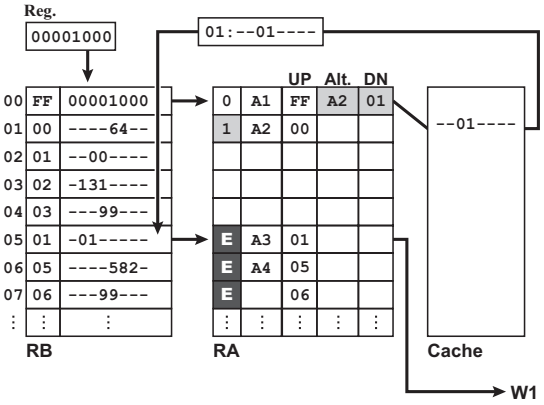

図 6 主記憶テストを削減した場合の動作

ると考えられる。

4.2 検索手順

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 5** 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 であるが，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。

4.3 エイジングとエントリ削除

再利用表は有限であるため，適宜エントリを削除する必要がある。エントリの削除は，LRU に基づき行う。このために，RB の各行にタイムスタンプを保持する 4bit のフラグを用意し，これを用いてエントリのエイジングを行う。具体的には，システム全体の時刻を表すサイクリックな 4bit カウンタを用意し，命令区間実行の際に，このカウンタをインクリメントする。インクリメント後に，カウンタの値と同じタイムスタンプを持つ RB 行を検索し，それらの行を最も古い行とみなして削除する。

命令区間に対し再利用が行われた場合には，その入力セットを構成する RB 行，すなわち参照された全ての RB 行のタイムスタンプを，現カウンタの値にセットする。これにより，ツリー構造を構成する各ノード (RB 行) において，必ず親ノードのタイムスタンプが子ノードのタイムスタンプと同じ，もしくはより現時刻に近くなることが保証できるため，上述した削除の際には必ずサブツリーのみが削除され，全体としてツリー構造に矛盾が生じることはない。

4.4 再利用テストの軽減

前項までで，再利用表構成の基本的な考え方を述べたが，キャッシュ参照の有無にかかわらず連想検索の回数は一定数必要となる。たとえば図 5 の例では，実際に主記憶から読みだして比較すべき入力は 1 つであるにもかかわらず，RB 検索は 5 回必要となっており，これは非効率的である。主記憶値テストの必要がない行に関しては，そもそも連想検索自体を省略したい。これを実現するために RA の構成を拡張する。具体的には，RA の各エントリに，

UP：親 RB 行のインデクス
Alt.：次に検索すべきアドレス
DN：次に検索すべきインデクス

の 3 項目を追加する。これにより図 5 に示した例の構成と動作は**図 6** のように変わる。

比較必要フラグがオンであるエントリに関しては，図 5 と同様に検索を行うが，比較必要フラグがオフであるエントリに関しては，Alt. および DN 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。

アドレス A4 に対しストアが発生したときの Alt., DN に対する操作を**図 7** に示す。まず RA から A4 を連想検索し，マッチした行の比較必要フラグを立てる。次にその行の UP 項が示す RF の行（図 7 では 05）に関し，Alt. 項を A4 で，DN 項を A4 の格納されている RF の行インデクス（図 7 では 06）で埋める。これにより，次回検索時には，ストア前には行われなかった A4 の内容に対する比較が行われるよう

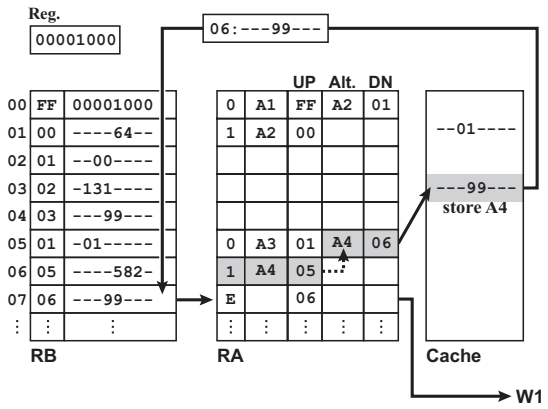

図 7 ストア時の操作 (破線) と次回から追加される検索

になる。

5. オーバヘッド評価機構

前章で述べた方法を用いることで，汎用 3 値 CAM による再利用表の実現が可能となる。しかし，CAM 自体の検索回数が増えるため，従来仮定してきた幅広の CAM を用いる場合よりも，再利用表操作のためのオーバヘッドは増大すると予想される。

再利用表操作のオーバヘッド削減については，特に命令区間の入力としての主記憶値参照を削減する方法[5] を既に提案しているが，本章ではこれに加えて，再利用オーバヘッドのコスト評価機構の追加を提案する。

5.1 概　　要

これまで並列事前実行機構では，プログラム中に出現する全ての命令区間を，事前実行および再利用の対象としてきた。このため，再利用によって得られる効果が小さいような短い命令区間においては，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ヘッドの方が大きいという場合も存在していたと考えられる。

そこで，命令区間に対し，再利用の効果があるか否かを事前に判断し，効果が得られないと判断された命令区間は，再利用の対象としないことにする。これにより，無駄な再利用を削減することができ，プログラム全体としての高速化が図れると考えられる。

具体的には，命令区間の再利用により削減できるステップ数と，その再利用に必要となるオーバヘッドについて概算を行う小さなハードウェアを再利用表に付加する。これにより，再利用効果の評価が行え，実際に効果が得られると考えられる命令区間についてのみ再利用を行うことができる。

5.2 オーバヘッドの算出

並列事前実行では，SSP による投機実行の対象とする命令区間をいかに選択するかが重要である。命令区間のうちでも，MSP による登録頻度が高く，SSP が登録したエントリの再利用頻度も高いものが，最も並列事前実行による効果が得られる。再利用機構では，この動的に変化する登録頻度や再利用頻度を把握するために，一定期間における登録および再利用の状況を，シフトレジスタを用いて記録している。そこで，この記録をオーバヘッドの算出に利用することにする。

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} は，上記のシフトレジスタから得られる。これらの値と当該命令区間の過去の省略ステップ数 S から，実際に削減できたステップ数を

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

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。

また，再利用が行われなかった場合でも，再利用表の検索オーバヘッドは存在する。このオーバヘッドは，

$$(T - N^{MSP} - N^{SSP}) \times Ovh^R \quad (2)$$

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，削減できたステップ数 (1) が大きいような命令区間は，再利用の効果が得られると考えられる。よって再利用表に小さなハードウェアを付加することによってこれを計算し，再利用の効果が得られると判断された命令区間に対してのみ再利用表への登録および再利用を行う。

なお，過去に再利用が行われたことがあり，再利用効果が得られる命令区間であっても，最近の T の間に再利用が一度も行われなかった場合 $N^{MSP}+N^{SSP}=0$ となってしまい，再び再利用対象とならなくなってしまう。このため，$N^{MSP}+N^{SSP}=0$ が検出された際は，N^{SSP} を T で初期化することによって，これを回避する。

6. 評　　価

以上で述べたオーバヘッドコスト評価機構を付加した，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。また，比較対象として，既存の予備実行技術を用いる場合の実装も行い，helper スレッドによるプリフェッチの効果と本手法との比較を行った。

本章では，単命令発行シミュレータを用いた評価により，本稿で提案する汎用 CAM による再利用表の実現手法が，従来仮定してきた幅広 CAM による手法よりも非常に小さいハードウェア量で実装可能であるにもかかわらず，幅広 CAM を用いた場合とほぼ同等の性能が得られることを示す。また，二次キャッシュの仮定により，我々の並列事前実行機構が，既存手法である予備実行と同様に効果的なプリフェッチ機構として働くことを示し，結果として予備実行よりも優れた性能が得られることを示す。

表 1 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
Cache ミス ペナルティ	10 cycles
共有 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
Cache 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
Read アドレス	256 word/RW
Write アドレス	256 word/RW
RF エントリ数	256
RB/RA/W1 エントリ数	4096
SSP 台数	3

6.1 シミュレーションモデル

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。再利用表の構成は，図 3 に示したとおりである。評価に用いた各パラメータを**表 1** に示す。なお命令レイテンシは，SPARC64-III[7] を参考にしている。

ハードウェア構成に関しては，一次キャッシュを 32KB : 4way とし，共有二次キャッシュは 2MB : 4way とした。また，一次キャッシュ，共有二次キャッシュ，主記憶のレイテンシはそれぞれ，2cycle，10cycle，100cycle と仮定した。再利用表については，まず RW を，32Byte 幅 ×256 エントリ ×4 セットで，一次キャッシュと同サイズの 32KB とし，レイテンシも一次キャッシュと同じと仮定した。RB についても，二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 10cycle/100cycle 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 100cycles 後以降は MSP/SSP 内の D1 上で hit すると仮定した。

6.2 Stanford

まず Stanford ベンチマークを，gcc-3.0.2 (-O2 -msupersparc) 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。ただし，再利用による効果が必要以上に高く現れないように，Queens および FFT において，単純繰り返しを行っている最外ループは各 1 回に変更してある。5 章で述べたオーバヘッド評価機構の効果を見るため，最初に，オーバヘッド評価機構を用いない場合の結果を

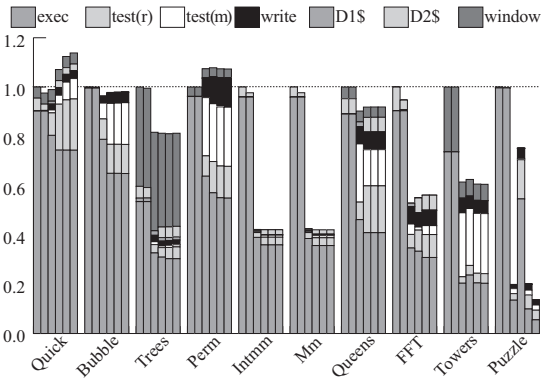

図 8 MSP の実行サイクル数
(Stanford，コスト評価なし，$1\tau/2\tau$)

図 8 に示す。なお，RB を構成する CAM のレイテンシとして，レジスタとの比較に 32Byte/cycle，キャッシュとの比較に 32Byte/2cycle (このパラメータセットを以下，$\mathbf{1\tau/2\tau}$ 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256×32=64MB)
(G_4) 汎用CAM(幅256bit× 深さ4K=128KB)
(G_{64}) 汎用CAM(幅256bit× 深さ64K=2MB)
(G_{256}) 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。

CAM の構成は，MOSAID 社の DC18288[8] を参考にしている。DC18288 は幅 288bit× 深さ 64K=2MB の大きさを持つ CAM であり，(G_4)，(G_{64}) で仮定した大きさは現実的なものである。また，二次キャッシュサイズを増大させた場合は二次キャッシュミスペナルティが減少するだけであるが，再利用表サイズは増大させることにより実行サイクル数自体を減らすことが可能である。つまり，二次キャッシュと再利用表を同じレイテンシと仮定するなら，二次キャッシュよりも再利用表のほうが，物量をかけることにより得られる効果は大きくなる。よって，再利用表にも二次キャッシュと同等の物量を仮定することは，評価を行う上で現実的である。

グラフ中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を表している。

まず，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 (G_4) に関しては，Puzzle で

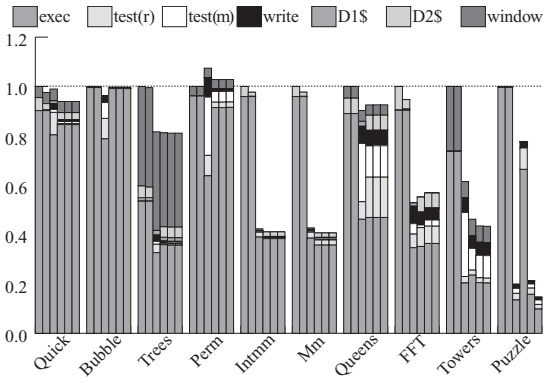

図 9 MSP の実行サイクル数
(Stanford，コスト評価あり，$1\tau/2\tau$)

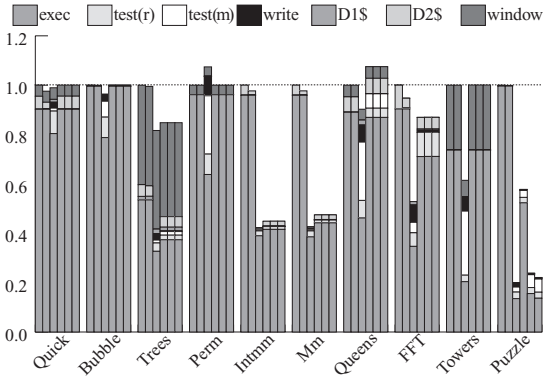

図 10 MSP の実行サイクル数
(Stanford，コスト評価あり，$9\tau/10\tau$)

は理想値である (W) に大きく及ばなかった。グラフの exec が増大していることより，RB 容量の不足が再利用のヒット率低下を引き起こしていることが分かる。しかし，Bubble，Queens，FFT での速度低下は僅かであり，またその他のベンチマークでは (W) と同等以上の高速化が得られた。

また，従来の 1/32，1/8 のハードウェア量で実現できる (G_{64})，(G_{256}) では，Bubble，Queens，FFT ではやはり僅かに (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。押し並べて，(W) と比較するとやはり test 部は微少増加しているが，それにも増して exec が減少しているものが多い。これは，汎用 CAM を用いることでハードウェア量を削減できているにもかかわらず，RB の記憶域の無駄がなくなったことにより，記憶できる実質的な入出力セットが増大し，再利用率が向上していることを示している。

次に，オーバヘッド評価機構を用いた場合の結果を**図 9** に示す。Bubble では僅かに悪化しているものの，Quick，Perm，Towers において顕著な高速化が見られ，特に，図 8 と比較すると Quick，Perm の結果より，再利用を適用することで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のが分かる。また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，(G_4)，(G_{64})，(G_{256}) 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。

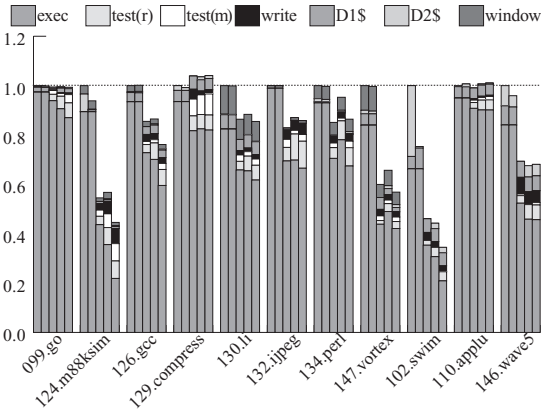

図 11 MSP の実行サイクル数
(SPEC95，コスト評価なし，$1\tau/2\tau$)

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/9cycle，キャッシュとの比較に 32Byte/10cycle (以下，$\mathbf{9\tau/10\tau}$ と記す) を仮定した。この結果を**図 10** 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。また，再利用テストのレイテンシの増大により，文献 5) の主記憶値テスト削減手法の効果も大きくなっていると考えられる。平均サイクル数削減率においても，(G_4)，(G_{64})，(G_{256}) はそれぞれ 21%，27%，28% となり，良好な結果となった。

6.3 SPEC95

次に，SPEC95 による結果を示す。Stanford と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 11**，**図 12** に示す。グラフは左から (M)，(P)，(W)，(G_4)，(G_{64}) である。

図 11 では Stanford の Puzzle と同様，(G_4) では 134.perl および 147.vortex において RB の不足により exec が増大し，(W) に劣る結果となっているが，(W) の 1/32 のハードウェア量で実現できる (G_{64}) では，これを解消できている。また図 12 では 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，(G_4)，(G_{64}) は 23%，29% という値となり，

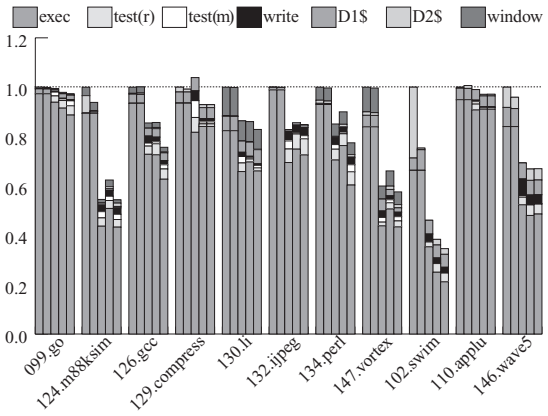

図 12 MSP の実行サイクル数
(SPEC95，コスト評価あり，$1\tau/2\tau$)

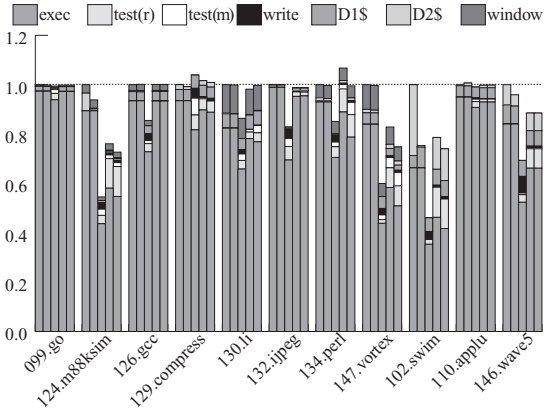

図 13 MSP の実行サイクル数
(SPEC95，コスト評価あり，$9\tau/10\tau$)

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 13** に示す。全体的に性能は低下し (W) にはかなり劣るものの，オーバヘッド増大による速度低下は回避できている。平均サイクル数削減率は，(G_4)，(G_{64}) それぞれ 7%，10%となり，やはり (P) の 4% を上回った。

また，予備実行のキャッシュミス削減率には及ばないものの，128.m88ksim や 102.swim などにおいて，共有二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 による評価結果にはほとんど現れていなかったが，本再利用機構が共有二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った。

7. おわりに

本稿では，並列事前実行を用いた非対称の投機的マルチスレッディングにおいて，その構成要素である再利用表を，汎用 3 値 CAM を用いて実現する方法について述べた。汎用 CAM は 256bit 程度の幅しか持たないため，命令区間の入出力セットを折畳んで格納する必要があることを述べ，その具体的方法について説明した。また，この格納方法により再利用表参照のオーバヘッドが増大することを予想し，再利用オーバヘッドを評価する機構を提案した。この機構を用い，オーバヘッド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抑止する手法を提案し，Stanford および SPEC95 により評価を行った。

その結果，汎用 CAM を用いて再利用表を構成した場合，幅広 CAM を仮定した場合の 1/512〜1/8 のハードウェア量で，幅広 CAM に匹敵する高速化が図れることを示した。また，オーバヘッド評価機構により，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を示した。また，本機構により共有二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。

今後は，入力セットの木構造を複数のサブツリーに分割するなどして，CAM の高いスループットをより効果的に利用する方法について検討したい。

参 考 文 献

1) Collins, J. D., Wang, H., Tullsen, D. M., Hughes, C., Lee, Y., Lavery, D. and Shen, J. P.: Speculative Precomputation: Long-range Prefetching of Delinquent Loads, *28th ISCA*, pp. 14–25 (2001).
2) Luk, C.: Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors, *ISCA'01*, pp. 40–51 (2001).
3) Collins, J. D., Tullsen, D. M., Wang, H. and Shen, J. P.: Dynamic Speculative Precomputation, *34th MICRO*, pp. 306–317 (2001).
4) Roth, A. and Sohi, G.S.: A quantitative framework for automated pre-execution thread selection, *35th MICRO*, pp. 430–442 (2002).
5) 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).
6) MUSIC SEMICONDUCTORS: *Using The MU9C1965A LANCAM MP For Data Wider Than 128 Bits* (1998).
7) HAL Computer Systems/Fujitsu: *SPARC64-III User's Guide* (1998).
8) MOSAID Technologies Inc.: *Feature Sheet: MOSAID Class-IC DC18288*, 1.3 edition (2003).