Vol.2011-ARC-196 No.1
2011/7/27

命令区間の特徴を用いた自動メモ化プロセッサの再利用率向上手法

山田龍寛[†1] 小田遼亮[†1] 池谷友基[†1]
津邑公暁[†1] 松尾啓志[†1] 中島康彦[†2]

我々は，計算再利用技術に基づく自動メモ化プロセッサ，および，これに値予測に基づく投機マルチスレッド実行を組合わせた並列事前実行を提案している．自動メモ化プロセッサは関数とループを再利用対象区間としているが，命令区間によってその実行回数や再利用回数，登録に要するエントリ数はさまざまである．本稿では，命令区間毎の特徴から再利用が成功するかを予測し，無益なエントリセットの登録を抑制する手法と，命令区間毎の登録量の違いを考慮して，1 つの区間のエントリが表を占有しないように，使用エントリ数に上限を設定する手法を提案する．SPEC CPU95 を用いてシミュレーションにより評価した結果，従来モデルでは最大 40.5%，平均 9.6%であったサイクル削減率が，前者では最大 41.6%，平均 13.7%，後者では最大 41.8%，平均 11.9%まで向上することを確認した．

Improvements for Auto-Memoization Processor by Considering Instruction Region Characteristics

Tatsuhiro Yamada,[†1] Ryosuke Oda,[†1]
Tomoki Ikegaya,[†1] Tomoaki Tsumura,[†1]
Hiroshi Matsuo[†1] and Yasuhiko Nakashima[†2]

We have proposed an auto-memoization processor based on computation reuse, and merged it with speculative multithreading based on value prediction into a parallel early computation. Instruction regions, such as functions and loops, will be applied computation reuse, and different regions will have different characteristics. This paper proposes a way of saving the wasted entry sets by predicting whether an instruction region is reusable or not from its behavior. This paper also proposes a way of limiting the number of usable entries for an instruction region by considering its usage. The result of the experiment with SPEC CPU95 suite benchmarks shows that the first method improves the maximum speedup from 40.5% to 41.6%, and the average speedup from 9.6% to 13.7%, and the second improves the maximum speedup to 41.8%, the average to 11.9%.

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，微細化による高クロック化で高速化を実現できた．しかし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高速化を実現しにくくなったことで，SIMD やスーパスカラなどの命令レベル並列性に基づく高速化手法が注目された．また，近年は電力効率と性能向上を両立させる観点から，複数コアを搭載したマルチコアプロセッサが主流となりつつあり，今後集積度の向上に伴なってコア数も増大していくと考えられている．これらの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．

一方我々は，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を提案している．[1)2)] これは，実行時に関数の入出力を表に記憶しておき，再び同関数が呼び出されたときにその入力が表上の入力と一致すれば，出力を書き戻すことで関数の実行を省略し，高速化を達成する手法であり，従来の高速化手法とは着眼点が異なっている．また我々は，ループイタレーション等の命令区間のうち入力が単調変化するものに対し，入力を過去の履歴から予測し，その予測された値を用いて命令区間を別コアで予め実行しておくことで出力を生成・記憶する並列事前実行と呼ぶモデルを提案している．これにより，予測が正しかった場合はメインコアによる当該イタレーションの実行が計算再利用により省略できる．

本稿では，命令区間毎の特徴から再利用が成功するかを予測し，無益なエントリセットの登録を抑制する手法と，命令区間毎の登録量の違いを考慮して，1 つの区間のエントリが表を占有しないように，使用エントリ数に上限を設定する手法を提案する．前者は，登録するエントリを選別することで，有益なエントリがより多くの再利用表の領域を使用することができるようになり，再利用率の向上と無益なエントリの検索によって発生していたオーバヘッドを削減することができると考えられる．また後者は，命令区間毎の登録回数や登録頻度，登録に要するエントリ数の差異によって生じる悪影響を抑制することで，再利用の成功率が高まると考えられる．

†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

Vol.2011-ARC-196 No.1
2011/7/27

2. 研究背景

本章では，本研究の背景となる自動メモ化プロセッサおよび並列事前実行機構について述べる．

2.1 自動メモ化プロセッサ

計算再利用（Computation Reuse）とは，主に関数などの命令区間に対してその入力と出力の組を実行時に記憶しておき，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，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，高速化を図る手法である．また，それら命令区間に計算再利用を適用することを**メモ化（Memoization）**[3]と呼ぶ．これまで，ハードウェア[4][5]によるものやソフトウェア[6]によるものなど，様々なメモ化手法が提案されている．

メモ化は元来，高速化のためのプログラミングテクニックであるが，我々が提案している**自動メモ化プロセッサ（Auto-Memoization Processor）**は，既存バイナリをメモ化実行可能なプロセッサである．実行時に動的に関数およびループイタレーションを再利用可能命令区間として検出し，実行時にその入出力を**再利用表**と呼ぶテーブルに保存する．`call` 命令のターゲットから `return` 命令までの区間を関数として，また，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イタレーションとして検出する．その後，再度同じ命令区間を実行しようとした際には，再利用表を検索し，現在の入力セットが過去のものと一致した場合，出力を再利用表からレジスタおよびキャッシュに書き戻すことで，当該命令区間の実行を省略する．

自動メモ化プロセッサは主に，メモ化制御機構，再利用表 **MemoTbl**，および MemoTbl への書込みバッファとして働く **MemoBuf** から構成される．命令区間実行開始時には MemoTbl を参照し，過去の入力との一致比較を行う．一致するエントリが存在した場合，対応する出力が書き戻され，命令区間の実行は省略される．一致するエントリが存在しなかった場合，入出力を MemoBuf に格納しつつ当該命令区間を通常実行し，実行終了時に MemoBuf の内容を MemoTbl に格納することで将来の再利用に備える．なお，入力には関数の引数はもちろんのこと，当該命令区間で発生した主記憶参照も全て含まれる．また出力には，関数の返り値および当該命令区間で発生した主記憶書込みが含まれる．

MemoTbl は，命令区間の開始アドレスを記憶する RF，入力値を記憶する RB，入力アドレスを記憶する RA，そして出力値を記憶する W1 の 4 つの表から構成されている（**図 1**）．

RF は，各再利用対象命令区間に対応する行を持っており，メモ化のためのフィールドおよび後述するオーバヘッドフィルタのためのフィールドを持っている．メモ化のためのフィールドには，関数およびループの別 (F or L)，命令区間開始アドレス (addr)，また後述する並列事前実行の入力ストライド予測に用いるための直近の入力値セット (prev. inputs) が記憶される．また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)，過去の再利用に要した入力検索および書き戻しオーバヘッド (Ovh read/write), 過去の再利用ヒット履歴 (hit hist) が保持される．

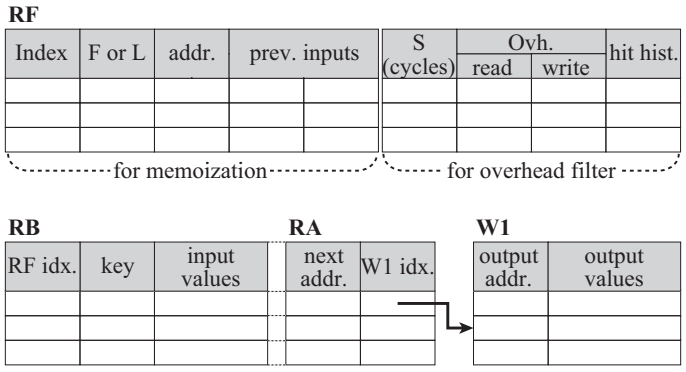

図 1 MemoTbl の構造
Fig. 1 Structure of MemoTbl

紙面の都合上，詳細は文献 1)，2) に譲るが，MemoTbl 検索手順の概要は以下の通りである．命令区間を検出するとまず対応する RF エントリを検索し，RF のインデクスと同じ値を RF idx フィールドに持ち，現在のレジスタ上の入力値を持つエントリを RB から検索する．RB の各エントリのインデクスは，次入力アドレスを格納する RA エントリへのインデクスと対応づけられている．RB 内で入力が一致するエントリが存在した場合，そのマッチ行と同一インデクスを持つ RA エントリから得た次入力アドレスを用いてキャッシュを参照し，次入力値を得る．この入力値を用いて再び RB を検索する．これを繰り返し，全ての入力の一致が確認できると，入力セットの終端を保持する RB エントリは W1 へのインデクスを保持しており，これを用いて W1 を参照して出力値を得，これをレジスタおよびキャッシュに書き戻すことで命令区間の処理を省略する．RB は 3 値 CAM で実装することにより高速な連想検索を実現している．入力および出力を記憶する RB,RA,W1 をこれ以降まとめて入出力表と呼ぶ．

このように自動メモ化プロセッサは計算再利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手法であるが，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値を 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．よって，命令区間の実行コストが非常に小さい場合や，入力一致比較のヒッ

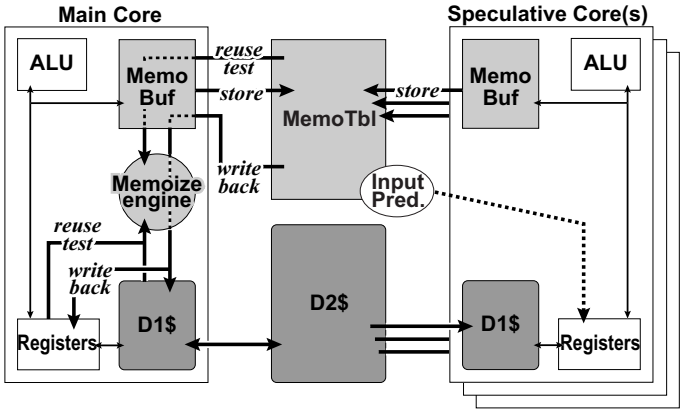

図 2 並列事前実行機構
Fig. 2 Structure of a parallel early computation.

ト率が低い場合には，性能が悪化してしまう場合もある．

2.2 並列事前実行機構

前節で述べた自動メモ化プロセッサは計算再利用に基づく手法であり，当然ながらある命令区間を過去に完全に同一の入力セットで実行したことがある場合にのみ効果が得られる．よってイタレータ変数を入力のひとつとして扱うループイタレーションでは，全く効果が得られない．

そこで，計算再利用を行いながら実行を進めるメインコアとは別に，値予測に基づいて同一命令区間をメインコアに先がけて実行する投機実行コアを複数備えるシステムを考える．これを我々は**並列事前実行**と呼んでいる．以下この投機実行コアを **SpC（Speculative Core）**と呼ぶこととする．プロセッサは複数の SpC を用いて構成可能である．**図 2** に，並列事前実行機構の概要を示す．

各 SpC は，それぞれ MemoBuf と一次キャッシュを持ち，二次キャッシュは全コアで共有するものとする．メインコアが計算再利用可能な命令区間を実行する際，SpC はこれに並行して，予測された入力値セットを用いて同一区間を実行する．そして，その実行に使用した入力値セットおよび実行の結果得られた出力値セットを，共有 MemoTbl に登録する．値予測が正しかった場合，メインコアが次に実行しようとする命令区間は既に SpC により実行済であり MemoTbl に結果が格納されているため，実行を省略できる．また値予測が誤っていた場合も，メインコアは当該区間を通常実行するだけであるので，MemoTbl 検索のコストは発生するものの，投機実行ミスに起因するオーバヘッドは発生しない．

値予測アルゴリズムにはさまざまな複雑な手法を用いることも可能であるが，必要となる追加ハードウェア量等の観点から，現在は直近の過去 2 回の実行に用いられた入力値セットの差分に基づく，単純なストライド予測を用いることを想定している．

2.3 オーバヘッドフィルタ機構

2.1 節で述べたように，計算再利用のためのオーバヘッドが大きい場合には，メモ化適用により却って性能が悪化する場合もある．また並列事前実行では，SpC 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．そこで RF では，各命令区間に対し一定期間における再利用の状況をシフトレジスタ（図 1 中 hit hist）を用いて記録し，これを用いてそれぞれの命令区間の再利用適応度を算出している．

ある命令区間について，最近の一定回数 T 回の再利用試行における再利用成功回数 M は上記シフトレジスタから得られる．この値と，当該命令区間の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

$$M \cdot (S - Ovh^R - Ovh^W) \tag{1}$$

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 からキャッシュ等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，MemoTbl の検索オーバヘッドは存在する．このオーバヘッドは，

$$(T - M) \cdot Ovh^R \tag{2}$$

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．すなわち，式 (1) から式 (2) を引いた値である

$$M \cdot (S - Ovh^W) - T \cdot Ovh^R \tag{3}$$

が正値であれば再利用の効果があると判断できる．MemoTbl 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 MemoTbl への登録および再利用を行っている．

3. 提 案 手 法

本章では，再利用対象としている命令区間毎の特徴を用いる 2 つの再利用モデルを提案する．

3.1 検索回数による再利用予測モデル

MemoTbl には過去に実行された結果に加え，この先実行されると考えられる命令区間の入出力が SpC によって登録されている．しかし，命令区間によっては再び同一の入力で呼び出さ

れることがないものや，入力を予測し事前実行してもその予測が外れて再利用がまったく成功しない命令区間も少なくない．こういった，MemoTbl に登録しても無益なエントリを登録してしまうことで，有益なエントリを追い出してしまい再利用効率が低下する場合がある．さらに，RF に登録されている命令区間を実行する際，再利用の成否に関わらず入出力表を検索するため，再利用が成功しないような命令区間は再利用検索オーバヘッドだけが増加してしまう．

そこで，命令区間毎に再利用が成功するかを予測して，成功しないと考えられる命令区間の登録を事前に抑制する手法を提案する．無益なエントリの登録を減らすことで，再利用の成功しやすい命令区間がより多くの MemoTbl の領域を使用することができ，再利用率が向上すると考えられる．

再利用成功予測の判断規準として，命令区間毎の検索回数に着目する．再利用対象の命令区間を検出すると，その区間の実行前に MemoTbl の検索が行われる．この検索の回数は命令区間毎に様々で，再利用の成功と失敗回数によっても変動する．また，ある命令区間の登録を始めてから，少ない検索回数で 1 度目の再利用が成功した場合，今後も再利用が成功しやすいと考えられる．一方で，1 度目の再利用の成功が遅い場合，再利用は成功しづらく，今後頻繁に成功したとしても，最終的な再利用回数は多くならないと考えられる．

そこで，命令区間毎に，初めて再利用が成功したときの検索回数と再利用回数の関係を調査した．その結果，関数，ループ共に初回再利用時の検索回数が少ない程再利用が成功しやすいという傾向があることがわかった．この関係を基に閾値を設定し，それを超える検索を行ったにも関わらず，再利用が成功していない命令区間は今後再利用が成功しにくいと判断し，入出力エントリの MemoTbl への登録を行わないようにする．

3.2 命令区間毎の使用上限設定モデル

関数やループはプログラム中のいたる所に存在し，その実行内容もさまざまである．そのため，命令区間毎に登録される入出力セット数や，1 つの入出力を MemoTbl に登録するのに必要となるエントリ数も様々である．また，プログラムの流れによっては，ある特定の命令区間の入出力の登録が頻発することによって，入出力表をその 1 つの命令区間の入出力が占有してしまうことがある．例えば，2 重ループでは，内側の命令区間のエントリがループの回数だけ登録され，その後外側のループのエントリがひとつ登録される，という工程を繰り返す．そのため，内側のループが実行され多数のエントリが登録されると，外側のループのエントリは再利用テストを行う前に MemoTbl から追い出されてしまい，再利用することができなくなる場合がある．

そこで，特定の命令区間が MemoTbl を占有してしまうのを防ぐために，各命令区間が登録できるエントリの上限を設定する手法を提案する．上限を設定することで，命令区間毎の登録回数や，登録頻度，1 入力セットあたりに要するエントリ数の差異によって生じる悪影響を抑制する．ある命令区間を MemoTbl に登録するときに，当該命令区間が入出力表に占めるエントリ数およびエントリセット数を各上限値と比較し，共に上限値を超えていた場合当該命令区間に対して LRU に基づいた入出力エントリのパージを行う．

1 セットあたりに要するエントリ数が多い命令区間の場合，エントリ数が上限を超えたときでもエントリセット数は上限を超えていない場合が存在する．エントリ数の上限値のみ設定してしまうと，多くのエントリ数を必要とする入出力は少ないセット数しか MemoTbl に保持しておくことができなくなってしまう．そのため，セット数の上限値も設定することで，こうした場面を回避している．しかし，セット数が上限を超えていなくても，登録しているエントリ数が上限値を大きく上回ってしまう事態が発生する．それを防ぐために，登録エントリ数の上限値をもうひとつ用意し，その上限値を超えてエントリを登録している場合には，セット数が上限に達していなくても当該命令区間に対して LRU に基づいた入出力エントリのパージを行う．

4. 実　　装

前章で述べた 2 つの再利用モデルを実現するために行ったそれぞれの実装について説明する．

4.1 検索回数による再利用予測モデル

3.1 節で述べた提案モデルを実現するためには，命令区間毎のこれまでの削減ステップや再利用オーバヘッドの履歴だけではなく，命令区間毎の検索回数と再利用成功の有無を記憶しておく必要があるため，RF に新たにフィールドを 2 つ追加してそこに保持しておく．そして，エントリを登録する際に，RF に保持している検索回数と閾値 TRYTHR を比較し，検索回数が一定値を超えているにも関わらず再利用が一度も成功していない命令区間は，今後エントリ登録を行わないようにオーバヘッドフィルタを拡張する．

さて，SpC によって事前実行が行われる場合，メインコアに先んじて当該命令区間を実行することになる．2.2 節で述べたように，メインコアと SpC は二次キャッシュを共有しているため，事前実行中にメモリへのアクセスを要する命令が存在する場合は，SpC はメモリから二次キャッシュへ値を読み出してくることになる．その結果，SpC がメインコアに先んじてキャッシュミスを発生させるため，メインコアのメモリへのアクセスレイテンシの一部を隠蔽している．このように，並列事前実行は，値予測によってループの再利用を試みるだけではなく，限定的ではあるがキャッシュプリフェッチの効果をもたらす．

しかし，オーバヘッドフィルタの拡張によってある命令区間に関する登録を完全に停止してしまうと，ストライド予測のための履歴が更新されず，並列事前実行が行われなくなってしまう．そのため，事前実行によるプリフェッチ効果が得られなくなり，メインコアの二次キャッ

シュミスが従来に比べ増加する可能性がある．そこで，メインコアは，エントリ登録を停止してもストライド予測のための履歴は更新し，SpC は並列事前実行を行うがその結果を MemoBuf へと登録しないように動作させることで，この問題に対処することとした．

4.2 命令区間毎の使用上限設定モデル

3.2 節で述べた手法を実現するために，まず命令区間毎の現在のエントリ使用数と登録エントリセット数を把握する必要がある．そのため，命令区間を記憶しておく RF を拡張し，新たに *#entry* と *#set* フィールドを追加する．

#entry は，当該命令区間の入力エントリが登録されるたびにインクリメントし，パージが行われるたびにデクリメントすることで，命令区間毎の現在の使用エントリ数を保持する．*#set* は，エントリセットの終端エントリが登録されるたびにインクリメントし，パージが行われるたびにデクリメントすることで現在の登録エントリセット数を保持する．さらに，新たにレジスタを 3 つ追加して，登録エントリの上限値 *ENTRYMAX*，*ENTRYMAXOVER* と登録セット数の上限値 *SETMAX* を格納しておく．*ENTRYMAXOVER* は，*ENTRYMAX* の 1.5 倍の値を格納している．

エントリを登録しようとする前に，RF の *#entry* フィールドと *ENTRYMAX* を，RF の *#set* フィールドと *SETMAX* をそれぞれ比較し，共に上限値を超えていた場合，その命令区間はエントリを登録しすぎていると判断し，当該命令区間のエントリを追い出す．この判定で，2 つとも上限値を超えていない場合でも，*#entry* フィールドが *ENTRYMAXOVER* の値を上回っている場合は，登録エントリが多すぎると判断し，当該命令区間のエントリを追い出す．最終的な上限値の判定式は以下に示す通りである．

$$((\#entry > ENTRYMAX) \&\& (\#set > SETMAX)) \\ \| (\#entry > ENTRYMAXOVER) \quad (4)$$

また，*#entry* と *#set* により現在の登録エントリ数と登録セット数が命令区間毎にわかるので，登録エントリ数が 0 の命令区間は RB を検索しに行かないように拡張した．これにより，さらなる検索オーバヘッドを削減できると考えられる．

5. 評　　価

5.1 評 価 環 境

以上で述べた 2 つの拡張をそれぞれ自動メモ化プロセッサシミュレータに追加実装し，サイクルベースシミュレーションによる評価を行った．シミュレータは単命令発行の SPARC V8 アーキテクチャをベースとしている．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[7] を参考とした．また，MemoTbl の RB を構成する 3 値 CAM は，ルネサステクノロジ社の R8A20410BG[8] を参考にし，プロセッサのクロック周波数が CAM のクロック周波数の 10 倍と仮定して検索オーバヘッドを見積もっている．

表 1 シミュレータ諸元
Table 1 Simulation Parameters

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
RB サイズ	32bytes × 4K 行 (32KBytes)
レジスタ ⇔ CAM	9cycles/32bytes
メモリ ⇔ CAM	10cycles/32bytes
CAM ⇒ レジスタ or メモリ	1cycle/32bytes

5.2 評 価 結 果

SPEC CPU95 (train) の 12 のプログラムを 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て 2 つの提案モデルを順に評価した．

5.2.1 検索回数による再利用予測モデル

評価は，メモ化を行わないモデル，従来モデル，再利用予測モデルについて行った．全てのモデルはメインコア 1 つに加え，SpC 3 つの合計 4 コア構成とした．**図 3** 中で各ベンチマークプログラムの結果を 3 本のグラフで示しているが，それぞれ左から順に

(N) メモ化を行わないモデル
(M) 従来モデル
(PS) 検索回数から再利用予測を行ったモデル

が要したサイクル数を表している．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，read は MemoTbl との比較に要した検索オーバヘッド．write は再利用成功時に発生する結果の書き戻しオーバヘッド，D$1, D$2, window はそれぞれ一次/二次キャシュミスペナルティとレジスタウィンドウミスペナルティである．なお，各サイクル数はメモ化なし (N) の結果を 1 とする正規化を行っている．

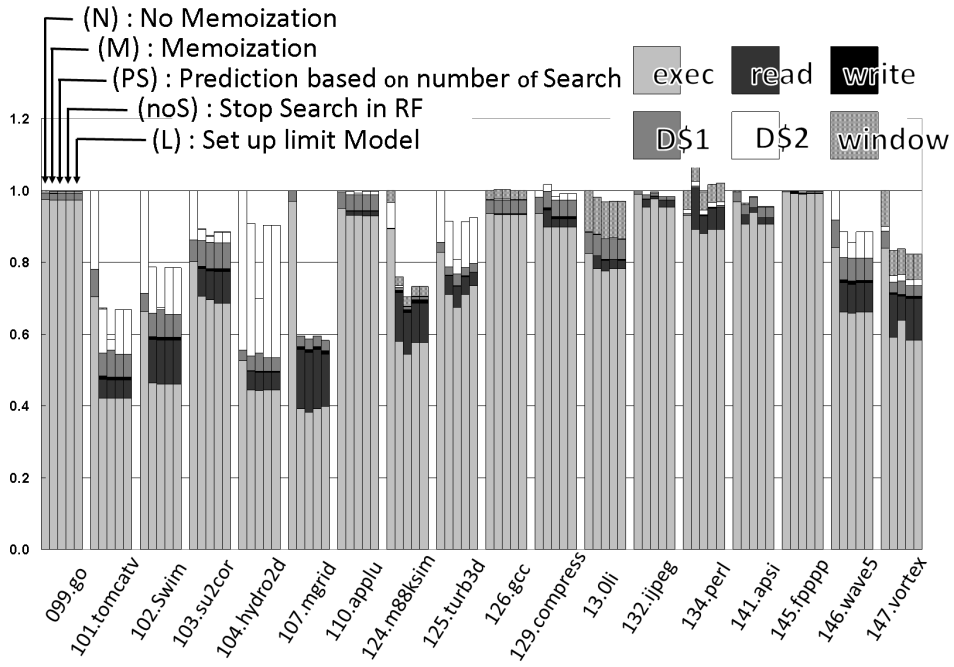

図 3 実行サイクル数比
Fig. 3 Ratio of cycles

4.1 節で述べた閾値 TRYTHR は，いくつかのパラメータで実行した結果 60 付近の値が最適だったため，60 と設定した結果を (PS) に示したが，今後パラメータについてはさらなる調査が必要である．

125.turb3d，124.m88ksim，130.li では exec を削減することができており，無益なエントリを登録しないことによって，有益なエントリが MemoTbl のより多くの領域を使うことができるようになったことで，再利用率が向上していると考えられる．また，129.compress，130.li，132.ijpeg は再利用オーバヘッドの大幅な減少によってサイクル削減率が向上している．これは，無益なエントリを登録しなかったことによって無駄な検索を行わずに済んだためだと考えられる．特に，129.compress と 134.perl についてはこれまで，再利用を適用することによって性能が悪化していたものが，(PS) では通常実行よりも高速化している．

一方，101.tomcatv，102.swim，104.hydro2d については，二次キャッシュミスを大きく削減することができており，104.hydro2d に至っては従来モデルに比べて 20%以上の高速化に成功している．これは，従来ではオーバヘッドフィルタによって登録が中止されていたために，SpC による並列事前実行が行われていなかった命令区間に対しても，検索回数が一定値以上に達しているならば 4.1 節で述べた SpC の動作によって事前実行が行われ，その結果 SpC によるプリフェッチ効果が高まったためと考えられる．

結果をまとめると，メモ化なし (N) に比べ，従来手法では平均で 9.6%，最大 40.5%のサイクル数の削減だったのに対し，(PS) では平均 13.7%，最大 41.6%のサイクル数の削減に成功した．

5.2.2 命令区間毎の使用上限設定モデル

登録エントリ上限設定モデルと *#entry* を利用した検索停止のみのモデルについても評価を行った．図 3 中の，右側 2 本のグラフは左から順に

(noS) RB にエントリが存在しない場合，RB の検索を行わないモデル (検索停止モデル)
(L) 命令区間毎の登録エントリ数の上限を設定したモデル

が要したサイクル数を表している．サイクル数の内訳は 5.2.1 節で述べたとおりである．

(noS), (L) 共に 102.swim, 103.su2cor, 124.m88ksim, 147.vortex の 4 つのベンチマークにおいて，exec を削減することができている．これは，これまで再利用が成功していなかったエントリに対して再利用が成功するようになったためだと考えられる．しかしながら，125.turb3d, 107.mgrid では exec が増加してしまっている．これは，命令区間毎に登録エントリ数を制限したために，これまでは登録できていたエントリセットを追い出してしまうようになったためである．ただし，107.mgrid は exec の増加量より検索オーバヘッドの削減量が多かったため，総実行サイクル数は削減できている．また，107.mgrid 同様に，141.apsi，129.compress，130.li，134perl の 4 つのプログラムは，exec は従来モデルに対してあまり変化がなかったが，検索オーバヘッドが削減できたため，総サイクル数は削減できている．

一方で，(L) と (noS) を比較すると，これら 2 つの総実行サイクル数はほとんど変わらず，上限を設定することからは期待した様な効果が得られなかった．そこで，これらの手法について Stanford ベンチマークの FFT，Intmm，Mm でも評価を行った．その結果を**表 2** に示す．

表 2 からわかるように，3 つのプログラム全てで，(L) が (noS) に比べてサイクル数を大きく削減できていた．FFT は 2 重ループを持つプログラムで，内側のループイタレーション回数が多いため，従来モデルでは外側のループのエントリを追い出してしまっていたが，(L) ではその外側のループも再利用が成功するようになったため，高速化している．Intmm と Mm は共に，再利用がほとんど成功しないループの中に，再利用が成功する関数が存在しているようなプロ

表 2 Stanford のサイクル削減率
Table 2 Reduced execution cycles(Stanford)

	FFT	Intmm	Mm
(M) 従来モデル	11.3%	27.2%	33.2%
(noS) 検索停止モデル	11.3%	24.3%	34.3%
(L) 上限設定モデル	14.0%	38.6%	37.9%

グラムである．このループと関数は 1 入出力あたりの使用エントリ数および 1 入出力セットの登録回数が多いため，従来モデルでは頻繁なエントリ追い出しが相互に行われてしまっていたが，上限を設定することで MemoTbl 内で 2 種類のエントリが共存できたため，高速化した．

結果をまとめると，SPEC CPU95 ではメモ化なし (N) に比べ，従来手法では平均で 9.6%，最大 40.5%のサイクル数の削減だったのに対し，検索停止と上限を設定を組み合わせた (L) では，平均 11.9%，最大 41.8%のサイクル数の削減に成功した．また，Stanford の 3 つのプログラムでは (N) に比べ，従来手法では平均で 23.0%，最大 33.2%のサイクル数の削減だったのに対し，(L) では，平均 30.2%，最大 38.6%のサイクル数の削減に成功した．

6. おわりに

本稿では，計算再利用技術に基づく自動メモ化プロセッサの再利用対象としている命令区間毎の特徴から再利用の成否を予測し，無益なエントリセットの登録を抑制する手法と，命令区間毎の登録量の違いを考慮して，1 つの区間のエントリが表を占有しないように，使用エントリ数に上限を設定する手法についてそれぞれ述べた．

SPEC CPU95 ベンチマークを用いて評価した結果, 従来モデルでは最大 40.5%, 平均 9.6%であったサイクル削減率が, 前者では最大 41.6%, 平均 13.7%, 後者では最大 41.8%, 平均 11.9%まで向上することを確認した．

今後の課題としては，まず上限を設定する手法をさまざまなパラメータで実行することで，プログラムごとの最適なパラメータを発見することがあげられる．また，SPEC CPU95 以外のさまざまなベンチマークプログラムによるシミュレーションを通じて，多様な命令区間の検索回数と再利用回数の関係をさらに調査することで，幅広いプログラムに適用可能な閾値を見つけることもあげられる．

参 考 文 献

1) 池谷友基，津邑公暁，松尾啓志，中島康彦：複数イタレーションの一括再利用による並列事前実行の高速化，情報処理学会論文誌 コンピューティングシステム（ACS），Vol.3, No.3, pp.31–43 (2010).
2) 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).
3) Norvig, P.: *Paradigms of Artificial Intelligence Programming*, Morgan Kaufmann (1992).
4) Sodani, A. and Sohi, G.S.: Dynamic Instruction Reuse, *Proc. 24th International Symposium on Computer Architecture*, pp.194–205 (1997).
5) González, A., Tubella, J. and Molina, C.: Trace-Level Reuse, *Proc. International Conference on Parallel Processing*, pp.30–37 (1999).
6) Huang, J. and Lilja, D.J.: Exploiting Basic Block Value Locality with Block Reuse, *Proc. 5th International Symposium on High-Performance Computer Architecture*, pp.106–114 (1999).
7) HAL Computer Systems/Fujitsu: *SPARC64-III User's Guide* (1998).
8) ルネサステクノロジ：ニュースリリース: R8A20410BG (2009).