情報処理学会論文誌　コンピューティングシステム　Vol. 3　No. 3　31–43 (Sep. 2010)

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

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み合わせた並列事前実行を提案している．従来の並列事前実行機構ではループの各イタレーションを再利用対象の命令区間として抽出していた．本稿では，実行バイナリに変更を加えることなく，複数イタレーションを動的にまとめて再利用対象区間とすることによって，再利用に要するオーバヘッドを削減し，同時に再利用表エントリの効率的な活用を実現する手法を提案する．また，いくつのイタレーションを再利用区間として統合すべきかは対象ループにより異なるため，動的に適切な数を検出するモデルを提案する．SPEC CPU95 FP 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 15.0%であったサイクル数削減率が，最大 57.6%，平均 26.0%まで向上することを確認した．

A Speed-up Technique for Parallel Early Computation by Collectively Reusing Multiple Iterations

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. In the past model, the parallel early computation detects each iterations of loops as reusable blocks. This paper proposes a new parallel early computation model, which integrates plural iterations into a reusable block automatically and dynamically without modifing executable binaries. We also proposes a model for automatically detecting how many iterations should be integrated into one reusable block. Our model reduces the overhead of computation reuse, and further exploits reuse tables. The result of the experiment with SPEC CPU95 FP suite benchmarks shows that proposing method improve the maximum speedup from 40.5% to 57.6%, and the average speedup from 15.0% to 26.0%.

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ともない，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 やスーパスカラ等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ともなってコア数も増大していくと考えられている．さて，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)]．

並列化が，処理全体の総量自体は変化させず複数の単位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計算再利用可能な命令区間と見なし，実行時にその入出力を記憶しておくことで，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，その実行自体を省略する．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でも効果が得られる可能性があり，並列化とも併用可能であるという利点がある．

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あらかじ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，従来の自動メモ化プロセッサがループの各イタレーションを計算再利用対象命

†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

令区間として扱っていたのに対し，ループ展開を施した場合と同様に複数イタレーションを単一の命令区間として扱い，一括して再利用する手法を提案する．これにより，既存バイナリをいっさい変更することなく，再利用テストオーバヘッドの削減および再利用表の有効活用が期待できる．また，各ループにより適切な展開数は異なるため，これらを動的に決定する仕組みについても提案する．

2. 関 連 研 究

値予測に基づく投機的実行により，命令レベルの並列度を確保する研究[3),4)]，またそれを発展させ，複数の予測値に基づいて，複数のプロセッサコアを投入して高速化を図る投機マルチスレッド（SpMT: Speculative Multi-Threading）の研究が数多く行われている．投機スレッドは，その参照アドレスが通常実行スレッドにより書き換えられた場合，通常は squash される．

投機的実行の結果を一部再利用する研究も行われている．Roth ら[5)] は，過去のレジスタマッピングを書き戻すことで，以前の失敗した投機実行により書き込まれた物理レジスタの中身を再利用する方法を提案している．

また我々の手法と同様，計算再利用と投機を組み合わせた方法も研究されている．Wu ら[6)] は，コンパイラが再利用区間の切り出しを行い，再利用不可能である場合には再利用区間の出力値を予測して，後続区間の実行を投機的に開始する手法を提案している．この手法では，出力値の予測が外れた場合，後続区間の投機的実行をキャンセルする必要があり，このための機構のコストおよびオーバヘッドが問題となる．Molina ら[7)] は，投機実行スレッドと通常実行スレッドを組み合わせる手法を提案している．この手法では，投機実行スレッドによる実行済の命令は FIFO に格納され，通常実行スレッドはそこから命令を取り出してソースオペランドを比較し，一致した場合その結果を用いる．

これに対し我々の提案している並列事前実行は，投機スレッドが通常スレッドの実行と並行して，予測された入力値を用いて事前に実行自体を済ませ，その結果を再利用表に格納する．これにより，通常スレッド実行時に再利用が効果的に行えるようになり，高速化が図れる．この方式の特長は，文献 6) とは異なりコンパイラ支援を必要としない点，および再利用区間の入力値を予測の対象としているため，失敗した投機実行をキャンセルする必要がない点があげられる．また，考え方は文献 7) に比較的近いが，入力の比較の際にキャッシュの内容まで比較することで，主記憶参照を必要とする命令区間に対しても再利用が適用できるという利点がある．

3. 自動メモ化プロセッサと並列事前実行

本章では，まず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．

3.1 自動メモ化プロセッサ

計算再利用（Computation Reuse）とは，主に関数等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）**[8)] と呼ぶ．

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（**図 1**）．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 Memo-

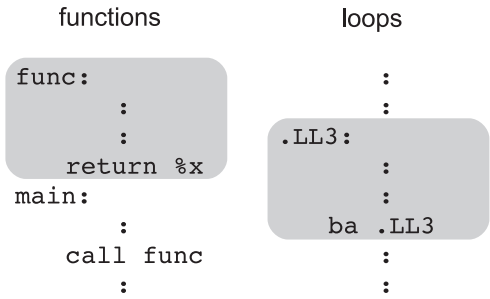

図 1　メモ化可能命令区間
Fig. 1　Memoizable instruction regions.

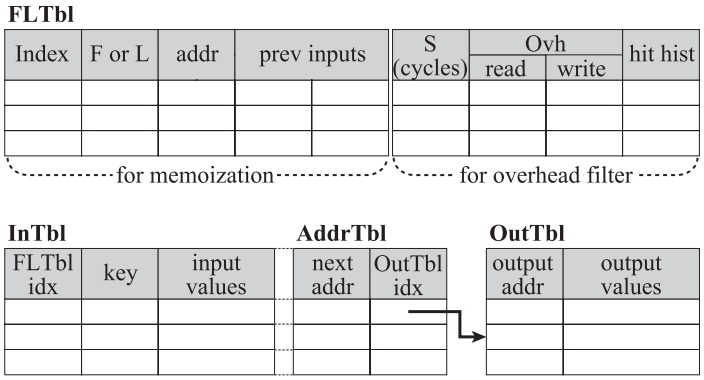

図 2　MemoTbl の構造
Fig. 2　Organization of MemoTbl.

Tbl 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すべて含まれる．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き込みが含まれる．

MemoTbl は，命令区間の開始アドレスを記憶する **FLTbl**，入力値を記憶する **InTbl**，入力アドレスを記憶する **AddrTbl**，そして出力値を記憶する **OutTbl** の 4 つの表から構成されている（**図 2**）．

FLTbl 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）を記憶するフィールドがある．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出力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．

InTbl は，命令区間の入力値を記憶する表である．各行は FLTbl の行番号 Index に対応する FLTbl idx を持ち，どの命令区間の入力値を記憶しているかがこの値により判別される．一般に命令区間内では，複数の入力が順に参照され使用されるが，同じ命令区間でもある入力の値が異なると，その次入力アドレスが変化する場合がある．これは，主記憶アドレス値自体が入力値として用いられる場合や，条件分岐の存在が原因である．つまりある命令区間の入力アドレスの列はその入力値によって分岐してゆくため，全入力パターンはツリー構造で表現できる．InTbl はこのツリー構造を折り畳んで格納する必要があるため，各エントリは入力値を記憶する input values フィールドに加えて，当該エントリの親エントリを指す key フィールドを持つ．

AddrTbl は，入力アドレスのセットを記憶する表であり，上で述べた次入力アドレスを保持している．AddrTbl は InTbl と同数のエントリを持ち，同じ Index を持つ InTbl エントリの次入力アドレスを，next addr フィールドで記憶している．

OutTbl は命令区間の出力を記憶する表であり，命令区間の出力列のアドレスおよび値を，output addr/output values フィールドで保持している．また，すべての入力の一致を確認した際に適切な OutTbl エントリを参照できるよう，入力列の末尾を構成する AddrTbl エントリは，対応する OutTbl エントリのエントリ番号を OutTbl idx フィールドに保持している．

紙面の都合上，詳細は文献 1)，2) にゆずるが，MemoTbl 検索手順の概要は以下のとおりである．命令区間を検出すると，その命令区間の開始アドレス，および関数/ループの別の 2 つの情報を用いて FLTbl を検索し，その Index を得る．次に，その Index を FLTbl idx フィールドに持ち，当該命令区間のレジスタ入力を input values フィールドに格納しているエントリを InTbl から検索する．このエントリが，これから検索しようとする入力ツリーのルートとなる．

ここでマッチしたエントリと同じ行番号を持つ AddrTbl を参照すると，次に参照すべき入力アドレスが格納されているため，この値をキャッシュから読み出すことで次入力値を得る．この入力値を input values フィールドに持ち，かつ先にマッチした親エントリの番号を key フィールドに持つエントリを，再び InTbl から検索する．これをすべての入力の一致が確認されるまで繰り返し行う．

すべ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 InTbl エントリと同じインデクスを持つ AddrTbl エントリが，対応する出力を格納している OutTbl エントリへのポインタを OutTbl idx フィールドに保持しているため，これを用いて OutTbl を参照し，得られた出力のアドレス/値の組をすべてレジスタおよびキャッシュに書き戻すことで

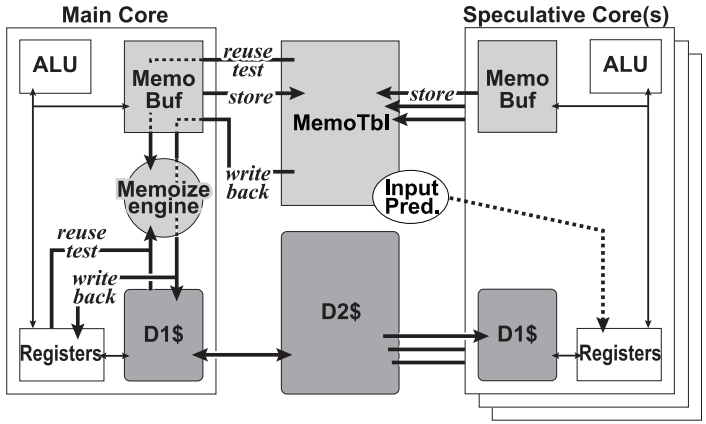

図 3　並列事前実行機構
Fig. 3　Execution mechanism of a parallel early computation.

命令区間の実行が省略される．なお，InTbl 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．

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，その際には再利用表を検索するコスト，および入力が一致したエントリに対応する出力値を再利用表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ト率が低い場合には，性能が悪化してしまう場合もある．

3.2　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，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 1 つとして扱うループイタレーションでは，まった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 3** 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すでに SpC により実行済みであり 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，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．

3.3　オーバヘッドフィルタ機構

3.1 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，メモ化適用によりかえって実行サイクル数が増加する場合もある．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 FLTbl 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 2 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．Ovh^R，Ovh^W はそれぞれ，過去の履歴より概算した，当該命令区間の再利用表検索オーバヘッド，および再利用表からキャッシュ等への書き戻しオーバヘッドである．なお，S，Ovh^R，Ovh^W は，図 2 に示した FLTbl の S フィールド，および Ovh read/write フィールドからそれぞれ取得される．

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，

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

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．式 (1) から式 (2) を引いたものを $\boldsymbol{Gain}$ とすると，

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

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

4. 複数イタレーションの一括再利用

本章では，本稿で提案するループイタレーションの再利用モデルについて説明する．

4.1 概　　要

前章で述べたように，入力が単調に変化するループイタレーションに対しては，並列事前実行機構が有効に働く．しかし，いくつかのループに対してその効果は限定的となる．特に効果が限定されるのは，

(a) ループボディの計算量が軽微である場合

(b) イタレーション回数が多い場合

の 2 つの場合である．

(a) ループボディの計算量が少ない場合，計算再利用成功時に省略できる計算量のほとんどが再利用オーバヘッドによって相殺され，速度向上はわずかとなる．よって計算再利用の効果を大きくするためには，再利用により省略される命令区間の計算量に対する，再利用オーバヘッドの比率を低減させる必要がある．

また，(b) イタレーション回数が多いループの場合，SpC による並列事前実行もそれに応じて多数回行われるが，これらの実行結果が登録されることで MemoTbl エントリを多数消費する．MemoTbl 容量は有限であるため，多数のエントリを消費することで空きエントリが枯渇し，将来計算再利用に利用できるエントリまで追い出されてしまう可能性が高くなってしまう．このことにより，場合によっては他の関数等の命令区間の再利用率を低下させることもある．よって，並列事前実行の結果をできるだけ少量のエントリで記憶できることが望ましい．

そこで本稿では，複数回のイタレーションをまとめて 1 つの再利用可能命令区間と見なすことで，これら 2 つの問題を解決する手法を提案する．以下，SPEC95 CPU から**図 4** に示した 103.su2cor のプログラムを例に，複数イタレーションの一括再利用がなぜ解決策となりうるのかについて述べる．

図 4 中の DO ループにおいて，プログラムカウンタを除き，ループボディに対応する再利用可能命令区間の入力は，イタレータ I および参照されている変数 NDIM，LSIZE(I)，

```
      :
      NPTS=1
      LVEC=1
C
      DO 10 I=1,NDIM
      IF(MOD(LSIZE(I),2) .NE. 0) STOP
      NVOL=NPTS
      NPTS=NPTS*LSIZE(I)
      LHALF(I)=LSIZE(I)/2
      LVEC=LVEC*LHALF(I)
10    CONTINUE
      :
```

図 4　103.su2cor のプログラムコード（部分）
Fig. 4　A part of 103.su2cor program code.

MOD(LSIZE(I),2)，NPTS，LVEC である．一方出力は，イタレータ I および NVOL，NPTS，LHALF(I)，LVEC である．よって従来の並列事前実行機構では，1 イタレーションごとに 1 つのレジスタ入力（イタレータ I）と 5 つの主記憶入力の入力一致比較が行われ，また再利用成功時にはイタレータ I および 4 変数の出力書き戻しが行われる．

ここで，上記 DO ループの 2 イタレーション分を 1 つの再利用可能命令区間として扱うことを考えよう．これは，2 イタレーション分を 1 イタレーションとするようなループ展開を適用した場合と同様の効果がある．展開後の新たなループでは，ループボディに対応する再利用可能命令区間の入力は，イタレータ I および変数 NDIM，NPTS，LSIZE(I)，MOD(LSIZE(I),2)，LSIZE(I+1)，MOD(LSIZE(I+1),2)，LVEC であり，展開前と比較して 2 つ増加するのみである．一方出力は，I，NVOL，NPTS，LHALF(I)，LHALF(I+1)，LVEC の 6 つとなり，わずかに 1 つ増加するのみである．

すなわちこの例の場合，イタレーション回数 NDIM に対し，ループ全体の実行を通じて元のコードでは NDIM × 6 個の入力に対する一致比較が必要であるのに対し，ループ展開後のコードでは NDIM ÷ 2 × 8 個の入力に対する一致比較を行えばよく，再利用テストオーバヘッドが約 2/3 に減少することとなる．再利用成功時の書き戻しオーバヘッドに関しても同様に削減される．これにより，本節冒頭で示した，並列事前実行の効果が限定されてしまう場合 (a) に対する有効な解決方法となる．

一方，並列事前実行機構が想定している単純なストライド予測が成功するようなループの

場合，その入力値は一定ストライドで変化しているということになる．このようなループに対しループ展開を施した場合，そのストライドは定数倍に変化することになるが，一定ストライドで変化するという特徴が失われることはないため，予測ヒット率にはほとんど影響を与えないと考えられる．

このようにループ展開は，対象ループに対して計算再利用を適用する際に入出力と見なされる変数の数を減少させる．一方で，これはすなわち MemoTbl に登録する必要のある入出力の数が減少することも意味しており，再利用オーバヘッドを低減させるだけではなく，MemoTbl エントリの使用量を削減する効果もある．図 4 の例で 2 イタレーション分をまとめて扱う場合，総入力数が 2/3 に減少することより，InTbl および AddrTbl の使用エントリ数も概算で 2/3 に減少すると考えられる．また，OutTbl の使用エントリ数も出力数に応じて減少する．よって，効果が限定されてしまう場合 (b) に対しても有効な解決方法となる．

なお，図 4 は入出力数ともに大きく削減できる例であり，すべてのループにおいてこれと同様に入出力が減少するわけではない．しかし，最悪の場合でも入力の 1 つであるイタレータ変数の一致比較は削減されるため，どのようなプログラムに対しても少なくとも 1 つ以上の入力値を削減可能である．

4.2　イタレーションの展開数

以上に述べた動作を並列事前実行機構に行わせるためには，簡単なカウンタを設け，イタレーション実行回数をカウントしながら計算再利用を適用していけばよい．その際に考慮すべき点は，何回のイタレーションを 1 つの再利用可能区間と見なすかである．1 つの再利用区間と見なすイタレーション回数（以下，**展開数**と呼ぶ）を $\boldsymbol{n}$ とする．図 4 の例では，総イタレーション回数 $\boldsymbol{N}$ にわたるループ全体の実行に対して一致比較が必要な入力数は $(N \div n)(4+2n) = N(2+4/n)$ となり，展開数 n が大きくなるにつれ再利用オーバヘッドは低減する．しかし，n を増大させることによる弊害も 2 つ存在する．

まず 1 つめの弊害は，ストライド予測を行うためのデータが収集されるまでに必要となるイタレーション実行回数が増加することである．たとえば，0 を初期値としストライド 1 で単調増加するようなイタレータ変数 i により制御される，あるループの実行を考える．SpC 数が 3 の場合の各コアの実行タイミングを**図 5** に示す．並列事前実行機構は $i = 0$ において後方分岐を発見し，命令区間を検出する．次に $i = 1$，2 を通常実行した後，メインコアが $i = 3$ を実行している間，$i = 1$ および $i = 2$ で使用した入力値セットからストライドを計算し，そのストライドから算出した予測値に基づいて SpC が $i = 4$ の実行を行う．よって予測が正しかった場合，メインコアは $i = 4$ 以降の実行を省略することができる（図 5 中央）[*1]．

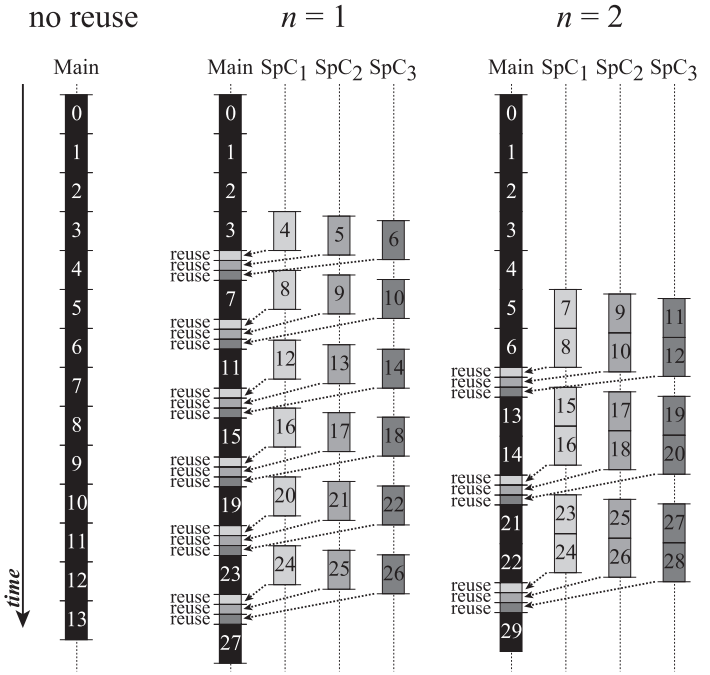

図 5　各コアの実行タイミング（SpC×3 の場合）
Fig. 5　Computation Timing with three SpCs.

一方，複数イタレーションをまとめる場合，並列事前実行機構は $i = 0$ において命令区間を検出した後，$i = 1, 2, \cdots, n$，続けて $i = n+1, \cdots, 2n$ を実行し，そこで初めてストライド予測が可能となる．よってメインコアが $i = 2n+1, \cdots, 3n$ を実行している間，SpC が $i = 3n+1, \cdots, 4n$ を投機実行することになり，メインコアが実行を省略できるのは最善の場合でも $i = 3n+1$ 以降の部分となる（図 5 右，$n = 2$ の場合）．従来再利用可能であった $i = 4, \cdots, 3n$ に対しては通常どおり実行する必要がある．すなわち，$i = 3n+1, \cdots, N$ に

[*1] ストライド計算のオーバヘッド等により，厳密には $i = 4$ よりさらに少し後になる．詳細については文献 2) を参照されたい．

対する再利用オーバヘッドの削減が，$i = 4, \cdots, 3n$ に対する実行コストよりも大きくなければ性能が悪化してしまう．

2 つめの弊害は，命令区間の実行時間が長くなることで，SpC による事前実行がメインコアの再利用に間に合わなくなる場面が増えることである．n イタレーションをまとめて扱うことで，命令区間の実行時間は約 n 倍に増加するが，一般に当該区間の再利用テストオーバヘッド増加が n 倍未満に抑えられるのは図 4 の例で見てきたとおりである．一方 SpC による事前実行は，キャッシュミス等の原因により実行に遅延が生じる場合がある．よって再利用が成功するメインコアは，従来モデルよりも早く SpC による事前実行に追いついてしまう可能性が高くなり，事前実行が終了する前に当該イタレーションの実行が開始されることで再利用の失敗率が増大することが予想される．

4.3　展開数の動的決定手法

前節で述べたように，展開数 n に関しては大きい値を設定すればよいわけではなく，適切な値を探る必要がある．また，命令区間の計算量や入力数にも影響を受けるため，命令区間（すなわちループ）ごとに異なる値を設定するのが望ましいと考えられる．

一般に，適切な展開数は対象となるループ命令区間によって異なると考えられる．しかし，プログラム解析等によって適切な展開数を事前に算出することは困難であるため，これを動的に決定するアルゴリズムが必要である．

展開数を増加させることによる影響は，4.2 節で述べたように，1 イタレーションあたりの入出力数の減少による再利用オーバヘッドの低減，および再利用表の利用効率化による再利用ヒット率の向上というメリットと，再利用可能開始時刻の遅延による再利用ヒット率の低下，および SpC の計算時間の増加による再利用ヒット率の低下というデメリットがあげられるため，これらを総合して結果的に性能が向上するような展開数を発見する必要がある．

一方，すでに 3.3 節で述べたように，FLTbl の各エントリにはそれぞれの命令区間で削減できる見込みサイクル数，再利用に要するオーバヘッド，および最近の再利用ヒット履歴が格納されている．そこで，これらの値を用いて計算再利用による効果を計測し，展開数を変化させたときにその効果がどう変化するかによって，適切な展開数を決定することとする．

具体的な手順は以下のとおりである．まず前節で述べたように，各ループに対応する FLTbl 内のエントリに展開数を保持させる．展開数は，後述するようにハードウェアコストの観点から 2^k で表される値とし，この初期値は 1（$k = 0$）とする．これにより，プロセッサの動作開始時には既存モデル同様，すべてのループに対して 1 イタレーション分が再利用対象命令区間として扱われる．

さて，ループイタレーションが実行され，また再利用が適用されると，当該イタレーションに対応する，再利用による削減サイクル数 S，再利用テストオーバヘッド Ovh^R，出力書き戻しオーバヘッド Ovh^W，および再利用ヒット履歴が FLTbl 内に記憶される．この際，3.3 節で述べたように，オーバヘッドフィルタ機構により，式 (3) で表される $Gain$ が計算されるが，この値を FLTbl 内に記憶しておく．これを $UnitGain_0$ とする．

$\boldsymbol{UnitGain_k}$ は，2^k イタレーション分を再利用対象区間として並列事前実行を行った際に得られる利得サイクル数 $Gain_k$ を，1 イタレーションあたりの利得として正規化した値を表す．すなわち

$$UnitGain_k = Gain_k / 2^k \tag{4}$$

である．展開数は 2^k で表される値としているため，$UnitGain_k$ の算出は複雑な機構を必要とせず，オーバヘッドフィルタによって算出される $Gain_k$ を k bit 右シフトすることで簡単に得られるため，追加ハードウェアもごくわずかで済む．なお，$UnitGain_0 = Gain_0$ である．

次に並列事前実行機構は，展開数を増大させようと試みる．当該ループに対応する k の値は 1 となり，次回登録時から前節で述べた手順により 2（$= 2^1$）イタレーション分が一括して処理される．3.3 節で示した一定回数 T の再利用試行の後，式 (3) および式 (4) により $UnitGain_1$ が得られる．ここで $UnitGain_1$ が $UnitGain_0$ より大きい場合，2 イタレーションを一括して扱うことで計算再利用が効率化されたと考える．

このようにして順次 k をインクリメントしながら，$k = x$ において $UnitGain_{x-1}$ と $UnitGain_x$ を比較する．$UnitGain_{x-1} < UnitGain_x$ が成立する間はインクリメントを続け，$UnitGain_{x-1} > UnitGain_x$ が成立した段階で，$x-1$ を当該ループに対応する k の値として採用し，以降その値で k を固定する．

このアルゴリズムでは必ずしも k の最適解が得られるとは限らないが，シフト演算と値比較という非常に小さなコストで算出できるという利点がある．本章で見てきたように，展開数を増大させることにはメリットとデメリットが存在する．そしてそれらのいずれの効果が大きいかが，結果的に性能を向上させるかどうかに影響する．本アルゴリズムは，$UnitGain$ が増加している間はメリットの影響が増大しており，また $UnitGain$ が減少に転じた時点でデメリットの影響が支配的になりはじめたと推測することで，近似的に k の最適解を求めている．

もちろん，$UnitGain_{x-1} > UnitGain_x$ が成立し，$k = x$ において性能低下に転じた場合でも，その後 $UnitGain$ が単調減少するとは限らず，再び向上する可能性もある．しかし $UnitGain$ の値が減少に転じた時点で，そこまでの探索空間における最良の値は直前の $UnitGain$ であるため，真に最適解である保証はないが現実的に探索可能な範囲における比較的適切な値として，これを採用することとした．

5. 動作モデル

4.2 節で述べたように，何回のイタレーションを 1 つの再利用可能区間と見なすべきかは，そのループの計算量や入力数等の特徴によって左右される．よって，命令区間を管理している表 FLTbl の各エントリにフィールドを追加し，当該ループが 1 つの区間と見なすべき展開数を格納することとする．以下，再利用表への登録時，および再利用の検索時に分けて動作を説明する．

5.1 登録動作

イタレーション回数をカウントするために，新たに MemoBuf にカウンタを付加する．このカウンタはプログラム開始時に 0 に初期化される．

プロセッサはイタレーションの実行を検出すると，以降そのイタレーション終端の後方分岐命令への到達回数をこのカウンタによりカウントする．そして，到達回数が当該命令区間に対応する展開数の値と一致するまで，MemoBuf への入出力登録を続行する．

当該ループの後方分岐命令への到達回数が展開数の値と一致すると，再利用対象区間の実行が終了したと判断し，その時点で MemoBuf 上に格納されている複数イタレーション分の入出力を，一括して 1 つのエントリとして MemoTbl に登録する．また，これと同時にカウンタをリセットする．このようにすることで，実行バイナリにはいっさい変更を加えることなく，複数イタレーションを単一の再利用対象命令区間として扱うことができる．

SpC は FLTbl を参照し，ストライド予測により得られた入力値セットを用いて命令区間を実行するのは既存モデル[2] と同様であるが，唯一異なるのは，カウンタがリセットされたときにのみ事前実行を行うという点である．

なお，イタレーションカウンタの値が指定された展開数に到達する前に，当該ループ末端の後方分岐命令が untaken となる，すなわち当該ループの実行が終了する場合がある．これは，ループの全回数が展開数で割り切れず，端数が出たことを表している．この場合，展開数に達していなくても再利用対象命令区間の実行が終了したと判断し，その時点で MemoBuf に格納されている入出力を MemoTbl に登録する．

5.2 検索動作

検索時に関しては，特に留意すべき点は存在しない．メインコアは，現在再利用が可能か否かをテストしようとしているループにおいて展開数がどのような値に設定されているか，すなわち，当該ループがいくつのイタレーション分を 1 単位として再利用表に登録されているかを，そもそも知る必要がない．再利用表に登録されている入力値すべてに対して一致比較を行えば，登録されているイタレーション分の入力比較が十分であることは保証される．

なお，入力すべてが一致し再利用が成功した場合には出力の書き戻しが行われるが，この際も同様に展開数を知る必要はない．再利用表に登録されている出力値セットにはイタレータ変数自体も含まれているはずであり，それは現イタレータ変数の値から，展開数ぶんだけ先の値を示しているはずである．よってその値が書き戻されることで，適切な回数のイタレーション処理がスキップされる．

6. 評　　価

6.1 評価環境

以上で述べた拡張を並列事前実行シミュレータに実装し，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[9] を参考とした．また，MemoTbl の InTbl を構成する大容量 3 値 CAM は，MOSAID 社の DC182888[10] の構成を参考にし，サイズは 32 Byte 幅 × 4 k 行 の 128 kByte とした．また，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．

6.2 速度評価

SPEC CPU95 FP (train) の 10 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評価を行った．結果を**表 2** および**図 6** に示す．なお，gcc-3.0.2 の -O2 オプションでは，ループアンローリングや関数のインライン展開は適用されない．

図 6 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，test(r)，test(m) はそれぞれレジスタ/キャッシュと InTbl(CAM) との一致比較オーバヘッド，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．

評価は，再利用を行わないモデル，従来の並列事前実行モデル，提案モデルについて行っ

表 1　シミュレータ諸元
Table 1　Simulation parameters.

MemoBuf	64 kBytes
MemoTbl CAM	128 kBytes
Comparison (register and CAM)	9 cycles/32 Bytes
Comparison (Cache and CAM)	10 cycles/32 Bytes
Write back (MemoTbl to Reg./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

表 2　削減サイクル数率（SPEC CPU95 FP）
Table 2　Reduced execution cycles (SPEC CPU95 FP).

	Mean	Max
(P) 既存モデル	**15.0%**	**40.5%** (107.mgrid)
(P_2) 参考：展開数 2 固定	19.3%	41.5% (101.tomcatv)
(P_4) 参考：展開数 4 固定	15.9%	42.7% (102.swim)
(D) 提案モデル	**26.0%**	**57.6%** (101.tomcatv)

た．また参考データとして，展開数を動的に変更せず，全ループに対し一定値 2（$k = 1$）および 4（$k = 2$）として固定したものについても測定を行った．なお，すべ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 6 中で各ベンチマークプログラムの結果を 5 本のグラフで示しているが，それぞれ左から順に

(M)　メモ化を行わないモデル
(P)　並列事前実行の既存モデル
(P_2)　展開数を 2 に固定した参考モデル
(P_4)　展開数を 4 に固定した参考モデル
(D)　展開数をループごとに動的に決定する提案モデル

が要したサイクル数を表している．なお，各サイクル数は (M) を 1 とする正規化を行っている．なお，提案モデル (D) において *UnitGain* を計算するための再利用試行回数 T は，3.3 節で述べた既存モデルのオーバヘッドフィルタ機構で用いているシフトレジスタが 64 bit を仮定しており，提案モデルも同機構を利用することを仮定していることから，64 とした．

まず，(P)（展開数 = 1），(P_2)，(P_4) の結果より，展開数による性能の変化が見てとれ，いずれの展開数が最適であるかがベンチマークプログラムごとに異なっていることが分かる．たとえば 103.su2cor，141.apsi では (P_4) が最も良い結果となっているが，110.applu では (P_2) が最も良く，107.mgrid では (P) が最も良い結果となっている．特に 107.mgrid の (P_4) による性能低下は激しく，プログラムによっては，すべてのループに対し無条件に展開数を増加させることが大きな性能低下を引き起こしうることを示している．

この結果から分かるように，各プログラムには適切な展開数が存在するが，それはプログラムごとに異なっている．また，各プログラム内にも一般に複数のループが存在し，それら個々のループごとにもそれぞれ異なった適切な展開数が存在する．しかし，それらの適切な展開数はプログラムの入力等によっても変化する場合があるため，実行前に知ることは非常に困難である．よって，あらかじめ静的にループアンローリングを施しておく方法で性能向上を得ることは難しいと考えられる．

これに対し提案手法である (D) は，非常に良い結果を示している．まず 102.swim，107.mgrid，145.fppp では，(P)，(P_2)，(P_4) のうち最も良いものとほぼ同等の性能となっている．また，101.tomcatv，104.hydro2d，125.turb3d，146.wave5 では，全モデル中最も良い結果となっており，ループごとに展開数を設定することができる効果が得られていると考えられる．

内訳を見ると，101.tomcatv，102.swim，103.su2cor，146.wave5 等で，(D) は既存モデル (P) に比べ exec を大きく削減できており，再利用表の利用効率向上によるヒット率の増大が起こっていると考えられる．また，102.swim では exec を増大させることなく再利用テストのサイクル数が削減されており，一括再利用によるオーバヘッド削減が効果的に実現できていることが分かる．

提案モデルでは各展開数における *UnitGain* を測定する必要がある理由から，やむをえず一定の期間不適切な展開数で実行されることになる．107.mgrid および 110.applu ではこの影響を受け，従来モデルよりそれぞれ 0.3%，および 1.2%性能が低下してしまっている．しかし全体として性能は向上しており，従来モデルの削減サイクル数が平均 15.0%，最大 40.5%であったのに対し，提案モデルでは平均 26.0%，最大 57.6%と，大きく改善された．

ここで，提案手法により性能が低下する場合に関し考察する．一部のプログラムでわずか

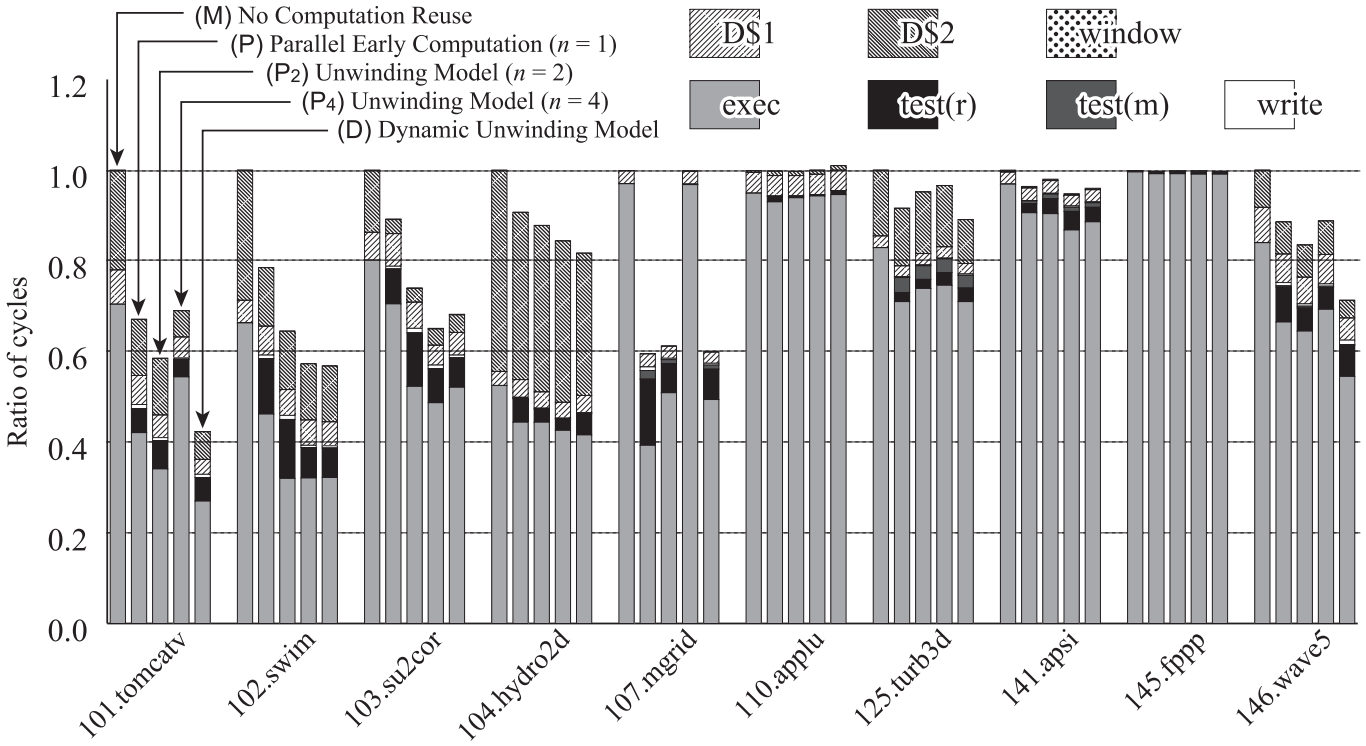

図 6　実行サイクル数比（SPEC CPU95 FP）
Fig. 6　Ratio of cycles (SPEC CPU95 FP).

ながら性能低下を引き起こしている原因は，$UnitGain$ が悪化に転じる際に一定期間（具体的には $T = 64$ 回）不適切な展開数で実行されることにある．以下あるループにおいて，この $UnitGain$ が悪化に転じる際の展開数を 2^x（$k = x$）とする．一方，既存モデルは展開数 $2^0 = 1$ で動作しており，1 回あたり $UnitGain_0$ の速度向上を得る．よって既存モデルに対し速度低下が起こる可能性があるのは，

$$UnitGain_0 > UnitGain_x \tag{5}$$

が成立し，かつ T 回の試行による速度低下

$$(UnitGain_0 - UnitGain_x) \times 2^x \times T \tag{6}$$

に比べて，最終的に得られる速度向上が小さい場合ということになる．

すなわち，提案モデルにより性能が悪化するループは，式 (5) を満たし，かつ式 (6) の値が大きくなるようなものであり，以下のような特徴を持つ．

(A)　$UnitGain_0 - UnitGain_x$ の値が大きい．

(B)　x の値が大きい．

しかし，(B) $x \gg 0$ である場合，それまで $UnitGain$ が単調増加してきていることから式 (5) が成立する可能性は低く，式 (5) が成立するような x は 1，もしくはそれに近い小さな値であると考えられる．図 2 の結果を見ても，性能が悪化している 107.mgrid と 110.applu はともに $k = 0 \rightarrow 1$ すなわち (P)→(P$_2$) で悪化に転じており，$x = 1$ である．一方で，4.2 節であげた，$UnitGain$ を悪化させる可能性のある 2 つの弊害の影響は，展

開数が増大するに従い大きくなる．よって上述のとおり x の値が小さいのであれば，(A) $UnitGain_0 - UnitGain_x$ の値はさほど大きくはならない．以上の議論より，ある特定のループの性能が低下するような場合でも，その性能低下は大きくはならないと考えられる．さらに，あるループが性能低下を引き起こす場合でも，他のすべてのループが同様に性能低下を引き起こす性質を持っているような特殊な場合でない限り，他のループにおける性能向上によってその性能低下は相殺されると考えられる．

次にハードウェアコストの観点から議論する．本提案手法は，単一コアの通常プロセッサと比較すると，約 4 倍のハードウェア量に加えて，再利用表の追加が必要となる．たとえば共有メモリ型マルチプロセッサでは，並列化コンパイラを用いて最適化を行った場合，4 台構成の環境において 101.tomcatv，102.swim，104.hydro2d，107.mgrid の 4 プログラムに対し，3.8 倍の速度向上，すなわち 72%のサイクル数削減が得られることが報告されている[11]．一方で本提案手法では，上記 4 プログラムでは平均 42%のサイクル数削減となり，これに劣る結果となる．しかし本提案手法は，(1) 並列化が困難であるプログラムに対しても有効である，(2) 既存のロードモジュールに対しバイナリ互換性を保ったまま適用可能である，(3) 並列化手法と共存可能である，等のさまざまな利点があり，これらを考慮すると有効な結果であるといえる．

6.3　再利用率の変化

前節で示した結果のとおり，ほとんどのプログラムで高速化を確認できた．また，多くのプログラムでは exec サイクル数を削減できており，期待したとおり MemoTbl の有効活用により再利用率が向上していると考えられる．これを確認するため，詳細な評価を行った．

まず，再利用表が有効活用されていることを確認するため，提案モデル (D) において，既存モデル (P) に対して exec サイクルが減少した 4 つのプログラム 101.tomcatv，102.swim，103.su2cor，146.wave5 について，MemoTbl で発生したパージの回数を計測した．この結果を**表 3** に示す．

MemoTbl では空きエントリが不足してしまった際に，LRU に基づき不要と判断したエントリをパージすることで，新たな登録のために空きエントリを確保している．すなわち，このパージ回数は MemoTbl が溢れた頻度を表している．

表 3 の結果より，exec が減少したすべてのプログラムにおいてパージ回数が減少していることが分かる．特に 103.su2cor では回数が半分以下にまで減少しており，同じプログラムでもループイタレーションの登録方法を変更することで再利用表エントリを有効に活用できることが分かる．

表 3　再利用表パージ回数の変化
Table 3　Number of purges.

	既存モデル (P)	提案モデル (D)	削減率
101.tomcatv	7515980	6945025	7.6%
102.swim	814208	455615	44.0%
103.su2cor	20712071	10169396	50.9%
146.wave5	1798492	1661194	7.6%

表 4　再利用率（101.tomcatv）
Table 4　Reuse rate (101.tomcatv).

			既存モデル (P)		提案モデル (D)	
	addr.	cycles	search	rate	search	rate
(func)	1f358	964	66048	99.9%	66048	99.9%
	19458	279	999	99.6%	999	99.6%
	22fa4	105	499	74.6%	499	74.2%
	1f908	1371	501	99.2%	501	99.2%
(loop)	1abb0	437	32384999	74.2%	16192468	72.8%
	1b074	70	126999	28.9%	8133	55.6%
	1b130	274	32257999	—	8128017	65.5%
	1b318	186	32257999	—	4064077	64.1%
	1b420	112	32384999	—	2040133	56.2%

次に，パージ回数の減少が実際に再利用率の向上に寄与したことを確認するため，上記 4 プログラムの中でも総実行サイクル数が大幅に削減できている 101.tomcatv と 102.swim に対し，各命令区間の再利用率の変化を調査した．この結果を**表 4** と**表 5** に示す．なお，命令区間の総数は膨大であるため，ここでは全体性能に与える影響が大きいと考えられる，実行サイクル数もしくは実行回数が多いものを代表的な命令区間として示した．

命令区間は関数およびループに分けて示しており，各行が 1 つの命令区間に対応している．各命令区間について，その命令区間の開始アドレス（addr.），実行サイクル数（cycles），既存モデルと提案モデルにおける，その命令区間の InTbl 検索回数（search），および再利用成功率（rate）を示した．(P) において 1 度も再利用が行われなかった命令区間の rate は空欄としてある．なお，自動メモ化プロセッサでは命令区間の入れ子構造に対応しており[2]，外側の命令区間の再利用が成功した場合内側の命令区間ごとスキップされるため，再利用状況の変化により search の値が増減する場合がある．また，(D) でループの search 値が大きく減少しているが，これはイタレーション統合により再利用可能命令区間としてのループの

表 5　再利用率（102.swim）
Table 5　Reuse rate (102.swim).

			既存モデル (P)		提案モデル (D)	
	addr.	cycles	search	rate	search	rate
(func)	11274	404	161070	17.6%	263660	25.7%
	114dc	409	161070	18.4%	263660	27.6%
	109f4	41	294006	1.8%	434699	0.0%
	10af0	24	264215	1.9%	386916	0.0%
(loop)	1b33c	310	2616319	33.7%	655379	73.1%
	1b6ec	330	2616319	33.4%	655379	71.2%
	1b020	118	262631	—	16519	65.1%
	1b200	68	262655	—	8467	51.4%
	103dc	14	32738	—	1284	68.2%
	1bcb0	256	4087	—	386	29.5%
	1bd84	256	4087	—	391	20.7%

検出回数が減少するためである．

表 4 から分かるように，101.tomcatv では代表的な関数の再利用率にはほとんど変化が見られなかった．一方ループでは，1abb0 等多少再利用率が低下してしまっているものも存在するが，1b074 の再利用率が大きく改善されているのに加え，従来再利用ができていなかった複数のループに対して再利用が可能となっており，全体として再利用の成功率が向上していることが分かる．

また表 5 の 102.swim では，11274，114dc 等，一部の関数の再利用率も向上していることが分かる．ループに関しては，表にあげた代表的なものでは再利用率が向上しており，また新たに再利用可能となったループも多く存在することが分かる．

以上の結果から，本提案手法が再利用表の効率的な利用，ひいては再利用率の向上に寄与していることが確認できた．

7. 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並列事前実行機構に対し，複数のループイタレーションを単一の再利用対象命令区間として扱うことで，既存バイナリにはいっさい変更を加えることなく，複数イタレーションに対し一括して計算再利用を適用する手法を提案した．また，各ループに最適な展開数を動的に決定する方法について述べた．

SPEC CPU95 FP ベンチマークを用いて評価した結果，提案手法が再利用オーバヘッドの削減および再利用率の向上に大きく寄与することを確認した．既存モデルでは平均 15.0%，最大 40.5%であったサイクル数削減率が，提案するモデルでは平均 26.0%，最大 57.6%まで向上させることができた．

今後の課題としては，まず SPEC CPU95 FP 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，本提案モデルの適用範囲や，他命令区間の再利用率への影響等について詳しく調査することがあげられる．また，試行回数 T 等の，今回評価に用いた各パラメータを変化させることによる影響も調査する必要がある．次に，本稿のモデルは再利用オーバヘッドの削減を 1 つの目的としているが，他のオーバヘッド削減モデル[1)] との融合も検討していきたい．また，本稿では比較的単純なアーキテクチャを想定して評価を行ったが，より複雑な環境を想定したシミュレーションを行うとともに，命令レベル並列性に基づく高速化手法とメモ化とを組み合わせた手法を探ってゆくことも今後の課題である．

謝辞　本研究の一部は，（財）栢森情報科学振興財団研究助成金による．

参 考 文 献

1) Kamiya, Y., Tsumura, T., Matsuo, H. and Nakashima, Y.: A Speculative Technique for Auto-Memoization Processor with Multithreading, *Proc. 10th Int'l. Conf. on Parallel and Distributed Computing, Applications and Technologies* (*PDCAT'09*), pp.160–166 (2009).
2) 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：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，先進的計算基盤システムシンポジウム SACSIS2004 論文集，pp.251–259, 情報処理学会 (2004).
3) Lipasti, M.H. and Shen, J.P.: Exceeding the Dataflow Limit via Value Prediction, *29th MICRO*, pp.226–237 (1996).
4) Wang, K. and Franklin, M.: Highly Accurate Data Value Prediction Using Hybrid Predictors, *30th MICRO*, pp.281–290 (1997).
5) Roth, A. and Sohi, G.S.: Register Integration: A Simple and Efficient Implementation of Squash Reuse, *33rd MICRO* (2000).
6) Wu, Y., Chen, D. and Fang, J.: Better Exploration of Region-Level Value Locality with Integrated Computation Reuse and Value Prediction, *28th ISCA*, pp.98–108 (2001).
7) Molina, C., González, A. and Tubella, J.: Trace-Level Speculative Multithreaded Architecture, *ICCD* (2002).
8) Norvig, P.: *Paradigms of Artificial Intelligence Programming*, Morgan Kaufmann (1992).
9) HAL Computer Systems/Fujitsu: *SPARC64-III User's Guide* (1998).
10) MOSAID Technologies Inc.: *Feature Sheet: MOSAID Class-IC DC18288*, 1.3 edi-

tion (2003).
11) 笠原博徳，小幡元樹，石坂一久：共有メモリマルチプロセッサシステム上での粗粒度タスク並列処理，情報処理学会論文誌，Vol.42, No.4 (2002).

(平成 22 年 1 月 26 日受付)
(平成 22 年 5 月 8 日採録)

池谷　友基（学生会員）

1987 年生．2010 年名古屋工業大学工学部情報工学科卒業．現在，同大学大学院工学研究科創成シミュレーション工学専攻修士課程在籍．計算機アーキテクチャ等に興味を持つ．

津邑　公暁（正会員）

1973 年生．1998 年京都大学大学院工学研究科情報工学専攻修士課程修了．2001 年同大学院情報学研究科博士後期課程学修認定退学．同年同大学院経済学研究科助手．2004 年豊橋技術科学大学工学部助手．2006 年名古屋工業大学大学院工学研究科助教授．2007 年同准教授．博士（情報学）．プロセッサアーキテクチャ，並列処理応用，脳型情報処理等に関する研究に従事．電子情報通信学会，日本神経回路学会，ACM，IEEE-CS 各会員．

松尾　啓志（正会員）

1960 年生．1985 年名古屋工業大学大学院修士課程修了．1989 年同大学院博士後期課程修了．同年同大学電気情報工学科助手．1993 年同講師．1995 年同助教授．2003 年同教授．2004 年同大学情報工学科教授．工学博士．計算機工学，分散協調システムに関する研究に従事．IEEE，人工知能学会，電子情報通信学会各会員．

中島　康彦（正会員）

1963 年生．1986 年京都大学工学部情報工学科卒業．1988 年同大学大学院修士課程修了．同年富士通（株）入社．スーパコンピュータ，汎用コンピュータ，命令エミュレーション，高速 CMOS 回路設計等に関する研究開発に従事．1999 年京都大学総合情報メディアセンター助手．同年同大学院経済学研究科助教授．2002 年より（兼）科学技術振興機構さきがけ研究 21（情報基盤と利用環境）．2006 年奈良先端科学技術大学院大学情報科学研究科教授．コンピューティング・アーキテクチャ講座担当．IEEE-CS，ACM 各会員．