自動メモ化プロセッサにおける複数イタレーションの一括再利用

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 Auto-Memoization Processor by Collectively Reusing Plural 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．

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．一般にプログラムとして記述される対象処理自体は半順序構造を成しており，その構造は時間軸に沿った縦方向への広がりと，時間軸に直交する横方向への広がり（並列性）を持っている．処理のこの横方向への広がりに着目し，その処理量を圧縮しようとするのがこれらの高速化手法であると言える．

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)] 計算再利用は上記の処理の半順序構造における縦方向の広がりを圧縮しようと試みる手法であり，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

†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

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令区間として扱っていたのに対し，ちょうどループ展開を施したかのよう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，既存バイナリを一切変更することなく，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合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おり，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，再利用技術を利用し，値予測に基づいた非対称な SpMT モデルである．この方式の特長は，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挙げられる．また，考え方は 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．

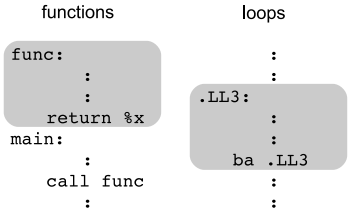

図 1 メモ化可能命令区間

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）**[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 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 2**）．

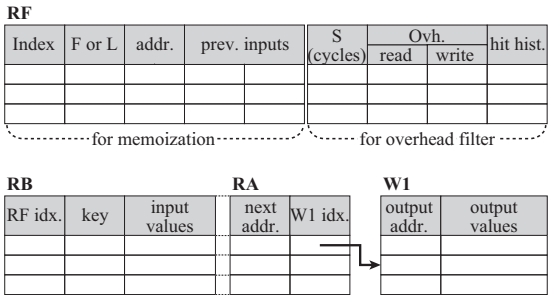

図 2 MemoTbl の構造

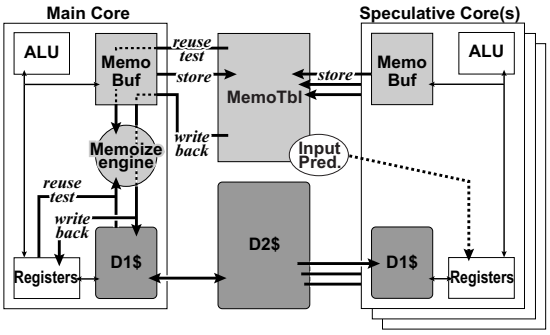

図 3 並列事前実行機構

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，命令区間開始アドレス，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，再利用に要するオーバヘッド，過去の再利用ヒット履歴が保持される．

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，再利用が必要と判断された場合，現在のレジスタ上の入力値を 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．

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，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 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．

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 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 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

$$M \cdot (S - Ovh^R - Ovh^W) \quad (1)$$

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，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．

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，

$$(T - M) \cdot Ovh^R \quad (2)$$

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，削減できたサイクル数 (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再利用表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，ループボディの計算量が軽微なものや，イタレーション回数が多いものに対してである．

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，速度向上は僅かとなる．また，イタレーション回数が多い場合，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，この実行結果が登録されることで MemoTbl 容量を圧迫し，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さえある．

そこで本稿では，複数回のイタレーションをまとめて一つの再利用可能命令区間とみなすことで，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得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)，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 イタレーション毎に 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．

```
       :
      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）

ここで，上記 DO ループの 2 イタレーション分を一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$\times 6$ 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$\div 2 \times 8$ 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．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，ループ展開は予測ヒット率に殆ど影響を与えないと考えられるため，計算再利用による高速化率を維持しつつ再利用オーバヘッドを削減できる．同時に，MemoTbl に登録される入出力セットの数も半減するため，当該ループによる MemoTbl のエントリ使用量も最大約半分に削減でき，再利用表を従来よりも有効に活用可能となる．これによって，他の関数等の命令区間における再利用ヒット率の向上も望める．

なお，図 4 は入出力数ともに大きく削減できる例であり，全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ひとつであるイタレータ変数の一致比較は削減され

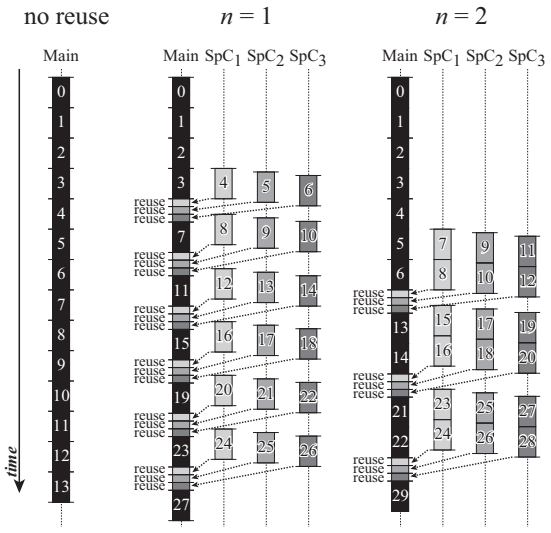

図 5 各コアの実行タイミング（SpC×3 の場合）

るため，どのようなプログラムに対してもある程度のオーバヘッド削減が見込める．

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一つの再利用可能区間とみなすかである．一つの再利用区間とみ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，全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いくつか存在する．

まず一つめの弊害は，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．例えば，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]

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．

二つめの弊害は，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 の例で見てきた通りである．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．

以上のように，n に関しては大きい値を設定すればよいわけではなく，ある程度最適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．

5. 実　　装

前章で述べた動作を実現するための，並列事前実行機構の実装モデルについて説明する．

5.1 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一つの再利用可能区間とみなすべきかは，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 RF の各エントリにフィールドを追加し，当該ループが一つの区間とみ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．

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．

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

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，一括してひと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，実行バイナリには一切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 RF を参照し，ストライド予測で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時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．

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全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，入力全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．

5.2 展開数の動的決定手法

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挙げられるため，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，既に 3.3 節で述べたように，RF 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 RF 内のエントリに，展開数を保持させる．展開数は 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，全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，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 RF 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 RF 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

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

であり，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僅かで済む．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，$k = n$ において $UnitGain_{n-1}$ と $UnitGain_n$ を比較する．$UnitGain_{n-1} < UnitGain_n$ が成立する間はインクリメントを続け，$UnitGain_{n-1} > UnitGain_n$ が成立した段階で，$n-1$ 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．

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 RB 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，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 評価結果

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．

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 RB(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，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 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) 展開数をループ毎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．また，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 64bit 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．例えば 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，全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得ることを示している．

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，ループ毎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見てとれる．また，102.swim では exec サイクル数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，止むを得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．

表 1 シミュレータ諸元

MemoBuf	64 kBytes
MemoTbl CAM	128 kBytes
Comparison (register and CAM)	9 cycles/32Bytes
Comparison (Cache and CAM)	10 cycles/32Bytes
Write back (MemoTbl to Reg./Cache)	1 cycle/32Bytes
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)

	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)

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

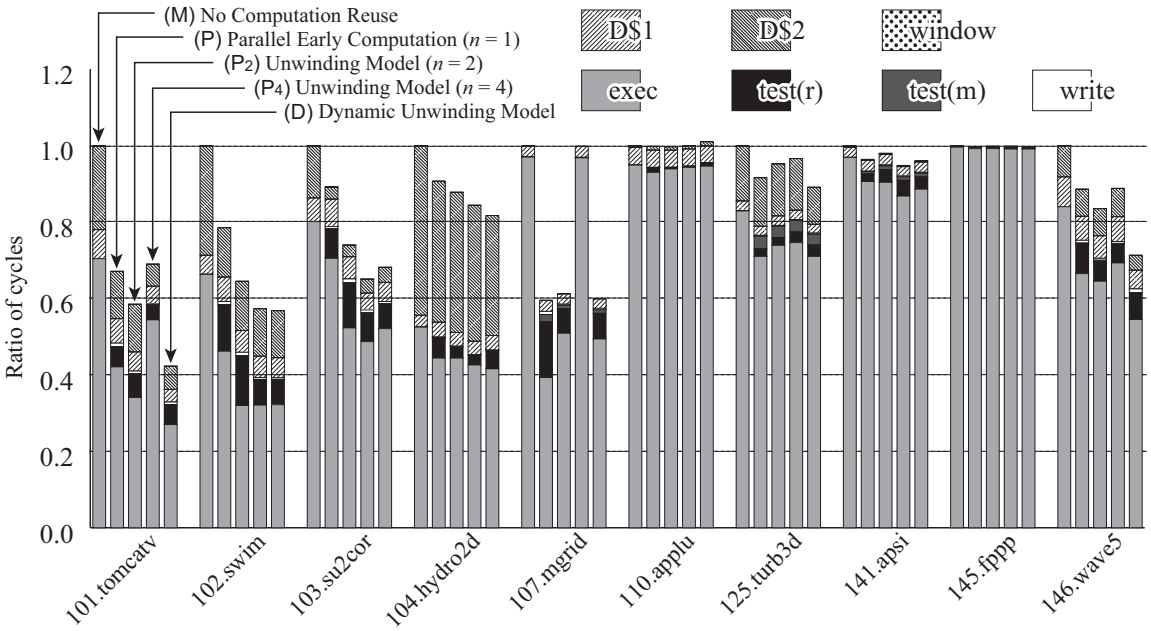

図 6 実行サイクル数比（SPEC CPU95 FP）

とで，既存バイナリには一切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 以外の様々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挙げられる．次に，本稿のモデルは再利用オーバヘッドの削減をひとつの目的としているため，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，情報処理学会，pp.251–259 (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 edition (2003).