並列事前実行機構における主記憶値テストの高速化

津邑公暁[†] 中島康彦[†,††] 五島正裕[†]
森眞一郎[†] 富田眞治[†]

区間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する．投機実行スレッドが参照する主記憶位置が通常実行スレッドにより書き換えられた場合，書き換えられたアドレスを記録しておくことで，通常実行による再利用時に最低限必要な主記憶入力値のみを比較することが可能となり，オーバヘッドが削減できることを示す．SPEC95 を用いた評価では，最大 55%，平均でも SPECint で約 20%，SPECfp で約 35%のサイクル数を削減することができ，いずれも，投機実行の無効化などの既存手法よりも良好な結果となった．

Fast Reuse Test of the Memory Values on Parallel Early Computation

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

This paper proposes an asymmetrical speculative multithreading with region reuse and parallel early computation. Comparing only the location which may be overridden by non-speculative store, we can reduce the overhead of reuse test of the memory values. It corresponds that the non-speculative store marks the address in reuse buffer that refers the same address. We show the maximum ratio of eliminated cycle reaches to 55%, and the average one reaches to 20% against SPECint95 benchmark programs and 35% against SPECfp95. These results are better than using existing methods such as invalidation.

1. はじめに

既存のロードモジュールを高速化する手法として，スーパスケーラがある．しかし命令レベル並列性を追究するだけでは性能向上が見込めなくなってきており，現在では**投機的マルチスレッディング**（Speculative Multi-threading：以下，**SpMT**）の研究が広く行われている．

さて SpMT において，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．ただしその実現のためには，従来研究が行われているような，投機実行スレッドと通常実行スレッドの 1 対 1 の引き継ぎデータ構造では不十分であり，多くの実行結果の中から 1 つを選ぶ，多対 1 の引き継ぎデータ構造が必要となる．我々はこの 1 モデルとして，再利用および事前実行による非対称な SpMT を提案している[1)]．

再利用とは，関数呼び出しやループなどの命令区間において，入力と出力の組を表に記憶しておき，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，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数が増えても再利用機構の複雑さは増大しない点にある．副次的な効果として，冗長な load/store 命令や消費電力を削減できることも報告されている[2)]．再利用手法は，ハードウェアによるものやソフトウェアによるもの，またその両方を利用したものなど，様々なものが提案されている．

Sodani ら[3)] は，最大 1024 エントリからなるフルアソシアティブの再利用表を用い，単命令を対象とした汎用的な再利用を提案している．再利用表の各エントリは，命令オペランドの値および命令結果を保持する．また，load 命令が再利用可能であることを保証するために，主記憶有効ビットおよび主記憶アドレスが保持される．store 時には，主記憶アドレスが連想検索さ

† 京都大学
Kyoto University
†† 科学技術振興機構さきがけ研究 21
PRESTO, JST

れ，無効化される．他の方法として，再利用表にオペランドレジスタの識別子を保存する方法も提案されている．González ら[4] は，最大 256 K エントリからなる Reuse Trace Memory（RTM）と呼ばれる表を用いて評価を行っている．RTM は PC の一部によるインデクシングを仮定しており，比較的小さい深さの CAM で実現可能となっているが，CAM の総容量は 32MB 以上にも及んでいる．Costa ら[5] は，load/store 命令を対象から除外したうえで，フルアソシアティブの表による再利用手法を提案している．PC およびオペランドの値は，この表により連想検索される．

ソフトウェア支援を追加したものとしては，Huang ら[6] が，コンパイラに機能追加することで，再利用区間内に閉じたレジスタによる出力の登録を省く，基本ブロックの再利用方式を提案している．キャッシュの内容まで比較しているため，ポインタにも対応している．Connors ら[7] は，コンパイラによる区間切り出しとハードウェアサポートを組み合わせた手法を提案している．最近の load から次の load までの間に store 命令が存在しない場合は，主記憶比較を省略している．Connors ら[8] はまた，コンパイル時情報に加え実行時情報も用いることで，再利用表の割当てを最適化する手法を提案している．Huang ら[9] は，一般に基本ブロックの入出力数は様々であるため有限幅の再利用表を効率的に使うことができないとし，基本ブロックをサブブロックに分割することにより，再利用表に登録可能な入出力数に揃える方法を提案している．

これらのほかにも再利用に関しては，入力の一致判定に寛容性を持たせる曖昧再利用[10] や，load 命令に限りアドレスおよび load 値を再利用する手法[11),12] など，様々な研究が行われている．

一方，**投機実行**（Speculative Execution）においては，値予測に基づく投機的実行を行う場合，予測の正否をつねに検証する必要があるため，先行命令の実行時間そのものを削減することができない．このため，厳密な検証が必要となる値そのものを投機対象とするのではなく，投機的マルチスレッド実行機構を利用して 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（以下，**予備実行**：Precomputation）が多く行われている[13)~15)]．また CMP や SMT において，簡略化したプログラムを事前に実行し，そこから将来に関する理想的な情報を受け取ることで通常実行を高速化するという，slipstream と呼ばれる方式も研究されている[16),17)]．

これに対して我々の提案している**事前実行**（Early Compuataion）は，予測される入力値に基づいて実行自体を事前に行い，結果を再利用表に登録する手法である．これにより，同一入力が出現する間隔が長い場合や，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．従来の予備実行とは本質的に異なる手法である．

さて，再利用を行う際，その命令区間の入力，すなわちその命令区間が参照した主記憶アドレスにおいて，以前実行した時点の格納値が書き換えられている可能性がある．このため，当該主記憶アドレスに格納されている値をつねにすべて読み出し，一致比較を行う必要がある．よって再利用可能か否かを判定するコストが大きくなり，再利用による高速化の効果が上がらないという問題がある．

一方，SpMT において，主記憶一貫性を保つ一般的な方法は，投機実行の無効化である．投機実行スレッドの開始後に，通常実行スレッドが投機実行スレッドのソースオペランドを上書きした時点でその投機実行の結果は利用できないと見なし，投機実行は無効化される．しかし，将来そのアドレスが再び以前の値に書き換えられる可能性もあり，その場合，以前の投機実行の結果を保存しておけばそれを利用することが可能である．

そこで本稿では，主記憶の書換えがあった場合にその再利用表のエントリを無効化するのではなく，その書換えがあったアドレスを記憶しておき，再利用時にそのアドレスのみを比較することによって，再利用率を低下させることなく主記憶入力値比較コストを軽減する手法を提案する．本稿で提案する方法をシミュレータ上に実装し，SpMT において一般的に用いられる無効化による手法，および従来の引数すべてを比較する手法とあわせて評価・比較を行った．

以下 2 章では，関連研究についてまとめる．次に 3 章で事前実行の詳細について説明し，その課題について述べる．4 章では本稿が提案する再利用時コストを低減させる具体的な手法について述べ，5 章ではハードウェアモデルの詳細とハードウェアコストについて述べる．最後に 6 章でその評価結果を示し，7 章でまとめを行う．

2. 関 連 研 究

本章では，本研究と関連する SpMT，および投機と再利用の組合せによる研究の動向についてまとめ，本研究の位置付けを明らかにする．

SpMT：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[18),19)]，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 SpMT 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．

Gopal ら[20] は，階層的マルチプロセッサシステムにおいて，各主記憶値の予測値を複数保持する Speculative Versioning Cache を提案している．各プロセッサは固有のキャッシュを持ち，各キャッシュラインには順序づけされたキャッシュセットが保持される．キャッシュはキャッシュセットから無効化応答を受け取ると，squash 信号をプロセッサに送信する．Marcuello ら[21),22] も，投機的マルチスレッドプロセッサのための，トレースに基づく増分予測を提案している．増分は，当該トレースの前回実行時における，トレース終了時の値からトレース開始時の値を減じた差分として計算される．Oplinger ら[23] は，call された関数本体およびその関数の復帰後に続くコードを，並列実行する方法を提案している．文献 20)〜23) とは対照的に Codrescu ら[24] は，ループイタレーションや関数呼び出しではなく load 命令を契機にスレッドを起動することで，細粒度の投機的スレッドを生成可能な自動分割ポリシーを提案している．投機的データキャッシュは通常実行のプロセッサが発行した store により生じる擾乱を検出するために保持される．また Marcuello ら[25] は，コントロールグラフおよび到達可能性に基づく，プロファイルベースの投機実行スレッド起動ポリシーを提案している．

投機＋再利用：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[26] 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，SPEC ベンチマークプログラムの実行時間を最大 11% 削減している．また，その再利用手法とデータ駆動スレッドを統合した，投機的データ駆動マルチスレッディングも提案しており[27]，SPEC ベンチマークプログラムの実行時間を最大 25% 削減している．事前に実行された結果は，レジスタリネーミングを利用して物理レジスタに格納される．

また我々と同様，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[28] 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．これにより，4 命令および 8 命令同時発行の構成で，SPEC ベンチマークプログラムの実行時間において 1.25 倍から 1.4 倍の速度向上を得ている．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[29] 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みの命令は 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．4 命令同時発行の構成で，SPEC ベンチマークプログラムの実行時間において最大 1.33 倍，平均で 1.16 倍の速度向上を得ている．

これに対し我々は，再利用技術を利用した，並列事前実行による非対称な SpMT を提案している．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．また，命令区間の複雑な入れ子構造についても多重的に再利用が適用できる機構を提案している．この方式の特長は，文献 28) 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 29) 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. 並列事前実行

本章では，我々の提案している，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，その課題を明らかにする．

3.1 機構と動作

まず並列事前実行機構の概要を**図 1** に示す．Main Stream Processor（以下，**MSP**）は，通常実行を行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，キャッシュは各プロセッサごとに独立しており，再利用表および主記憶は全プロセッサで共有される．再利用表への登録パスを，図中では破線で示した．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 はつねに通常実行のみを行い，SSP はつね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）

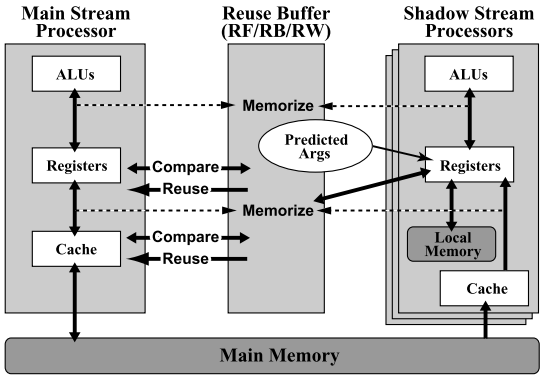

図 1 並列事前実行機構
Fig. 1 Structure of parallel early computation.

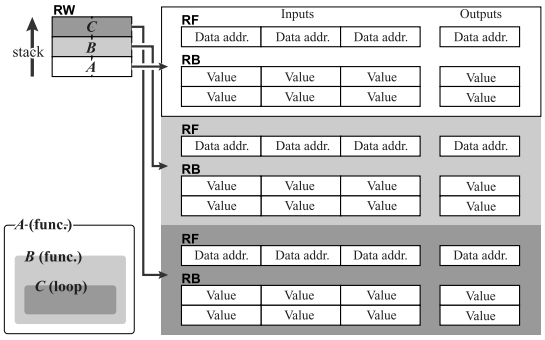

図 2 再利用表の概要
Fig. 2 Structure of reuse buffers.

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，命令区間はすでに SSP により実行済みであり，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．

さて，本機構では，図 1 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，**関数管理表（RF）**，**入出力記録表（RB）**，**再利用ウィンドウ（RW）**からなる．再利用表の概略を**図 2** に示す．RF は再利用対象となる命令区間の先頭アドレスおよび読み出し/書き込みで参照されるアドレスを保持し，1 エントリが 1 命令区間に対応する．RB は，命令区間の実行が開始されたときのスタックポインタの値や，命令区間の入出力を記録するための表である．命令区間の実行時には，参照されたレジスタおよび主記憶のアドレスを RF に，また参照された値を命令区間の入力として RB にそれぞれ登録する．書き込みが発生したレジスタおよび主記憶に関しては同様に出力として登録する．

図 2 左下に示すように，関数 A が関数 B を呼び出し，関数 B がループ C を実行するような多重構造を形成する命令区間に対し，最外の命令区間 A を一度実行しただけで，その内部に含まれるすべての命令区間 B および C が再利用可能となるように登録を行うために，RW を用いて，RF および RB を命令区間の入れ子構造と関連づける．RW は，現在実行中かつ登録中の RF および RB の各エントリをスタック構造として保持する．命令区間の実行中は，RW に登録されているすべてのエントリについて，レジスタおよび主記憶参照を登録する．

3.2 課　　題

以上に述べた並列事前実行機構における課題は，(1) いかにして命令区間を選択するか；(2) いかにして入力を予測し，プロセッサに割り当てるか；(3) いかにして主記憶一貫性を保つか；にある．以下に，順に詳述する．

命令区間の選択

どの命令区間を投機実行の対象として選択するかが重要である．同一パラメータが出現する間隔が長い命令区間や，パラメータが単調変化する命令区間に対して効果があることが予想されるものの，各命令区間の性質や事前実行の効果は，実行してみなければ分からない．このため，RF に新規登録された命令区間については，ただちに SSP により数回分の事前実行を試みる．試行の結果，MSP による登録頻度が高く，かつ SSP が登録したエントリの再利用頻度も高い RF を，継続して SSP の実行対象とすることで，事前実行に向くと思われる命令区間を選択することができる．

また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，RF ごとに小さなハードウェアを付加し，各エントリの有効性を $E =$ (過去の省略ステップ数) $\times$ (登録回数) $\times$ (再利用回数) として計算する．一定期間における登録回数および再利用回数は，RF ごとに付加したシフトレジスタによって記憶しておく．各 SSP は，E が最大となる RF を選択し，1 セットの予測引数セットを受け取り，その区間の投機実行を開始する．これにより，つねに削減ステップ数や再利用頻度の高い命令区間を選択することができる．

入 力 予 測

投機実行のためには，RB の使用履歴に基づいて将来の入力を予測し，SSP へ渡す必要がある．このために，RF の各エントリごとに小さなハードウェアを設け，MSP や SSP とは独立に入力予測値を求める．具体的には，最後に出現した入力 a および最近出現した 2 組の入力の差分 d に基づいてストライド予測[19]を行う．$a + d$ に基づく命令区間の実行は MSP がすでに開始していると考えると，n 台の SSP が存在す

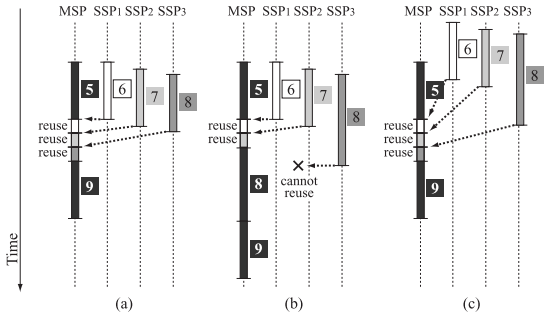

図 3 投機実行および再利用のタイミング
Fig. 3 Timing requirement for reuse and early computation.

る場合，SSP_i に対し入力予測値 $a + d \times (i + 1)$ を用意する．$a + d \times (n + 2)$ は MSP に割り当てる．

例として，SSP 3 台，ストライド 1 の場合を考え，MSP が入力値 ‘5’ に対する処理を実行中だとする（**図 3**）．最も効率良く再利用が行われるためには，MSP が入力値 ‘5’ に対する処理を終え入力値 ‘6’，‘7’，‘8’ の実行に移ろうとしたとき，3 台の SSP でそれぞれの入力に対する処理が終わっていなくてはならない（図 3 (a)）．各入力に対する処理は，通常ほぼ同じ時間で終了すると考えられるが，キャッシュミスの発生などにより遅延することがあり，SSP における処理の開始時刻が MSP に近い場合，MSP が SSP と同じパラメータによる実行を開始してしまう（図 3 (b)）．このため SSP における ‘6’，‘7’，‘8’ に対する処理は，MSP の ‘5’ に対する処理よりも早めに開始しておく必要がある（図 3 (c)）．つまり，現在 MSP で実行中の入力よりもある程度先まで，SSP への入力割当てを行っておかなくてはならない．

SSP 台数 n に対し，我々は $2(n + 1)$ エントリからなる引数の割当表をシフトレジスタを用いて構成することで，実行中の入力より $2(n + 1)$ ストライド先の入力までを予測し，MSP および各 SSP に割り当てている．SSP 3 台の例を**図 4** に示す．ある入力に対する処理に要する時間を通常平均で x，1 度の再利用に要する時間を $r(\ll x)$ とする．この長さのシフトレジスタを用意することで，各 MSP，SSP にはそれぞれ 2 つ先までの入力が割り当てることができる．これによって SSP の処理時間に余裕を持たせることができ，ある程度並列事前実行が進み安定状態に入った後，たとえば図 4 において入力 ‘22’ に対する SSP_1 の処理は，$2x + 5r \simeq 2x$ すなわち通常要する 2 倍程度の時間以内に終了すれば MSP による再利用に間に合うことになる．

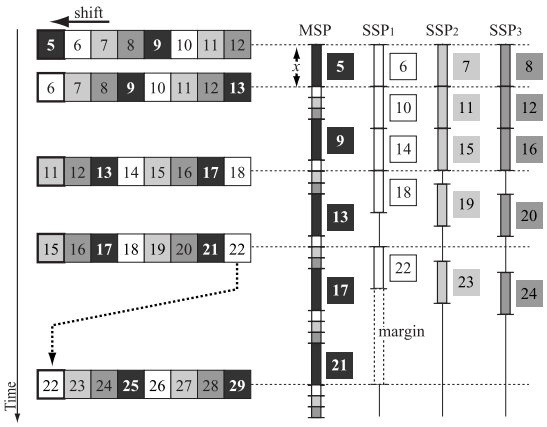

図 4 投機実行の入力割当てのためのシフトレジスタ
Fig. 4 Shift register for early computation.

主記憶一貫性制御

各 SSP は予測された入力に基づいて命令を実行するため，SSP により生成される主記憶値は，MSP により将来的に書き込まれる正しい値とは異なる．このため図 1 に示したように，キャッシュおよび主記憶に値を書き戻すことができるのは MSP のみとしている．各 SSP は局所変数以外を記録するために，キャッシュや主記憶の代わりに専用の RB エントリを用いる．また，スタックエリア内の局所変数を記録するために，局所メモリを使用する．MSP が主記憶に対して書き込みを行った場合は，対応する SSP のキャッシュラインは無効化される．RB への登録対象のうち，読み出しが先行するアドレスについては主記憶を参照し，MSP 同様アドレスおよび値を RB へ登録する．以後，主記憶ではなく RB を参照することで，他のプロセッサからの上書きによる矛盾の発生を避けることができる．一方，局所メモリに対する，書き込みに先行する読み出しアクセスは，変数を初期化せずに使うことに相当し，値は不定でかまわないため，主記憶を参照する必要はない．関数フレームの大きさが局所メモリ容量を超えた場合は，SSP による実行を打ち切る．

さて以下では，主記憶一貫性を保証しながら，いかに再利用時の主記憶値テストを高速化するかという点に焦点を当てる．

4. 主記憶値テストの高速化手法

本章では，主記憶一貫性保証のための具体的手法について述べる．特に MSP が主記憶値を書き換えた場合の再利用表の扱いに関して考えられる手法を説明し，高速化のための手法を提案する．

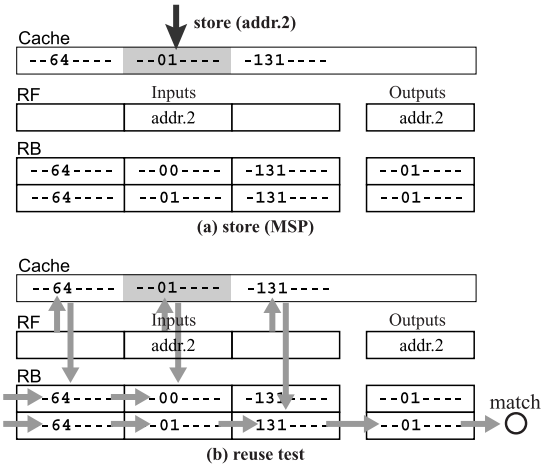

図 5 全比較
Fig. 5 Comparing all inputs.

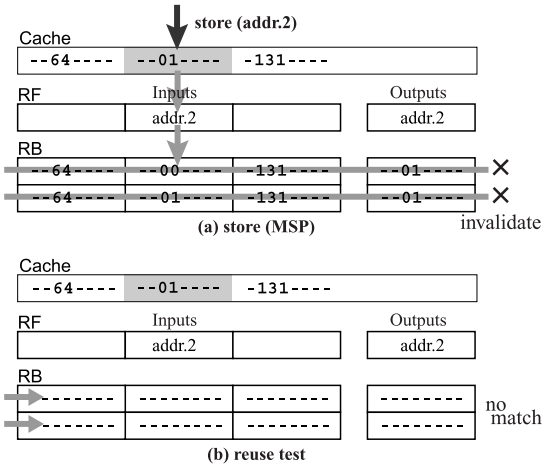

図 6 無効化
Fig. 6 Invalidation.

4.1 全 比 較

主記憶一貫性を保つうえで，これまで我々は，再利用時に，RB に記録されている主記憶値をすべて比較検査する方法（以下，**全比較**）を採用し，再利用機構の評価を行ってきた[1)]．この方法を**図 5** に示す．

Store 発生時：MSP による store が発生した場合，主記憶の内容が書き換えられるが，RB の各エントリに対しては特に何も操作は行わない（図 5 (a)）．また，SSP による store は主記憶自体を書き換えないため，これもやはり特に操作は行わない．

再利用テスト：一方，再利用テスト時においては，入力となるすべての主記憶値について参照，比較を行う（図 5 (b)）．これにより，MSP による store によって登録時と再利用時の主記憶値に不整合が起こっている場合でも，不正な再利用を抑止することができる．しかしすべての入力に関して主記憶値を比較するため，再利用テストのオーバヘッドが大きくなり，再利用で得られる性能向上を一部相殺してしまうという問題がある．

4.2 無 効 化

これに対し，SpMT で最も一般的に用いられている方法は**無効化**（invalidation）である．事前実行機構に適用した場合を**図 6** に示す．

Store 発生時：RF に登録されている主記憶アドレス addr.2 に対し MSP による store が発生した場合，アドレス addr.2 を入力として参照するような RB エントリをすべて検索し，これを無効化する（図 6 (a)）．SSP による store の場合は，主記憶自体は書き換えないため，特に RB に対する操作は必要ない．

再利用テスト：内容が変更された主記憶アドレスを入力として参照していた RB エントリは無効化されるため，RB 内で有効であるエントリにおいては，主記憶値が変更されていないことが保証される．これにより，再利用時の主記憶値比較が不要となり，再利用テストのオーバヘッドは大幅に削減される．しかし有効な RB エントリが減少するため，再利用率が低下してしまい，全比較を用いた場合のような高速化は望めないと考えられる（図 6 (b)）．

4.3 一 部 比 較

そこで本稿では，主記憶において書き換えられた可能性のある箇所のみを比較する方法（以下，**一部比較**）を提案する．各アドレスに対し比較の必要があるか否かを記憶しておき，再利用テスト時に，そのアドレスのみに関して主記憶値の読み出しおよび比較を行う方法である．

本機構を実現するために，図 2 に示した RF に対し，各入力アドレスに 1bit のフラグを付加する拡張を行う．そして，RF に登録したアドレスから読み出した主記憶値と，RB 内に保持されている主記憶値を比較する必要があるか否かを，本フラグを用いて表す．この様子を**図 7** に示す．新しい命令区間の実行にともない，再利用表への登録を開始する時点で，このフラグはリセットされる．以下では，このフラグをどのような場合にセットするかについて説明したうえで，再利用テストの手順を示す．

MSP による store 発生時：再利用表登録後に MSP や I/O が同一アドレスに store を行った場合，store は実際に主記憶値を更新するため，当該アドレスのフラグを立てる必要がある．

RB への登録時における主記憶書き込みの際には，

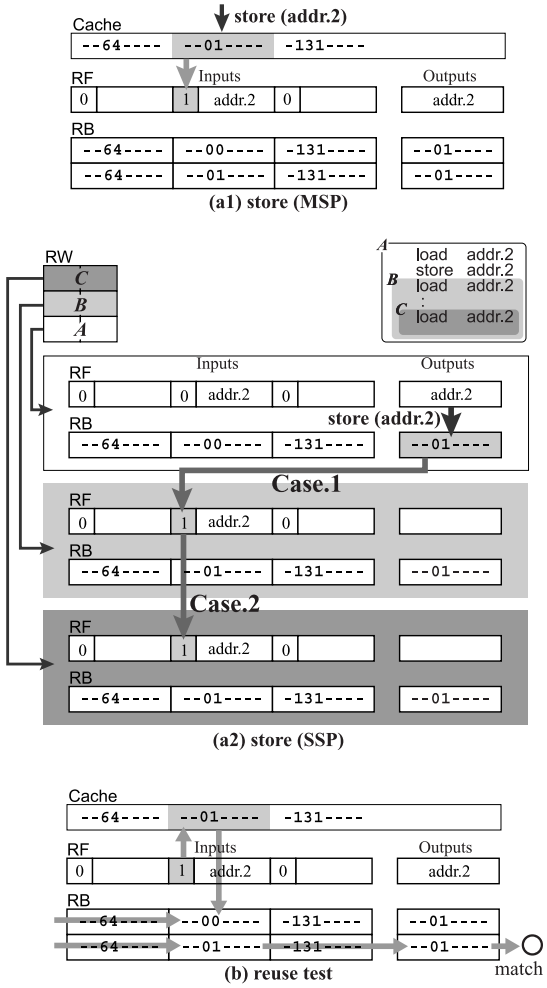

図 7 一部比較
Fig. 7 Comparing inputs partially.

まず全登録表において，登録されている主記憶入力アドレスが，これから登録しようとする主記憶書き込みアドレス addr.2 と一致するか否かを検査する．一致した場合，前述のフラグをセットする（図 7 (a1)）．なお主記憶読み出しの際には，過去に当該主記憶に書き込んだことがある場合はその値を，書き込みがなく読み出したことがある場合はその値を使用する．いずれもない場合は実際に主記憶から読み出して値を得る．

SSP による store 発生時：SSP で発生した store は実際には主記憶値は更新せず，再利用表内のデータを更新するのみである．このとき一般には，同一区間における store 後 load はその区間の入力とはならないため，比較の必要は生じない．ただし再利用区間が多重構造をなしている場合，外側区間 A で store した値を内側区間 B が load しうるため，B における再利用時にはその主記憶値の比較が必要となる．よって B の入力が当該アドレスを参照している場合，フラグを立てておく必要がある（図 7 (a2) の Case.1）．また，さらに内側に再利用区間 C が存在し，入力が当該アドレスを参照している場合，もし将来 B が再利用表から追い出されても比較の必要の有無を記憶しておくために，C の入力にもフラグを伝搬しておく必要がある（図 7 (a2) の Case.2）．

以上のように，SSP による store では，区間をまたぐ store 後 load が存在する場合，比較の必要が発生し，内側区間のすべてで当該アドレスのフラグを立てておく必要がある．

再利用テスト：RF 内の主記憶入力アドレスにおいて比較必要フラグが有効となっている列と同じ列に有効な値を持つ，RB 内の有効エントリに関してのみ，主記憶値の比較を行う（図 7 (b)）．比較が必要なすべての入力に関して値が一致すれば，記憶してある主記憶出力値を書き戻し，再利用を終了する．これにより，再利用率を損なうことなく，主記憶比較のためのオーバヘッドを抑えることができると考える．

5. ハードウェアモデル

本章では，再利用機構のハードウェアモデルおよびそのハードウェアコストについて説明する．

図 8 に，図 2 に示した再利用表のうち，1 つの命令区間の再利用に必要となる部分の詳細構造を示す．RF，RB 各表の V は有効エントリを示す．LRU は，エントリ入換えのヒントである．RF は命令区間の先頭アドレスおよび，読み出し /書き込みのための主記憶アドレスを保持する．また RB は，命令区間のレジスタ引数，主記憶の読み出し /書き込み値，および返り値を保持する．レジスタ値は有効フラグ V および値 Val.，主記憶値は有効バイトマスク Mask および値 Val. の組として記憶する．返り値は，%i0〜1（リーフ関数の場合 %o0〜1）または %f0〜1 に格納され，%f2〜3 を使用する返り値（拡張倍精度浮動小数点数）は対象プログラムには存在しないと仮定している．

さて，図 8 の矢印が示すように，命令区間を実行する前に，以下の手順により再利用を試みる．

(1) 命令区間の先頭アドレスを RF から検索し，
(2) RF に登録済みの命令区間であれば，対応する RB からレジスタ引数が完全に一致するエントリを検索し，
(3) 候補となった複数の RB エントリについて，主記憶値の Mask が有効であるエントリに対応して，RF に登録されている主記憶アドレスからデータを読み出し，Mask と Val. を用いて一致

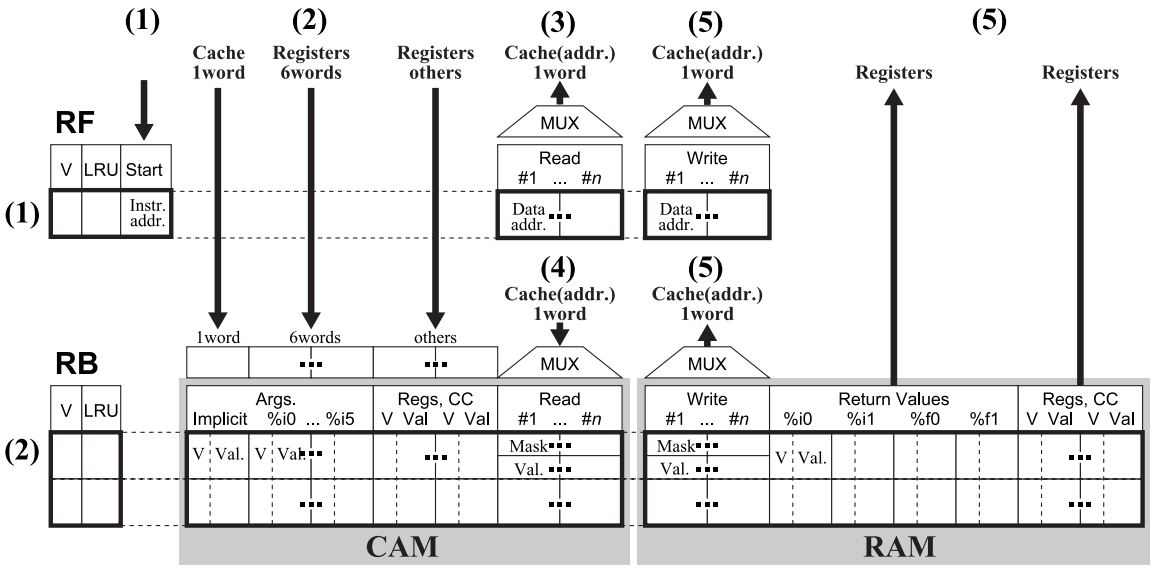

図 8 再利用表の詳細構造
Fig. 8 Structure details of reuse buffer.

比較を行う．

(4) これを，内容を比較すべき主記憶アドレスがなくなるまで繰り返す．

(5) そして，すべての入力が一致した場合，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．

再利用表本体である RB は，大きく命令区間の入力を格納する部分と，出力を格納する部分からなる．このうち前者は CAM を用いて構成し，複数の RB エントリを一度に比較することを可能としている．後者は，RAM により構成する．

本稿では入力および出力として記憶するアドレス数（図 8 中に $\#n$ として示した数）を，それぞれ 2048 と仮定した．アドレスを保持する各項は 1 word の幅があり，CAM 幅は 2 K word となる．また 1 サイクルにより連想検索可能な構成とするために，一般的なフルアソシアティブ TLB がエントリ数 256 程度であることを参考にし，RB の深さは 256 と仮定した．

さて，このような RB を構成するには 2Kword × 256 = 2MB の大きさの CAM が必要となる．これは，幅および深さは異なるものの，現在市販されている汎用 CAM と同程度のハードウェア規模である．ネットワークルータなどに用いられる汎用 CAM のうち，たとえば MOSAID 社の DC18288[30] は幅 256 bit× 深さ 64 K（総容量 2 MB）であり，文献 31) では，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いる．

なお，以下の評価では，並列事前実行機構による高速化がどこまで可能であるかを調査するため，再利用対象として記憶できる命令区間の数を多く仮定し，RF のエントリ数を 32 とした．

6. 評　　価

以上のような 3 つの手法に基づいて，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．

6.1 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．評価に用いた各パラメータを**表 1** に示す．なお，キャッシュ構成や命令レイテンシは，SPARC64-III[32] を参考にしている．

再利用機構のサイズに関しては，RF のエントリ数を 32 とした．また RF あたりの RB のサイズに関しては，前章で述べたように 1 サイクルでの検索を可能とするため深さを 256 と仮定した．市販されている汎用 CAM と同じハードウェア規模を想定し，RB の横幅に関わる主記憶アドレス数については，読み出しと書き込みのそれぞれについて 2048 アドレスとした．なお，再利用表の総容量を同じとした場合，このパラメータの組合せが最適となることを，予備評価により確認している．

6.2 SPEC95 による評価

SPEC95（train）を gcc-3.0.2（-msupersparc -O2）

表 1 シミュレーション時のパラメータ
Table 1 Simulation parameters.

命令発行幅	1	
D-Cache 容量	64 KBytes	
ラインサイズ	64 Bytes	
ウェイ数	4	
Cache ミス ペナルティ	20 cycles	
Register Windows 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	
RW 深さ	4	
RF エントリ数	32	
RB エントリ数	256 / RF	
Read アドレス	2048 / RF	
Write アドレス	2048 / RF	
RB（引数）⇔Register 比較	8 words/cycle	test(r)
RB（Read）⇔Cache 比較	4 Bytes/2 cycles	test(m)
RB（Write）⇒Cache 書込	4 Bytes/cycle	write
RB（返り値）⇒Register 書込	8 words/cycle	write
SSP ローカルメモリ	64 KBytes	

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．**図 9** として SPECint，**図 10** として SPECfp による測定結果を，それぞれ示す．

図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) および write は，表 1 に示したとおり再利用表の操作に要したサイクル数である．また，cache および window はそれぞれキャッシュミスおよび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．本稿で問題としている主記憶値の比較コストは，図中では test(m) として現れており，この test(m) が占めるサイクルが少ないほど，高速化されていることになる．

各ベンチマークプログラムのグラフに関しては，それぞれ左から順に

(M) SSP と再利用のいずれもなし
(Ma) SSP なし，全比較
(Sa) SSP 3 台，全比較
(Si) SSP 3 台，無効化
(Sp) SSP 3 台，一部比較

が要したサイクル数を表している．各サイクル数は，(M) を 1 とする正規化を行っている．

まず図 9 の SPECint では，MSP のみの(Ma) でも再利用の効果が得られているものが多い．これは，プログラム中の同じ命令区間が参照する主記憶アドレスに，ある程度重なりが存在することを示しており，無効化

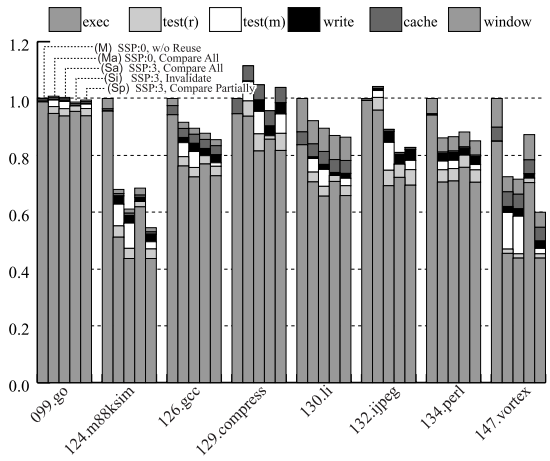

図 9 MSP が実行したサイクル数（SPECint 95）
Fig. 9 Ratio of executed cycles on MSP (SPECint 95).

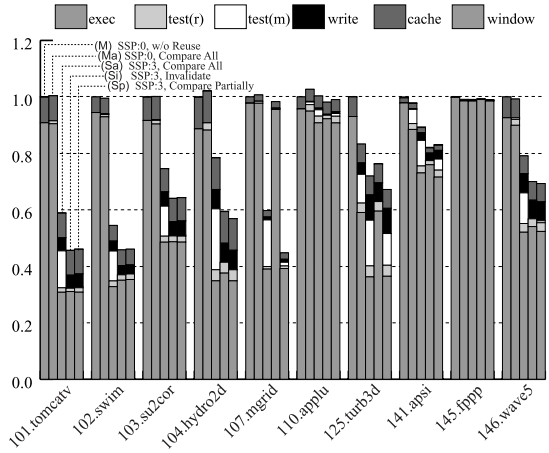

図 10 MSP が実行したサイクル数（SPECfp 95）
Fig. 10 Ratio of executed cycles on MSP (SPECfp 95).

を用いた場合，再利用率の低下が生じる．このようなプログラム（特に 124.m88ksim および 147.vortex）では，(Sp) は(Si) に対して，大きなサイクル数削減を実現できている．加えて，(Sa) において test(m) の値が大きいプログラムに関しては，(Sp) によるオーバヘッド削減の効果も大きくなり，これら 2 つの特徴をあわせ持つ 124.m88ksim では 45%，147.vortex では 40% の実行サイクル数削減率が得られた．また，SPECint における(Sp) の平均サイクル数削減率は約 20%となり，(Sa) の 15% および(Si) の 14% を上回った．

次に，図 10 に示す SPECfp では，MSP のみでは再利用の効果があまり得られていないものが多い．これは，そもそもプログラム中で同じ命令区間により参照される主記憶アドレスの重なりがほとんどないことと関係があり，エントリを無効化しても再利用率が影響

を受けにくいと推測できる．ただし，主記憶入力アドレスが単調に変化しているため SSP の導入によって得られる台数効果は大きくなると考えられる．SPECint 同様，SPECfp における(Sp) の平均サイクル数削減率は 35%となり，(Sa) の 25%，(Si) の 29%よりも良好な結果が得られた．

この結果を速度向上として計算すると，(Sp) は SPECint では最大 1.83 倍，平均 1.24 倍，SPECfp では最大 2.23 倍，平均 1.54 倍となり，Molina ら[29] の手法よりも優れた値となった．さらに，本手法はコンパイラ支援を必要とせず既存のロードモジュールに適用可能であるものの，コンパイラ支援を用いる Wu ら[28] の手法と比較しても，同等以上の結果となった．

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 SPECfp95 の 101.tomcatv，102.swim，104.hydro2d，107.mgrid 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[33]．これに対し本稿が提案する(Sp) では，上記の 4 ベンチマークでの結果は平均 52%のサイクル数削減となり，我々の MSP+SSP3 台の環境を 4 台のプロセッサによる並列実行と見なした場合，これに劣る結果となる．しかし我々の手法は，そもそも並列化が困難であるプログラムに対しても有効であり，上記の 4 ベンチマーク以外においても有効な結果が得られている．また，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というのも大きな利点である．

以上のように，(Sp) は SPECint/fp のいずれにおいても，(Sa) よりも明らかに有効な結果が得られた．また，(Si) に不利なベンチマークが多い SPECint でも有効な結果が得られた．比較的(Si) に適した SPECfp においても，(Sp) は(Si) と同等の性能を示した．これらのことより，本稿で提案する一部比較による手法が，無効化および全比較を含めた 3 つの手法のうち最も優れているといえ，その有効性が示せた．

なお，再利用率が高く exec の大幅な削減が実現できているものに関しても，キャッシュミスによるペナルティは減少していない．たとえばループの場合，個々の MSP および SSP においては入力の局所性は必ずしも高くなく，単純にキャッシュラインの 1/4 程度しか有効利用できていないと考えられる．対策としては，二次キャッシュを用意し，MSP およびすべての SSP で共有させることでプリフェッチを有効にする方法が考えられる．

7. おわりに

本稿では，区間再利用および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を提案した．また，主記憶一貫性を保つ手法として，投機的マルチスレッディングで一般的に用いられる無効化の方法に対し，我々は主記憶入力の比較を必要最小限にとどめる方法を用いることで，再利用の際のオーバヘッドを軽減する方法を示し，SPEC95 により評価を行った．

結果，一部比較による手法が，従来の無効化による手法および全比較による手法の両方の利点を生かすことで，ほとんどの場合において従来手法より良好，あるいはほぼ同等の結果が得られた．全比較では 46%，無効化では 54%であった最大サイクル数削減率においては，一部比較では 55%という値が得られた．また 1 プロセッサで再利用を行わない場合に対するサイクル数削減率の平均を算出したところ，本稿の提案する一部比較では SPECint95 で 20%となり，全比較の 15%および無効化の 14%を，SPECfp95 で 35%となり，全比較の 25%および無効化の 29%を，それぞれ上回る結果となった．

今後は，共有二次キャッシュを考慮した場合事前実行によりキャッシュミスオーバヘッドが削減できるかどうかについて検討したいと考えている．

参考文献

1) 中島康彦，津邑公暁，五島正裕，森眞一郎，富田眞治：動的命令解析に基づく多重再利用および並列事前実行，情報処理学会論文誌：コンピューティングシステム，Vol.44, No.SIG 10(ACS 2), pp.1–16 (2003).
2) Yang, J. and Gupta, R.: Energy-Efficient Load and Store Reuse, *International Symposium on Low Power Electronics and Design*, pp.72–75 (2001).
3) Sodani, A. and Sohi, G.S.: Dynamic Instruction Reuse, *Proc. 24th International Symposium on Computer Architecture*, pp.194–205 (1997).
4) González, A., Tubella, J. and Molina, C.: Trace-Level Reuse, *Proc. International Conference on Parallel Processing*, pp.30–37 (1999).
5) Costa, A.T., França, F.M.G. and Filho, E.M.C.: The Dynamic Trace Memorization Reuse Technique, *PACT*, pp.92–99 (2000).
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) Connors, D.A. and Hwu, W.W.: Compiler-Directed Dynamic Computation Reuse: Rationale and Initial Results, *32nd MICRO* (1999).
8) Connors, D.A., Hunter, H.C., Cheng, B. and Hwu, W.W.: Hardware Support for Dynamic Activation of Compiler-Directed Computation Reuse, *9th ASPLOS*, pp.222–233 (2000).
9) Huang, J. and Lilja, D.J.: Exploring Sub-Block Value Reuse for Superscalar Processors, *PACT* (2000).
10) Álvarez, C., Corbal, J., Salamí, E. and Valero, M.: On the Potential of Tolerant Region Reuse for Multimedia Applications, *ICS'01* (2001).
11) Yang, J. and Gupta, R.: Load Redundancy Removal through Instruction Reuse, *International Conference on Parallel Processing*, pp.61–68 (2000).
12) Önder, S. and Gupta, R.: Load and Store Reuse Using Register File Contents, *ICS'01*, pp.289–302 (2001).
13) 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).
14) Luk, C.: Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors, *ISCA'01*, pp.40–51 (2001).
15) Collins, J.D., Tullsen, D.M., Wang, H. and Shen, J.P.: Dynamic Speculative Precomputation, *34th MICRO*, pp.306–317 (2001).
16) Purser, Z., Sundaramoorthy, K. and Rotenberg, E.: A Study of Slipstream Processors, *33rd MICRO* (2000).
17) Ibrahim, K.Z., Byrd, G.T. and Rotenberg, E.: Slipstream Execution Mode for CMP-Based Multiprocessors, *9th HPCA* (2003).
18) Lipasti, M.H. and Shen, J.P.: Exceeding the Dataflow Limit via Value Prediction, *29th MICRO*, pp.226–237 (1996).
19) Wang, K. and Franklin, M.: Highly Accurate Data Value Prediction Using Hybrid Predictors, *30th MICRO*, pp.281–290 (1997).
20) Gopal, S., Vijaykumar, T.N., Smith, J.E. and Sohi, G.S.: Speculative Versioning Cache, *4th HPCA*, pp.195–205 (1998).
21) Marcuello, P., González, A. and Tubella, J.: Speculative Multithreaded Processors, *ICS*, pp.77–84 (1998).
22) Marcuello, P., Tubella, J. and González, A.: Value Prediction for Speculative Multithreaded Architecture, *32nd MICRO*, pp.230–237 (1999).
23) Oplinger, J.T., Heine, D.L. and Lam, M.S.: In Search of Speculative Thread-Level Parallelism, *PACT*, pp.303–313 (1999).
24) Codrescu, L., Wills, D.S. and Meindl, J.: Architecture of the Atlas Chip-Multiprocessor: Dynamically Parallelizing Irregular Applications, *IEEE Trans. Comput.*, Vol.50, No.1, pp.67–82 (2001).
25) Marcuello, P. and González, A.: Thread-Spawning Schemes for Speculative Multithreading, *8th HPCA*, pp.55–64 (2002).
26) Roth, A. and Sohi, G.S.: Register Integration: A Simple and Efficient Implementation of Squash Reuse, *33rd MICRO* (2000).
27) Roth, A. and Sohi, G.S.: Speculative Data-Driven Multithreading, *7th HPCA*, pp.37–50 (2001).
28) 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).
29) Molina, C., González, A. and Tubella, J.: Trace-Level Speculative Multithreaded Architecture, *ICCD* (2002).
30) MOSAID Technologies Inc.: *Feature Sheet: MOSAID Class-IC DC18288*, 1.3 edition (2003).
31) MUSIC SEMICONDUCTORS: *Using The MU9C1965A LANCAM MP For Data Wider Than 128 Bits* (1998).
32) HAL Computer Systems/Fujitsu: *SPARC64-III User's Guide* (1998).
33) 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 15 年 5 月 9 日受付)
(平成 15 年 10 月 5 日採録)

津邑 公暁（正会員）

1973 年生．1996 年京都大学工学部情報工学科卒業．1998 年同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単位取得退学．同年より同大学経済学研究科助手，現在に至る．計算機アーキテクチャ，並列処理応用，脳型情報処理に興味を持つ．人工知能学会，日本神経回路学会各会員．

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 年同大学院修士課程修了．同年富士通入社．スーパコンピュータ VPP シリーズの VLIW 型 CPU，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．工学博士．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，現在に至る．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）．計算機アーキテクチャに興味を持つ．IEEECS，ACM 各会員．

五島　正裕（正会員）

1968 年生．1992 年京都大学工学部情報工学科卒業．1994 年同大学院工学研究科情報工学専攻修士課程修了．同年より日本学術振興会特別研究員．1996 年京都大学大学院工学研究科情報工学専攻博士後期課程退学，同年より同大学工学部助手．1998 年同大学大学院情報学研究科助手．高性能計算機システムの研究に従事．2001 年情報処理学会山下記念研究賞，2002 年同学会論文賞受賞．IEEE 会員．

森　眞一郎（正会員）

1963 年生．1987 年熊本大学工学部電子工学科卒業．1989 年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了．1992 年同大学大学院総合理工学研究科情報システム学専攻博士課程単位取得退学．同年京都大学工学部助手．1995 年同助教授．1998 年同大学大学院情報学研究科助教授．工学博士．並列/分散処理，可視化，計算機アーキテクチャの研究に従事．IEEE，ACM 各会員．

富田　眞治（正会員）

1945 年生．1968 年京都大学工学部電子工学科卒業．1973 年同大学大学院博士課程修了．工学博士．同年京都大学工学部情報工学教室助手．1978 年同助教授．1986 年九州大学大学院総合理工学研究科教授，1991 年京都大学工学部教授，1998 年同大学大学院情報学研究科教授，現在に至る．計算機アーキテクチャ，並列処理システム等に興味を持つ．著書「並列コンピュータ工学」（1996），「コンピュータアーキテクチャ第 2 版」（2000）等．電子情報通信学会，IEEE，ACM 各会員．平成 7 年，8 年度，10 年，11 年度本会理事．平成 13 年，14 年度同関西支部長．