計算機アーキテクチャ 147－28
ハイパフォーマンス
コンピューティング 89－28
（2002．3．8）

関数値再利用および並列事前実行による高速化技術の提案と評価

緒方勝也[†1] 正西申悟[†1] 中島康彦[†2]
五島正裕[†4] 森眞一郎[†4]
北村俊明[†3] 富田眞治[†4]

SPARC Application Binary Interfaceにしたがって記述されたプログラムについて，コンパイラによる専用命令の埋め込みを必要とすることなく，ハードウェアによる関数レベルの値再利用および並列事前実行が可能であることを示す．また，Stanfordベンチマークを用いて，その効果を定量的に評価分析する．

A Speedup Technique with Function Level Value Reuse and Parallel Precomputation

Katsuya Ogata,[†1] Shingo Masanishi,[†1] Yasuhiko Nakashima,[†2]
Masahiro Goshima,[†4] Shinichiro Mori,[†4] Toshiaki Kitamura[†3]
and Shinji Tomita[†4]

This paper describes how to apply the function-level value-reuse and parallel precomputation technique against the programs based on SPARC application binary interface without any special instructions. We evaluate the effectiveness quantatively with Stanford benchmark programs.

1. はじめに

我々は，関数値を再利用するとともに，将来実行されるであろう関数およびパラメタを予測し，事前に実行しておくことにより高速化を図る基礎的技術の確立をめざしている．事前実行機構は，再利用表（メモリ）自身がパラメタの時系列変化から将来のパラメタを予測する機構と，複数の命令処理装置（プロセッサ）が予測パラメタに基づいて関数を実行し結果を再利用表に登録する機構の組み合わせにより構成される．

本稿では，前半において，SPARC Application Binary Interface （以下SPARC ABIと略する）にしたがって記述された，ある一定の条件を満たすプログラムに対して，専用命令を追加することなく関数値再利用を適用できることを示し，再利用機構および事前実行機構の詳細について述べる．後半では，Stanfordベンチマークを用いた評価を行う．

値再利用（以下，再利用と略する）は，プログラムの一部分に関する入力値および出力値を再利用表に登録しておく．同じ箇所を再度実行する時，入力値が既知である場合には，途中の命令を実行することなく，正しい出力値を直ちに求めることができる．本方式の特長は，1）入力値さえ一致すれば，実行結果を検証する必要がない；2）入力値および出力値の総数によってのみ，ハードウェアコストが決定され，省略可能な命令列の長さを制約しない；3）命令間の依存関係の多少は，再利用機構の複雑さに影響を与えない；ことである．副次的な効果として，冗長なロード/ストア命令や消費電力を削減できることも報告されている[2),3)]．

ただし，近年報告されている再利用の具体的実現方法[4),5)]は，プロセッサに専用命令を追加し，コンパイラが再利用を行うための命令列を生成することを前提としている．これは，プロセッサが動的かつ効率良く基本ブロックを切り出すことが難しく，簡単化のためには，コンパイラが基本ブロックの範囲をハードウェアに伝達しなければならないためである．残念ながら，専用命令を前提とする場合は，既存ロードモジュールをそのまま高速化することができないという問題が生じる．また，プリフェッチ機構のないキャッシュと同様，過去の実行結果を登録するだけの単純な再利用では，入力が単調に変化する場合に効果がない．すなわち，既存ロードモジュールの高速化とヒット率向上が重要な課題である．

2. SPARC ABIに基づく再利用

既存ロードモジュールに対して再利用を適用するには，命令列から，入力と出力を明確に特定できる命令

†1 京都大学工学部情報学科
Department of Information Science, Faculty of Engineering, Kyoto University
†2 京都大学大学院経済学研究科
Graduate School of Economics, Kyoto University
†3 京都大学総合情報メディアセンター
Center for Information and Multimedia Studies, Kyoto University
†4 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University

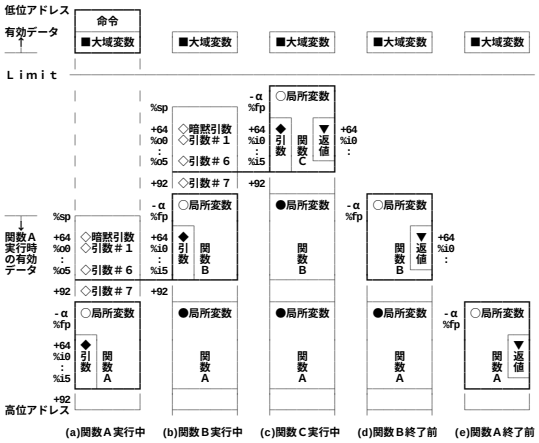

図 1　引数およびフレームの概略
Fig. 1　Overview of parameters and frames.

区間を切り出す必要がある．さらに，効果を上げるためには，命令区間が多くの命令を含むことが望ましい．このようなことから，我々は，関数を再利用の単位とした．引数を特定するために，プログラムが，SPARC ABI に規定されている以下の条件を満たすものと仮定した．なお，%fp はフレームポインタ，%sp はスタックポインタを意味する．

- スタック上の有効データは%sp 以上の範囲である．
- %sp から 16 ワードはレジスタ退避空間であり関数の入出力には関連しない．
- %sp+64 の 1 ワードは構造体を返り値とするための暗黙的引数である．
- %sp+68 から 6 ワードは引数の一時退避空間であり関数の入出力には関連しない．
- レジスタ%o0〜5，および，%sp+92 以上に関数への明示的引数が格納される．

さらに，大域変数と局所変数（フレーム内変数）を区別するために以下を仮定した．

- 大域変数は Limit（実行時の固定アドレスを仮定）未満の範囲に配置される．
- %sp が Limit 未満になることはない．
- Limit 以上%sp 未満のデータは無効．

前述の条件を満たしながら，関数 A が関数 B を呼び出し，さらに，関数 B が関数 C を呼び出す場合（以後 A，B，C と略する）の，引数およびフレームの概略を**図 1** に示す．

(a) は A 実行中の状態である．Limit 未満の太枠部分に命令および大域変数，また，%sp 以上に有効な値が格納されている．%sp+64 には，B が構造体を返り値とする場合の暗黙的引数として，構造体の先頭アドレスが格納される．B への明示的な引数は，先頭の 6 ワードがレジスタ%o0〜5，第 7 ワード以降は%sp+92 以上に格納される．ベースレジスタを%sp とするオペランド%sp+92 が使用された場合，この領域は B への第 7 引数，すなわち B の局所変数である．

(b) は B 実行中の状態である．A と同様に大域変数（■）および引数（◆）を入力とする．ポインタを通じて他の大域変数や A の局所変数（●）も入力となり得る．B の局所変数には，引数の先頭 6 ワードのアドレスを扱うために必ず確保される%fp+68〜91，および，第 7 ワード以降が存在する場合に確保される%fp+92 以上の領域が含まれる．ただし，%fp 相対アドレスにより参照されるとは限らないため，一般に，%fp+92 以上の領域が A の局所変数か B の局所変数かの区別ができない．(a) においてオペランド%sp+92 以上が出現した場合に，次に呼び出される B に第 7 ワード以降が存在すると考える．出現頻度が低いと予想されることおよび簡単のために，第 7 ワード以降を検出した関数は再利用の対象外とする．

(c) はリーフ関数 C を実行している状況である．C の入力は，大域変数，および，save 命令の有無によりレジスタ%i0〜5 または%o0〜5，同様にアドレス%fp+α または%sp+α（α≧0）に格納されている引数である．一方 C の出力は，大域変数，および，save 命令の有無によりレジスタ%i0〜1 または%o0〜1 に格納される返り値である．なお，返り値が浮動小数点数の場合は%f0〜1，構造体の場合は%fp+64 または%sp+64 に格納されている構造体先頭アドレスへ書き込まれる．ポインタを通じて，大域変数および AB を含む上位関数内の局所変数も入出力となり得る．(d) および (e) は同様に B および A の終了直前の状況である．

さて，(b) において，C に対する入力が既知であり，対応する出力が計算済みである場合，再利用により (c) を省略することができる．ただし，(b) の時点では C の局所変数が場所として存在しないため，再利用を行う際に C の局所変数を参照してはならない．C 実行時に C の入出力として登録すべきフレーム上データは AB の局所変数である．この区別には，前述のように引数の第 7 ワード以降を扱わない場合，AB の局所変数が (b) における%sp+92 以上に対応することを利用する．

同様に，(a) において，B に対する入力が既知の場合，(b)(c)(d) を一度に省略することができる．B 実行時に B の入出力として登録すべきフレーム上データは A の局所変数であり，途中の C 実行時に B の入出力として登録すべきフレーム上データも A の局所変数である．

以上のように，同じく C の実行中であっても，どのレベルの関数を登録中であるかにより，B の局所変数が関数の入出力に含まれるか否かが異なる．この区別には，B の局所変数が，(a) では%sp+92 未満，(b) では%sp+92 以上であることを利用する．すなわち，登録開始時点の%sp を記憶しておくことにより，C を実行中に，B，C それぞれの関数として登録すべき入出力を特定することができ，複数レベルの登録作業を同時に行うことができる．

3. 再利用機構の構成と動作

前述した再利用を実現するための再利用表の論理構成を**図 2** に示す．再利用表は，再利用ウィンドウ（RW），関数管理表（RF），本体（RB）からなる．RW は現在実行中かつ登録中である関数呼び出しの入れ子関係を表現しており，各々の関数呼び出しに対応する RF および RB のエントリを指している．RF の各エントリは互いに異なる関数に対応しており，V=有効エントリを表示；LRU=エントリ入れ換えのヒント；関数アドレス=関数の先頭アドレス；Read=読み

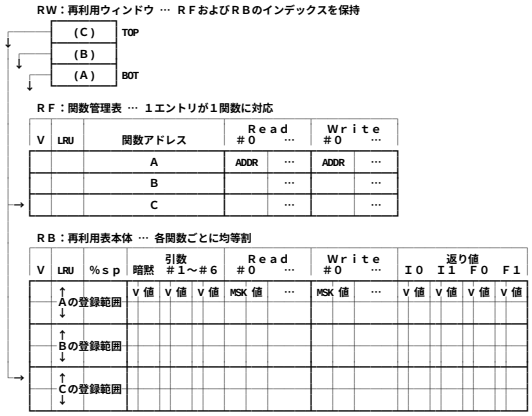

図 2　再利用表の論理構成
Fig. 2　Logical structure of reuse-buffer.

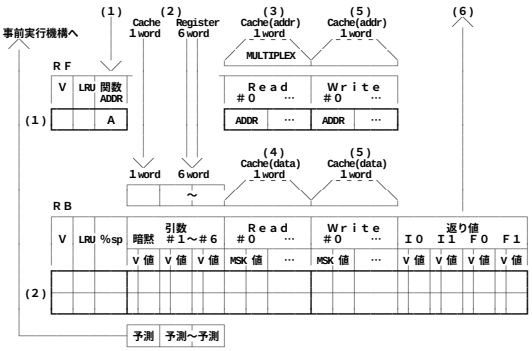

図 3　再利用表の物理構成
Fig. 3　Physical structure of reuse-buffer.

出しアドレス（4 の倍数）; Write=書き込みアドレス（4 の倍数）; から構成される．RB は RF の各エントリに対応する複数エントリがブロック化されており，V=有効エントリを表示；LRU=エントリ入れ換えのヒント；%sp=前述の関数呼び出し時の%sp；引数=有効エントリを示す V および入力値；Read=RF の各 Read アドレス 4 バイトの有効バイトを示すマスクおよび入力値；Write=同じく各 Write アドレスに関するマスクおよび出力値；返り値=汎用レジスタまたは浮動小数点レジスタに格納される出力値；から構成される．%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ものと仮定する．Read アドレスの内容と RB の複数エントリを一度に比較するために，Read アドレスは RF において一括管理し，RB ではマスクおよび値のみを管理している．

次に，RF の 1 エントリ分に対応する再利用表の物理構成を**図 3** に示す．引数および主記憶読み出しデータと RB の内容との比較には CAM を利用する．関数呼び出しが再利用可能か否かを判定するためには，まず，関数アドレスが一致する RF エントリ（1）を特定し，次に，引数がすべて一致する RB エントリ（2）を特定する．さらに，少なくとも 1 つのマスクが有効である Read アドレス（3）を RF から順に選択し，主記憶から読み出した各 4 バイトのデータ（4）と RB の対応する列に属するすべての値との比較を行う．マスクが有効であるすべての値が一致したとき，書き込みデータ（5）および返り値（6）を主記憶およびレジスタへ格納する．引数の予測機構については後述する．

続いて，命令実行手順について詳述する．

3.1　関数呼び出し

関数呼び出しの契機は，call 命令または%o7 へ現 PC を書き込む jmpl 命令である．RF を参照し，前述の手順により再利用を試みる．また，再利用した関数（例えば C）を含む上位関数 AB が登録中である場合，C の再利用を行った RB エントリの内容のうち，主記憶参照に関する部分を AB それぞれに対応する登録中 RB エントリに追加する．ただし前述のように，AB それぞれの呼び出し時における%sp+92 未満に対する参照は対象外とする．

ところで，以上の方法により入れ子の関数を登録すると，より上位の関数（例えば B）において登録すべき主記憶参照箇所が RB の各エントリが収容可能な数を越える．この場合は，制限を越えた関数を含む上位関数 AB それぞれに対応する登録中 RB エントリおよび RW エントリを無効化し，引続き登録可能な C のみを RB および RW に残して登録を続行する．

一方，再利用できなかった場合，まず，関数が RF に登録されていなければ LRU アルゴリズムに基づいて RF に新規登録する．次に，RF の該当エントリに対応する RB のブロックに LRU アルゴリズムに基づいて新たなエントリを確保し，RW に，これから実行しようとする関数，すなわち，RF および RB の該当エントリを積む．このとき，RW に登録可能な上限値を越えた場合，最も上位の関数に対応する RB エントリおよび RW エントリを無効化する．

3.2　関数本体の実行

関数値の再利用ができない場合，関数本体の実行を開始する．ただし，以後の再利用に備えて，各命令の実行と同時に，再利用に必要な関数の入力および出力を RB に登録していく．命令の種類に応じた動作は以下のとおりである．

【trap 命令】 システムコールを含む関数は再利用できないと判断し，RW が保持している実行中のすべての関数について，RB エントリおよび RW エントリを無効化する．

【レジスタ参照】 レジスタ%i0～5（save 命令を伴わない関数では%o0～5）には明示的引数の先頭 6 ワードが格納される．関数内においてまず読み出しを行った対象を引数として RB へ登録する．まず書き込みを行った対象は引数ではないため比較対象外として登録する．ただし，%i0～1（save 命令を伴わない関数では%o0～1），および，%f0～1 への書き込みは，返り値の可能性があるので返り値としても RB へ登録する．その他のレジスタ参照は，関数への入力から得られる中間結果であるため，登録は不要である．

【Limit 以上，呼び出し時%sp+64 未満】 前述したように無効領域または関数の局所変数であり，RB への登録は不要である．

【呼び出し時%sp+64】 暗黙的引数が格納される．関数内においてまず読み出しを行った場合，引数として RB へ登録する．まず書き込みを行った場合は引数で

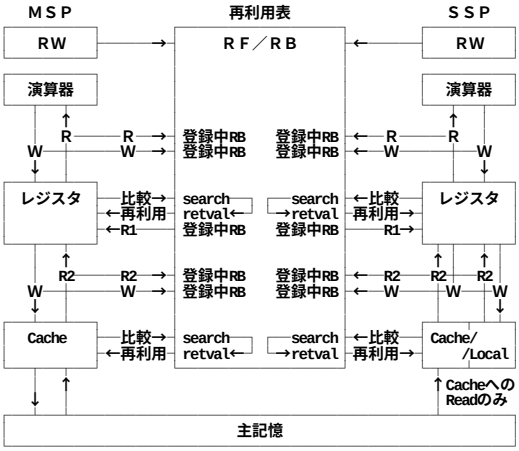

図 4　並列事前実行機構
Fig. 4　Logical structure of precomputation.

はないため比較対象外として登録する．

【呼び出し時%sp+68 以上】　%sp+68〜91 は局所変数である．%sp+92 以上への書き込みを検出した場合，引数の第 7 ワード以降が存在するため，現関数から復帰する前に次の関数が呼び出された場合，次の関数を含む上位関数の登録を中止する．

【その他の主記憶参照】　上記以外の場合，対象は，大域変数，上位関数の局所変数のいずれかである．これらは，関数に対する入出力として登録が必要である．RW に登録されている全ての RF/RB エントリについて，以下を行う．

(1)　Limit 以上，各 RB の%sp+92 未満であるアドレスは無視する．

(2)　読み出しアドレスが，Write または Read として既登録である場合は，内容がすでに上書きされたか，または，登録済であるため，新たな登録は行わない．

(3)　書き込みアドレスが，Write として既登録である場合は，登録内容を更新する．

(4)　未登録の場合，登録数の上限を越えていなければ，Read/Write に応じて登録を行う．上限を越えた場合，その関数を含む上位関数の登録を中止する．

3.3　復　　帰

関数からの復帰の契機は，%g0 へ現 PC を書き込み，%o7 または%i7 の内容へ無条件分岐する jmpl 命令である．登録中の RB エントリを有効にし，最後に RW から該当エントリを削除する．

4.　並列事前実行機構の構成と動作

これまでに述べた単純な再利用では，RB エントリの生存時間よりも同一パラメタが出現する間隔が長い場合や，パラメタが単調に変化し続ける場合に全く効果がない．我々は，通常どおり再利用を行いながら命令列を実行するプロセッサ（Main Stream Processor：以下 MSP と略する）とは別に，先行して RB エントリへの登録を行うプロセッサ（Shadow Stream Processor：以下 SSP と略する）を複数個設けることにより，さらなる高速化が可能ではないかと考えた．

並列事前実行機構の概要を**図 4** に示す．RW，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，RF，RB，主記憶は全プロセッサが共有する．R はレジスタやキャッシュからの読み出しおよび RB への登録，W はレジスタやキャッシュへの書き込みおよび RB への登録を表している．R1 は，すでに RB へ登録されているアドレスからの読み出しは，キャッシュではなく RB からデータを直接得ることにより，キャッシュを汚さない工夫である．R2 は，RB に未登録のアドレスは，従来通りキャッシュを参照することに対応する．

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

D-Cache 容量	64 Kbyte
ラインサイズ	64 byte
ウェイ数	4
Cache ミスペナルティ	20 cycle
Register-Window	6 set
Window ミスペナルティ	20 cycle/set
ロードレイテンシ	2 cycle
整数乗算〃	8 cycle
整数除算〃	70 cycle
浮動小数点加減乗算〃	4 cycle
単精度浮動小数点除算〃	16 cycle
倍精度浮動小数点除算〃	19 cycle
RW の最大深さ	6
RF のエントリ数	16
RB(引数) ⇔ Register 比較	1 cycle
RB(Read) ⇔ Cache 比較	4 byte/cycle
RB(Write) ⇒ Cache 書き込み	4 byte/cycle
RB(返り値) ⇒ Register 書き込み	1 cycle
SSP 局所メモリ容量	64 Kbyte

3章に述べたように，MSP はレジスタや主記憶に対する参照を RB に登録しながら通常どおり命令列を実行し，可能であれば関数呼び出し時に再利用を行う．これに対し SSP は，特定の関数のみを実行し，SSP が有する局所メモリおよび RB への読み書きは行うものの，キャッシュおよび主記憶への書き込みは行わない．MSP と異なる点は以下の通りである．

4.1　関数の選択

過去に RB への登録を行ったことがあり，かつ，RB への登録回数が多いにも関わらず再利用時のヒット率が低い関数を事前実行の対象とする．図 3 に示すように，RB の参照履歴をもとに，引数の時系列変化を外挿し，将来出現するであろう引数を予測する．本稿では，最近出現した 2 組の引数の差分に基づいて，ストライド予測を行っている．予測した引数が SSP のレジスタに格納された後，SSP が関数の実行を開始する．

4.2　主記憶の参照

関数フレームに対する参照には SSP ごとに設けた局所メモリを用いる．これら以外の主記憶読み出しは，MSP と共有する主記憶から，SSP ごとに設けたキャッシュを経由して行う．主記憶書き込みは一切行わず，書き込みアドレスおよびデータは RB に登録する．局所メモリの容量は有限であり，関数フレームの大きさが局所メモリを越えた場合には，その関数の実行を打ち切る．また，0 番地の参照など，実行を継続できない例外が発生した場合も，実行を打ち切る．

4.3　事前実行の終了

jmpl 命令により，選択した関数の実行が終了した場合，MSP と同様，登録中の RB エントリを有効に

し，RW から該当エントリを削除する．そして，次に事前実行する関数を選択し，以上を繰り返す．

4.4 RB エントリの入れ替えアルゴリズム

MSP だけでは再利用ができず，パラメタの変化が予測できないために SSP の効果もない関数や，入出力データが極めて多い，システムコールを含む，入れ子のレベルが RW の容量を越えて深過ぎるなど，そもそも RB への登録が不可能である関数を除くと，再利用が可能な関数は以下の 3 つに分類できると予想される．

【第 1 種関数】 パラメタの変化が小さく，MSP のみの再利用でも十分高速化が可能なもの．

【第 2 種関数】 パラメタの変化が大きく，MSP のみでもある程度高速化が可能であるものの，SSP によりさらに高速化できるもの．

【第 3 種関数】 パラメタが単調変化し，MSP だけでは全く効果がなく，SSP により高速化できるもの．

MSP が登録した RB エントリが全く再利用されない場合，第 3 種関数であると判断し SSP の実行対象に加える．一度 MSP により再利用されたエントリは再び再利用されることはないと考え，SSP は FIFO に従って RB エントリの入れ替えを行う．逆に，SSP が登録した RB エントリが全く再利用されない場合，第 1 種関数であると判断し SSP の実行対象から除く．MSP，SSP いずれの登録エントリも再利用される場合には，第 2 種関数であると判断し，SSP による実行を継続する．このような状況に対応するために，RB を 2 分割し，MSP，SSP それぞれが使用する領域を分けた上で，MSP は常に LRU，SSP は常に FIFO に基づいて RB エントリを入れ替えている．

5. Stanford ベンチマークを用いた評価

評価には，再利用機構を搭載した単命令発行の SPARC-V8 アーキテクチャ・シミュレータを用いた．シミュレータの諸元を**表 1** に示す．キャッシュ構成や命令レイテンシは HAL の SPARC64[1)]を参考にした．測定対象は Stanford ベンチマークを 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である．ただし，FFT と Queens において，単に同じ処理をそれぞれ 20 回と 50 回繰り返している最外ループは，再利用の効果が無意味に高く現れないよう，各 1 回に変更した．**図 5** に，再利用を適用しない場合の総実行命令ステップ数（複数サイクルを要するものはサイクル数を加算）を 1 とした場合の，各構成における総実行命令ステップ数の比（縦軸）を示す．横軸は，MSP1 台を含むプロセッサ数（1，2，4，8）／RF あたりの RB エントリ数（32，64，128，256），また，凡例の MA は，RF に登録可能な Read アドレスおよび Write アドレス各々のエントリ数（64，256，1024，4096）である．なお，FFT および Bubble は乱数発生以外に関数呼び出しが無くステップ数の減少が最大 4%程度，Queens および Quick は最大 10%程度であるため，それぞれ省略している．FFT，Bubble，Quick における減少は，第 3 種関数による乱数発生が SSP により高速化されることによる．第 3 種関数が多い Trees，Intmm，Mm では，MSP のみでは再利用の効果がないものの，MA=4096 の場合には SSP により 15〜75%のステップ数を削減できている．Intmm および Mm の最内ループが積和を求める関数を呼び出しており，この部分の並列事前実行が寄与している．第 1 種関数の再帰呼び出しが多い Perm，Towers，Puzzle では，MSP のみでも 30〜75%のステップ数を削減できており，SSP を投入してもほとんど効果がない．ところで，Puzzle の MA=256 および MA=1024 において，RB エントリ数の増加につれて再利用の効果が下がっている．これは，後述の**図 6** に示すように，Read/Write アドレス数が極めて多い RB エントリが存在することに起因する．RB エントリ数を増やすことにより，このようなエントリが RB から追い出されずに残ると，RF に割り当てられているアドレス領域が枯渇し，有効な RB エントリが減少する．

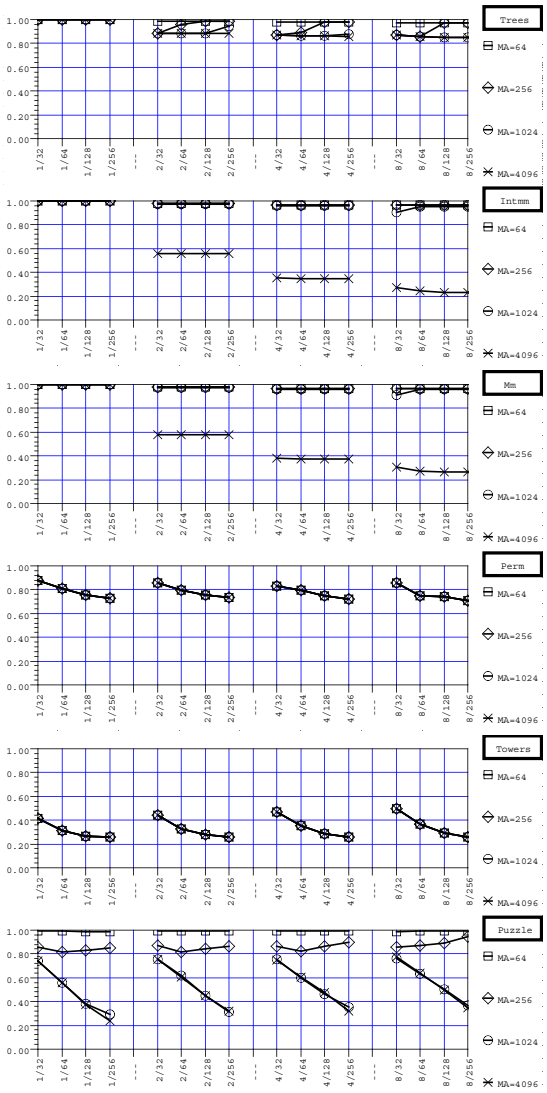

図 5 MSP の実行ステップ数
Fig. 5 Steps of MSP.

図 6 は，プロセッサ数=8，RB エントリ数=256，

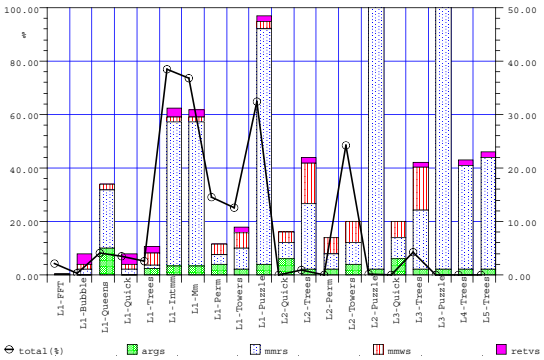

図 6　関数の深さごとの入出力数
Fig. 6　Relation between depth and I/O.

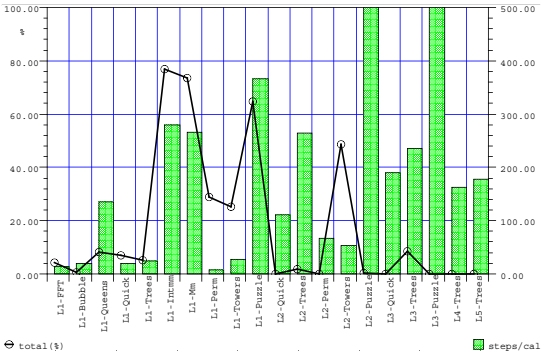

図 7　関数の深さごとの再利用ステップ数
Fig. 7　Relation between depth and reused steps.

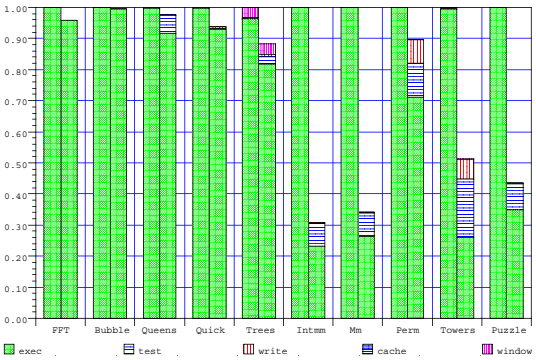

図 8　MSP の実行サイクル数
Fig. 8　Cycles of MSP.

MA=4096 の構成において，再利用可能であった関数呼び出しを入れ子の深さ（L1〜L6）ごとに分類し，再利用を適用しない場合のステップ数に対する削減ステップ数の比を折れ線グラフ（左目盛），また，引数（args），Read アドレス（mmrs），Write アドレス（mmws），返り値（retvs）の平均個数を棒グラフ（右目盛）により示したものである．上端を越えている L2-Puzzle は mmrs=173，mmws=1，retvs=1，L3-Puzzle は mmrs=341，mmws=5，retvs=1 である．深さ 4 以上になると削減ステップ数の比がほぼ 0 となり，入れ子の深さは 3 程度までを考慮すればよいこと，引数の個数はほぼ 2 以下であること，また，削減ステップ数が 2%を越えるものは，Read/Write アドレスの平均個数は 44/8 を越えないことがわかる．

同様に**図 7** は，図 6 と同じ折れ線グラフ（左目盛）に，1 回の関数呼び出しあたりの平均削減ステップ数（右目盛）を重ねたものである．上端を越えている L2-Puzzle は 1633，L3-Puzzle は 3998 である．削減ステップ数が 2% を越えるものだけを見ても，300 ステップ程度の関数呼び出しを再利用できている．

さて，再利用により実際に高速化を達成するには，再利用に伴うオーバヘッドを見極める必要がある．**図 8** は，命令ステップ数（exec）に，表 1 に示した RB(Read) ⇔ Cache 比較（test），RB(Write) ⇒ Cache 書き込み（write），キャッシュミス（cache），レジスタウィンドウミス（window）の各オーバヘッドを加えたサイクル数の内訳である．左側棒グラフは再利用を適用しない場合，右側棒グラフは再利用を適用した場合の内訳である．オーバヘッドのほとんどは test である．RB(Read) ⇔ Cache 比較を 4byte/cycle から 8byte/cycle に増加させるなど，比較の高速化が課題であると言える．

6. おわりに

本稿では，関数値を再利用するとともに，将来実行されるであろう関数およびパラメタを予測し，事前に実行しておくことにより高速化を図る手法を提案した．Stanford ベンチマークによる評価では，プログラムによる効果の差は大きいものの，それぞれの性質に合った効果が得られている．現在，SPEC ベンチマークなど他のプログラムに対する効果を調査中である，今後は，回路レベルの実現方法について研究を進める予定である．

謝辞　本研究の一部は文部科学省科学研究費補助金（基盤研究 (B)(2) 課題番号 12480072，12558027 ならびに 13480083）による．

参 考 文 献

1) FUJITSU/HAL SPARC64-III User's Guide, www.sparc.com/standards/ (1998).
2) Yang J. and Gupta R.: Load Redundancy Removal through Instruction Reuse, 2000 ICPP (2000).
3) Yang J. and Gupta R.: Energy-efficient load and store reuse, International Symposium on Low Power Electronics and Design, pp. 72–75 (2001).
4) Connors D.A., Hunter H.C., Cheng B.C. and Hwu W.W.: Hardware Support for Dynamic Activation of Compiler-Directed Computation Reuse, 9th ASPLOS, pp. 222–233 (2000).
5) Huang J. and Lilja D.J.: Extending Value Reuse to Basic Blocks with Compiler Support, IEEE Trans. on Comp., Vol.49, No.4 (2000).