RC-010

自動メモ化プロセッサにおけるアドレス比較削減手法

高木伴彰† 島崎裕介† 新美明仁†
津邑公暁† 松尾啓志†

我々は，計算再利用技術に基づく自動メモ化プロセッサを提案している．自動メモ化プロセッサは過去の関数の実行結果を利用することで関数の実行を省略し，高速化を図る．関数の入力を過去の入力と一致比較することで，再利用の可否は判断される．自動メモ化プロセッサはこの一致比較対象に無用なアドレス情報を含めているために，本質的に再利用可能な場合でも再利用ができないという問題がある．そこで，本稿では無用なアドレス情報を動的に検出，削減する機構を追加したことで自動メモ化プロセッサの更なる高速化が達成できたことを示す．汎用 GA ソフトウェア GENEsYs で評価を行い，従来手法に比べて，平均約 2.7%，最大約 34%の実行時間の削減ができたことを確認した．

Reducing the Comparison of Input Addresses on Auto-Memoization Processor

Tomoaki Takagi,† Yusuke Shimazaki,† Akihito Niimi,†
Tomoaki Tsumura† and Hiroshi Matsuo †

We have proposed an auto-memoization processor based on computational reuse technology. This processor skips function execution by reusing past result of the very function and gains speedup. Whether reuse can be applied or not is judged by comparing inputs of the function with inputs at past. There have a problem that auto-memoization processor compares unnecessary input addresses wastefully, and sometimes reduces reuse rate. In this article, We show a mechanism for reducing unnecessary address comparison dynamically. The result of experiment with GENEsYs: a general purpose GA software shows that our method reduces up to 34% execution cycles and 2.7% on average.

1. はじめに

動作クロックを向上させるためにパイプラインの多段化は進み，それに伴い増大するハザードによるペナルティを緩和するために命令レベル並列性 (ILP:Instruction Level Parallelism) に着目した研究が行われてきた．また，プロセスの微細化により 1 チップに複数の CPU コアを搭載することが可能となったことで，スレッドレベル並列性 (TLP:Thread Level Parallelism) による高速化手法が研究されるようになった．これら並列性に着目した手法はプロセッサの IPC を向上させることで高速化を達成する．一方，プロセッサに命令実行を省略させることで高速化を図る計算再利用という技術がある．我々はこれまでに自動メモ化プロセッサという計算再利用技術のモデルを提案してきた[1)]．自動メモ化プロセッサは，実行時に関数の入出力を表に記憶しており，再び同関数が呼び出されたときに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．本稿では，この一致比較対象から無用なアドレス情報を削減することで自動メモ化プロセッサ更に高速化する手法を提案する．

† 名古屋工業大学
Nagoya Institute of Technology

2. 自動メモ化プロセッサ

2.1 概要と動作モデル

図 1 に自動メモ化プロセッサの構成を示す．自動メモ化プロセッサは，プログラムの実行時に関数の入出力を MemoBuf に記憶する．関数の入力とは，レジスタを介して受け渡しされる引数と，関数実行中に主記憶から読み出されてくる値を指し，出力とは，レジスタを介して受け渡しされる返り値と，関数実行中に主記憶へ書き込まれる値を指す．関数の実行終了時に MemoBuf に記録された入出力は MemoTbl に書き込まれる．再び同じ関数が実行される時に，関数の入力と MemoTbl に記録されている過去の入力とを比較する．入力が一致した時には対応する出力をレジスタおよびキャッシュに書き戻し，当該関数の実行は省略される．MemoTbl は以下の要素で構成される．

RF 関数の開始アドレス等の関数情報を記憶する RAM
RB 関数の入力セットを記憶する CAM
RA 関数の入力セットのアドレスを記憶する RAM

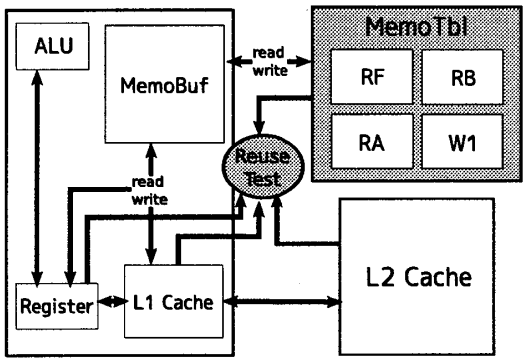

図 1　自動メモ化プロセッサ概要

W1　関数の出力を記憶する RAM

入力の一致比較は RB とレジスタ，キャッシュ間で行われるため，RB を CAM(Contents Addressable Memory)[2] で構成することで高速な一致比較が行えるようになっている．

2.2 問 題 点

自動メモ化プロセッサには，関数が特定の入力パターンを取るとき，本質的には再利用可能であるにも関わらず，再利用できないという場合がある．

例えば，配列へのポインタを引数に受取り，その配列の要素の合計値を返す関数 f を自動メモ化プロセッサが実行する場合を考える．過去に関数 f を実行した時に参照された配列と，要素の値がすべて等しい配列へのポインタを引数に関数 f が呼び出された時，再利用は可能であるかのように思われる．しかし，この場合，自動メモ化プロセッサは再利用を行うには配列の要素の一致だけでなく，その配列のポインタの一致までを要求するので，配列へのポインタが異なってしまうと再利用対象から外れてしまう．入力の一致比較は当然同じ変数同士で行うが，これを保証するために現在は，参照アドレスの一致を確認した上で値比較を行うという実装になっているためである．

このように，本質的には再利用可能であるにも関わらず，関数の処理結果に影響を与えないアドレス情報が一致せず，再利用ができないという状況が生まれてしまっている．

3. 提 案 手 法

3.1 概　　要

以上の問題点を解決するために，本稿では無用なアドレス情報の一致比較を削減することで，再利用率を向上させる手法を提案する．

MemoTbl 内に無用なアドレス情報が登録されているのは，関数入力のアドレスを絶対番地のみで登録していたことが原因である．そこで，本稿では関数の入力のアドレスをベース修飾指定の形式でも登録可能とすることで，無用なアドレス情報の登録を削減する．具体的には，2.2 で挙げた関数 f において関数内で参照される配列要素を，アドレスを引数の受け渡し用レジスタ+ オフセットという形式で MemoTbl に登録する．一方，ポインタである引数自体は MemoTbl に登録しない．このようにすることで、無用なアドレス情報の一致比較の回数は減り、再利用率の向上が期待できる。

提案手法の実現には従来の自動メモ化プロセッサに対して MemoTbl の構造変更とベースアドレスの検出機構の追加が必要となる．MemoTbl の構造変更は入力のアドレス登録方式が,「従来の絶対番地アドレスでの登録」か,「ベース修飾指定の形式での登録」かを示す 1bit のフラグと，ベースアドレス格納場所を示す領域が各エントリに必要となる．

3.2 ベースアドレス検出機構

ベースアドレスを検出するには，関数内での入力の扱われ方—アドレスとして用いられるのか，処理結果に影響を与えるような演算のソースに用いられるのか—を判別する必要がある．これは，関数内でのオペランド間の依存関係を解析することで可能となる．ここでいう依存関係とは，ある命令のディスティネーションがソースに依存するということを指す．関数のコードを解析すると，オペランドをノード，依存関係をエッジとした依存関係木を生成できる．関数内のある時点で主記憶アクセスが発生したとき，アドレス指定に用いられたソースノードから木をルート方向に探索することで，ベースアドレスを検出することができる．

しかし，このような解析には，ソフトウェアによる静的な支援，もしくは動的に行う場合は膨大なハードウェアおよびオーバヘッドが必要になる．そこで，以下に述べるように，解析を近似的に行う動的な実装方式を提案する．ハードウェアを小規模かつ固定量のものとするために，提案する機構が検出するベースアドレスの格納場所は，関数の引数受け渡し用レジスタに限定する．よって，一致比較から排除できる入力はレジスタを介した関数の引数のみということになる．

解析は木を生成するのではなく，各レジスタに対してベースレジスタ候補を記憶させる方式をとる．関数実行中のある時点で，各レジスタの値は 0 個以上の引数と依存関係にあるが，その中で，依存関係のパスがオフセット修飾する為に使われる命令 (add sub mv) のみによって生成されている一つを，ベースアドレス候補とし，その格納レジスタをベースレジスタ候補と定義する．あるディスティネーションレジスタの値がソースレジスタの値に依存するとき，ディスティネーションレジスタのベースレジスタ候補はソースレジスタのベースレジスタ候補を継承する．このようにすることで，主記憶読み込み発生時にベースアドレスが存在するならば，瞬時にこれを検出できる．

3.3 ハードェアモデル

図 2 に提案するベースアドレス検出機構の概要を示す．なお，命令デコーダからの制御信号等は省略している．

Value or Address Table(以下，V/A-Tbl) は関数の引数受け渡し用レジスタ数分の 1bit メモリセルで構成される．各メモリセルの内容は対応するレジスタの

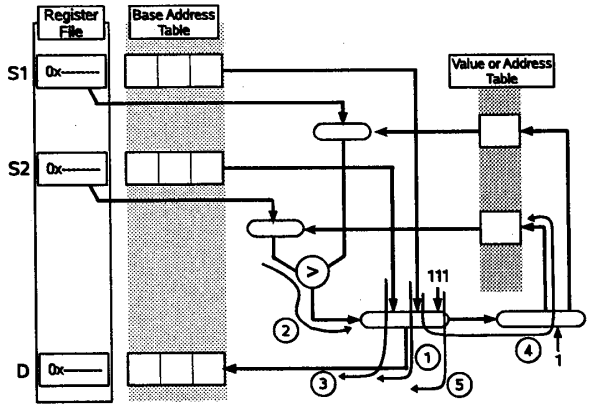

図 2 ベースアドレス検出機構

```
fnc :
      mv     r0   r2
      add    r1   r2   r3
      ld     [r3] r0
      ret
※ r0～r3 は引数受け渡し用レジスタ
```

図 3 プログラム例

関数実行開始時の値 (つまり引数) が「1:関数の処理結果に影響を与えた」か「0:ベースアドレスとして用いられた，もしくは不定」を表す．関数実行開始直後は引数の扱われ方は「不定」であるため，当然各メモリセルの初期値は 0 である．あるレジスタが，ベースアドレスにオフセット修飾する為に使われる命令 (add sub mv) 以外の命令のソースオペランドに指定されるのを検出すると，そのレジスタのベースレジスタ候補でインデックスされるメモリセルは 1 にセットされる．

Base Address Table(以下，BA-Tbl) は，N をレジスタ数とすると，$\log_2(N+1)$bit のメモリセルをレジスタの数だけ用意した表である．各エントリは，対応するレジスタのベースレジスタ候補を保持する．候補がない場合，全 bit は 1 である．各エントリが対応するレジスタがディスティネーションに指定される毎に，候補は更新される．候補の更新は，ソースレジスタの候補を継承する形で行われる．

図 3 に示すプログラムを例に検出機構の動作を説明する．また，その時のレジスタ，V/A-Tbl，BA-Tbl の様子を**図 4** に示す．

まず，mv 命令の実行で r2 は r0 に依存することになる．そこで，r2 が保持する値のベースレジスタ候補は r0 ということになり (図 2 中①)，r2 に対応する BA-Tbl のエントリは 000 となる．

次に，add 命令の実行で r3 は r1, r2 に依存することになる．ここで，ベースレジスタ候補はより大きい値を持つレジスタのものを継承することにしている．r1 と r2 の比較により，より大きい値を持つ r2 のベースレジスタ候補が選ばれ，(図 2 中②)，r3 に対応する

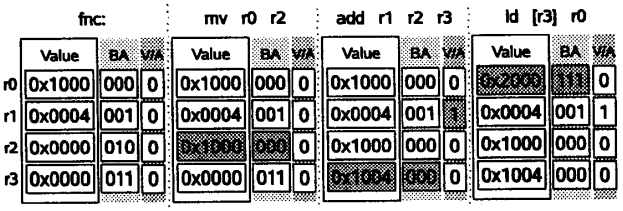

図 4 ベースアドレス検出の様子

BA-Tbl のエントリは 000 となる (図 2 中③)．また，以降 r1 は「関数の処理結果に影響を与える値」として取り扱う必要があり，対応する V/A-Tbl のエントリを 1 にする (図 2 中④)．

そして，ld 命令の実行により r3 は主記憶読み出しのアドレスとして用いられる．この時，r3 に対応する BA-Tbl のエントリは 000 を指しており，この主記憶アクセスは r0 を介した引数をベースアドレスとしたものであると解釈できる．また，r0 は読み出されてきた値で上書きされるので，対応する BA-Tbl のエントリを 111(ベースレジスタ候補無) とする (図 2 中⑤)．

最後に，ret 命令により関数実行の終了が検出されると，V/A-Tbl のチェックが行われる．r0 に対応する V/A-Tbl のエントリは 0 であり，これは r0 が保持していた引数は関数の処理結果に影響を与えておらず，主記憶読み込み時のアドレス指定としてだけ用いられたことを意味する．よってこれを関数の入力から排除できることが保証される．

4. 評　　価

以上で述べた提案手法を自動メモ化プロセッサシミュレータに追加実装し，評価を行った．シミュレータは単命令発行の SPARC-V8[3] をベースとしている．評価プログラムに汎用 GA ソフトウェアである GENEsYs を用いた．自動メモ化プロセッサと GENEsYs のパラメータを，それぞれ**表 1**，**表 2** に示す．

入力が一致しているか比較する際のコストについて説明する．引数受け渡し用のレジスタの内容との一致比較は 1cycle 要するとする．主記憶上の 32byte との比較にかかるサイクル数は，再利用表上で参照アドレスが固定番地で表されている時とベース修飾指定の形式で表されている時で異なる．固定番地である場合，L1 Cache のレイテンシと同様の 2 cycles とする．ベース修飾形式である場合は，実際の参照アドレスを計算により求める時間を考慮して 3 cycles とする．

評価結果を**図 5** に示す．横軸は評価関数名，縦軸は実行サイクル数を示している．一つの評価関数内の 3 本のグラフは左から，メモ化無しの通常実行 (N)，従来のメモ化による実行 (M)，提案手法によるメモ化による実行 (A) を示している．なお，実行時間は (N) を 1 に正規化してある．

評価関数 f11 において最大約 34%，また，従来手法よりわずかに実行サイクル数が増えてしまっている評価関数があるものの，平均でも約 2.7%の実行サイ

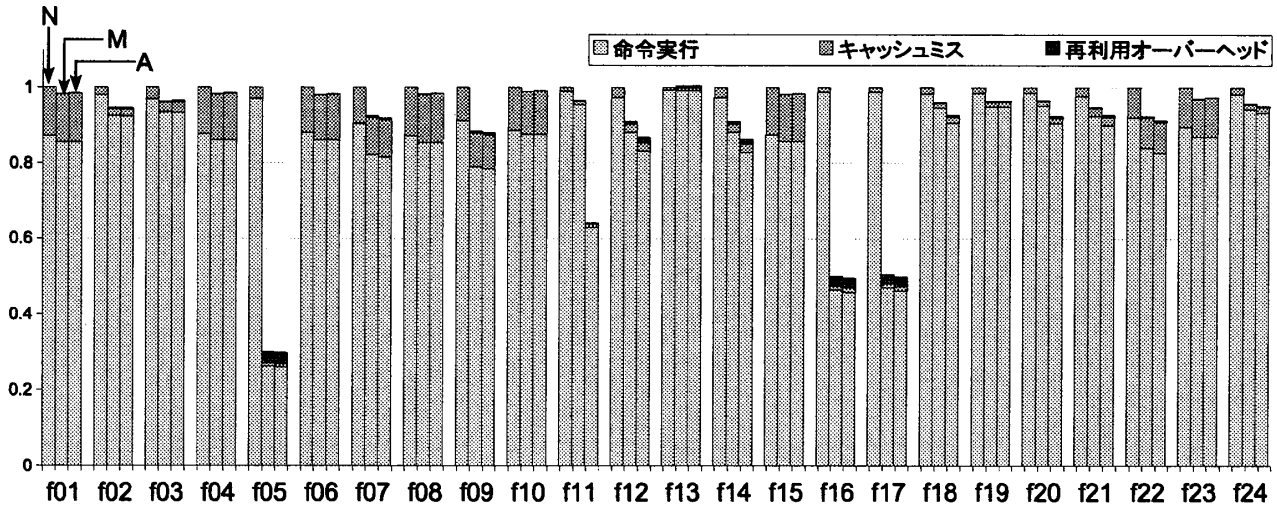

図 5 GENEsYs による評価

表 1 自動メモ化プロセッサに関するパラメータ

L1 Cache 容量	32 KBytes
ラインサイズ	32 Bytes
ウェイ数	4
レイテンシ	2 cycles
ミスペナルティ	10 cycles
L2 Cache 容量	2 MBytes
ウェイ数	4
レイテンシ	10 cycles
ミスペナルティ	100 cycles
ロードレイテンシ	2 cycles
Register Window 数	4 sets
Window ミスペナルティ	20 cycles
整数乗算レイテンシ	8 cycles
整数除算レイテンシ	70 cycles
浮動小数点加減乗算レイテンシ	4 cycles
単精度浮動小数点除算レイテンシ	16 cycles
倍精度浮動小数点除算レイテンシ	19 cycles
CAM サイズ	132 Kbytes

表 2 GENEsYs のパラメータ

交叉率	60 %
突然変異率	0.1 %
個体数	50
世代数	25 世代

クル数の削減ができおり，提案手法の有効性を証明できた．ここで，一部の評価関数で実行サイクル数がわずかに増えてしまっている理由を説明する．提案手法では，ハードウェアによる動的な依存解析を行うために幾分解析の精度を犠牲にしている．そのため，提案手法によるメモ化は関数内で当該関数の局所変数外への書き込みが行われると，本来再利用不可能なものを可能と判断してしまう場合がある．これを回避するために，主記憶への書き込みを検出した際，MemoBuf 内の当該関数の入出力の登録情報をフラッシュし，次に同関数が呼び出された時は従来手法によるメモ化を行うこととしている．関数は入れ子状に呼び出されるが，この時，最も内側の関数内で主記憶への書き込みが行われていると，提案手法によるメモ化は行えなくなる．いくつかの評価関数はこの条件に当てはまってしまっているが，メモ化が一回失敗するだけで後は通常のメモ化が行われるので，実行サイクル数の増加はごく微少である．

5. おわりに

計算再利用技術の 1 モデルである自動メモ化プロセッサにおける入力の一致比較対象から，関数の処理結果に影響を与えないものを削減する手法を提案した．シミュレーションによる評価を行い，GENEsYs で従来手法に比べて最大約 34%，平均約 2.7%のサイクル数の削減を確認した．

今後の課題として，本稿で取り上げたようなアドレス情報以外に，関数の処理結果に影響を与えない入力の調査と，それを検出，削減する機構の検討が挙げられる．

謝辞 本研究の一部は，文部科学省科学研究費補助金（萌芽研究 18650005，若手研究 (B) 19700041），(財) 栢森情報科学振興財団研究助成金，(財) 堀情報科学振興財団研究助成金による．

参考文献

1) Tsumura, T., Suzuki, I., Ikeuchi, Y., Matsuo, H., Nakashima, H. and Nakashima, Y.: Design and Evaluation of an Auto-Memoization Processor, *Proc. of Parallel and Distributed Computing and Networks*, pp.245–250 (2007).
2) MOSAID Technologies Inc.: *Feature Sheet: MOSAID Class-IC DC18288*, 1.3 edition (2003).
3) Sun Microsystems: *UltraSPARC User's Manual* (1997).