# 自動メモ化プロセッサにおける検索コスト削減のためのヒント命令挿入手法

津村 高範*，佐藤 裕貴，津邑 公暁（名古屋工業大学）

Hinting for Auto-Memoization Processor based on Static Analysis

Takanori Tsumura, Yuuki Sato, Tomoaki Tsumura (Nagoya Institute of Technology)

## 1. はじめに

プロセッサの高速化手法として，プログラムの並列性に着目した手法が注目されている．一方，これとは全く異なる概念である計算再利用と呼ばれる高速化手法がある．本研究では，この計算再利用の実行モデルの 1 つである**自動メモ化プロセッサ**[(1)]において，プログラムの静的解析情報を用いることで，計算再利用を適用する際に発生する検索コスト削減を図る．

## 2. 自動メモ化プロセッサ

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の対象区間と見なし，実行時にその入力と出力の組を再利用表と呼ばれる表へ登録する．なお，読み出しが先行したレジスタやアドレスを入力，書き込みが発生したレジスタやアドレスを出力として記憶する．その後，再び同じ再利用対象区間に侵入する際，再利用表を参照し現在の入力の組と過去の入力の組を比較する．もし，現在の入力の組が再利用表上のいずれかの入力の組と一致する場合，その入力の組に対応する出力の組をレジスタやキャッシュに書き戻すことで，再利用対象区間の実行を省略する．しかし，再利用を適用する際，再利用表に対するアクセスに伴いオーバヘッドが発生する．このオーバヘッドが計算再利用によって得られる利得よりも大きい場合，かえって性能が悪化してしまう．そこで，自動メモ化プロセッサは再利用を適用することで性能が悪化するか否かを判定する，オーバヘッドフィルタと呼ばれる機構を備えている．もし，再利用を適用することで性能が悪化すると判定された場合，その命令区間に関するエントリの再利用表への登録および検索を中止する．

## 3. 提案

自動メモ化プロセッサは入力一致比較の際，アドレスの指す値が一度も書き換えられていない場合，そのアドレスに対応する入力の一致比較を省略する．しかし，レジスタに対する一致比較は省略しない．なぜなら，メモリに比べてレジスタはその値が頻繁に書き換わることから，レジスタに書き込みが発生したか否かを検出する処理量が膨大になってしまうからである．しかし，ループ内で値が変化しないレジスタ値を入力とするループも存在する．このようなループを事前に検出できれば，その値が変化しない入力レジスタに対する一致比較を省略することができる．そこで，静的解析によりループ内で値が変化しない入力レジスタを検出し，その入力レジスタに対する一致比較を省略する手法を提案する．なお本稿では，解析情報を自動メモ化プロセッサが利用する方法として，解析情報を含んだヒント命令を実行バイナリに付加する方法を用いる．この手法により，入力の一致比較時に要する再利用表の検索オーバヘッドを削減できる．また，オーバヘッドフィルタ機構により再利用表への登録および再利用の適用が中止されていたループでも，検索オーバヘッドが削減されることで，再利用が適用されるようになる可能性もある．

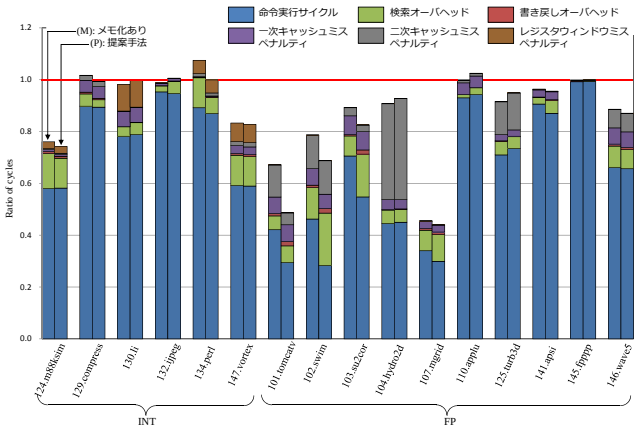


Fig.1　evaluation result

## 4. 評価

評価対象のプログラムには，汎用ベンチマークプログラムである SPEC CPU95 を用いた．評価結果を図 1 に示す．結果は，メモ化無し (baseline)，従来モデル (M)，提案モデル (P) を示しており，メモ化無し (baseline) を 1 として正規化している．まず，124.m88ksim 等，いくつかのプログラムでは，検索オーバヘッドが削減できている．これは，提案手法により不要な一致比較を削減できたためである．一方，101.tomcatv 等，いくつかのプログラムでは，命令実行サイクル数が削減できている．これは，提案手法により検索オーバヘッドが削減されることで，オーバヘッドフィルタ機構により再利用の適用が中止されていたループに対して再利用を適用できるようになったためである．結果をまとめると，メモ化無し (baseline) と比較し，従来モデル (M) ではサイクル削減率が平均 13.3 %，最大 54.6 %であったのに対し，提案モデル (P) では，平均 16.5 %，最大 56.1 %のサイクル数を削減できた．

### 文　献

(1) Tsumura, T., et.al.: Design and Evaluation of an Auto-Memoization Processor, *Proc. Parallel and Distributed Computing and Networks*, pp. 245–250 (2007)